LISPマシン システムの研究

瀧, 和男

(Issue Date)
1979-02

(Resource Type)
master thesis

(Version)
Version of Record

(URL)
https://hdl.handle.net/20.500.14094/90001857

ＬＩＳＰマシン システムの研究

昭和54年２月

神戸大学大学院工学研究科
システム工学専攻

瀧　　和　男

昭和53年度

修士論文

題目

LISPマシン システムの研究

神戸大学 大学院
工学研究科 システム工学専攻
瀧 和男

Abstract

As the application fields of computers are developed, the non-numerical computation by a computer becomes very important. Especially the LISP language is used widely in the artificial intelligence researchers. However LISP processing systems have several important problems to be solved. One of the most important problems is that the computing time for getting the solution is much slower than the case of numerical problems. The main reason for the slow speed of computation is due to a mismach of the LISP language and a conventional computer architecture at present. One of the approach to solve this mismach is to design a machine whose architecture is LISP language oriented.

This paper presents an experimental LISP machine which is specially designed powerful enough to implement an efficient LISP interpreter in microcodes. The LISP processor module and a main memory module are connected to a minicomputer whose roles are I/O processing and controls of the system. The LISP processor module consists of an arithmetic and logic unit, a computer control unit, a high speed hardware stack, a mapping memory, a field extractor, and others. The micro-instruction code is made up into a horizontal type to enable the concurrent operation of the hardwares and the recursive call at microprogram level can be used. The interpreter is designed by placing the focuses on improvements of control structures in the EVAL and the APPLY functions, and on effective utilization of specially installed hardwares.

Measurements of dynamic performances of the machine were made during executions of test programs which are prepared for the LISP contest of IPSJ. The experimental system has been found that the execution times of the test programs of this system are shorter than those of any LISP systems which joined the contest.

内容梗概

ＬＩＳＰは人工知能向き言語と呼ばれ、多くの特徴を有しているが、もともとインタープリタ向きの言語であり、実行速度は遅い。その主な原因は、言語の制御構造と、従来の計算機のハードウェア構造とがよく適合していないためであり、ＬＩＳＰの高速処理には、ＬＩＳＰ処理向きの構造を持つＬＩＳＰマシンが必要だといわれている。本論文では、そのようなＬＩＳＰマシンの開発を目的として、インタープリタのマイクロプログラム化と市販ＬＳＩの利用を前提とし、ソフトウェアからの要求を極力ハードウェアに盛りこんだシステムの設計をしている。そして、システム構成は、ミニコンピュータを入出力とバックアップに用いた計算機複合体としており、ＬＩＳＰ処理の中心をなすプロセッサモジュールには、ハードウェアスタックをはじめとする数々の高速化のための機能を取り入れている。またインタープリタについても制御構造の改良等を行ってステップ数を減少させている。また完成したシステムにおいてＬＩＳＰの例題プログラムを実行させ、それが超大型計算機上のＬＩＳＰよりも高速であることを示している。さらに、プログラム実行時のハードウェアの利用状況を測定し、それぞれのハードウェアが高速化に貢献し、とりわけ高度な並列処理とパイプライン処理が高速化に寄与していることを示している。また、システムの問題点とより大きなシステムへの適応性を示し、最後に試作システムの速度が、コンパイラの速度に近づいていることを示している。

目　　次

第1章　緒　論

ＬＩＳＰ言語は、記号処理や人工知能の研究に適した言語で、プログラムとデータの区別がないことやアルゴリズムの再帰的定義を許すこと、処理アルゴリズムがＬＩＳＰ自身で簡潔に定義できること等の特徴を有し、応用面だけでなく言語自体に対しても理論的な興味がそそられるものである。

ＬＩＳＰシステムが計算機上で動き出したのは、1960年代の初めであり、その後ソフトウェア上の多くの改良が加えられ、実用的なシステム[13][14][15][16]が育っていったが、ＬＩＳＰは本来インタープリタ向きの言語で、実行時の処理負担が大きく、ソフトウェアの改良だけでは処理速度の向上に限界があった。その主な原因として、ＬＩＳＰに特徴的なスタック操作や動的記憶領域の割り付けその他が、従来から存在する計算機システムのハードウェア構成に適合しないことで、ＬＩＳＰ処理向きハードウェアを備えたＬＩＳＰマシンの必要性が論じられるようになり、いくつかの研究成果が上げられている。[6][10][11][17]　また、大規模なプログラムを走らせるために、ミニコンピュータに補助記憶を付けたようなシステムも開発されている[12]が、本格的なＬＩＳＰマシンとして稼動しているシステムは、１例[20]を除いて他にない。

本研究は、ＬＩＳＰを高速で処理できるハードウェア、ファームウェア構造をもつ計算機システムの開発を主題として、インタープリタのマイクロプログラム化、市販ＬＳＩの使用を前提に、高速の中規模ＬＩＳＰマシンシステムを製作するものである。そうして、採用したハードウェア、

ファームウェアに評価を加え、それらのあるべき姿を追求し、ＬＩＳＰマシンシステムの問題点と発展すべき方向を見極め、合わせて新しいアーキテクチャを持つ計算機システムに先鞭をつけようとするものである。

第2章　ＬＩＳＰマシンシステムの設計

Ⅱ．1　緒　言

ＬＩＳＰ言語は、記号処理や人工知能の研究に適した言語で、他のプログラミング言語にはない多くの特徴を有している。このＬＩＳＰを処理するシステムは、1960年代の初めに登場して以来、ソフトウェア上の多くの改良が加えられてきたが、近年になって、ＬＩＳＰ処理専用の計算機システムである、ＬＩＳＰマシンの研究が行われるようになった。

本章では、ＬＩＳＰ言語の特徴を示すとともにＬＩＳＰマシンシステムに要求される機能を挙げ、本研究におけるＬＩＳＰマシンシステムの設計方針を明らかにする。さらにそのもとで決定された、ハードウェア、ソフトウェア構成を示し、本ＬＩＳＰマシンシステムの全体像を明らかにしてゆく。

Ⅱ．2　ＬＩＳＰとＬＩＳＰマシン

ＬＩＳＰはJ. McCarthy（M.I.T.）によって1960年に発表されたプログラミング言語で、リスト処理、記号処理などに適し、とりわけ人工知能、言語理論、ゲーム理論、グラフ理論などに現われる組み合わせ理論的探索問題を解くのに適している。ＬＩＳＰ言語の特徴は、第一にプログラムとデータの区別がないこと、第二にデータ構造が二進木リスト構造を基本とする簡単な形であること、第三にアルゴリズムの再帰的定義を許していること、第四にデータ型

の宣言等が不要であること、第五に処理アルゴリズムがＬＩＳＰ自身で簡潔かつ厳密に定義できることなどである。[8][9]　これらの特徴により、ＬＩＳＰでは複雑な階層構造を有するアルゴリズムを極めて簡潔に表現することが可能である。しかしながらその反面、関数の再帰呼び出しのためにスタックを大幅に必要としたり、動的記憶領域の割り付けやデータ型判別などの実行時の処理負担が大きく処理に時間がかかったり、リスト表現を用いるために単位領域当たりに格納できるデータ量が少ないといった欠点を持っている。

ＬＩＳＰがコンピュータ・システムとして実現されたのは、1962年、J. McCarthyらによるＬＩＳＰ1.5[23]が最初であり、その後ＬＩＳＰ1.5をもとにして、前述の欠点を補うべく多くの改良が加えられ、ＭＡＣＬＩＳＰ、ＩＮＴＥＲＬＩＳＰ、EPICS-LISP1.9、HLISPなどのすぐれたシステムが作られてきた。[13][14][15][16]　そこでは主に処理アルゴリズムの改良と、データ表現方法の改良による処理の高速化、記憶領域の有効利用が図られてきた。[9][13]　ＬＩＳＰ処理システムの改良とＬＩＳＰ応用研究の進む中で、従来はＴＳＳを備えた大型計算機でしか利用できなかった強力なＬＩＳＰを、より身近なところでより安価に実現し、さらにより大規模なプログラムをより高速に処理したいという要求が増し、そこからＬＩＳＰ処理専用の計算機システム、すなわち、ＬＩＳＰマシンの研究へと発展するに至っている。[6][17]

ＬＩＳＰは本来インタープリタ向きの言語であり、スタック操作やデータ型の判別、リストの探索など実行時の処理負担が大きく、それらはいずれも、従来の計算機上では

効率よく処理できない。それは、現在の汎用計算機の命令体系とＬＩＳＰの制御構造が大きくかけ離れているためであり、ＬＩＳＰが遅いといわれた最大の原因である。そこで、ＬＩＳＰマシンの開発に当たっては、命令体系の整理だけでなく、ハードウェア構造そのものをＬＩＳＰ向きに改めることが必要であるといわれている。[5][6] LISPマシンに要求される機能を列挙してみると、総合的に見た場合には、

(1) ＬＩＳＰ処理に適したハードウェア構造をもつこと。
(2) 大容量のメモリを制御できること。
(3) コンパクトなデータ表現が可能なこと。
(4) コンパイルされたコードを効率よく実行できること。
(5) 強力な入出力機能を有すること。
(6) 強力なデバッグ機能を有すること。

などが考えられ、また特にハードウェアに着目した場合には、

i) スタック操作が容易でありかつ高速であること。
ii) フィールド抽出機能、ビット処理機能を有すること。
iii) 強力な条件ジャンプ機能を有すること。
iv) メモリアクセスが高速に行えること。

などがあげられる。

Ⅱ.3 ＬＩＳＰマシンシステムの設計方針

近年の半導体技術の進歩にはめざましいものがあり、大容量の半導体ＩＣメモリや、バイポーラＬＳＩ技術によるビットストライスマイクロプロセッサエレメントなどが、つぎつぎに利用可能になっている。このことは、大きな設

備や高度な生産技術を持たない研究室等においても、LISPマシンのような専用計算機システムの研究や試作が可能になったことを意味している。

本LISPマシンシステムにおいても、このようなIC、LSIを積極的に利用し、研究室レベルで実現可能なところにシステムの規模を設定して研究を開始した。以下に、本システムの設計方針の要点を述べる。

(1) ビットスライスプロセッサエレメントと大容量半導体ICメモリの使用。

(2) マイクロプログラム制御方式の採用。

(3) LISPに適するハードウェア構成とするため、インタープリタの基本設計を先行させ、ソフトウェアからの要求をハードウェアに反映させること。

(4) インタープリタをマイクロプログラム化したインタープリタオリエンテッドなシステムとすること。

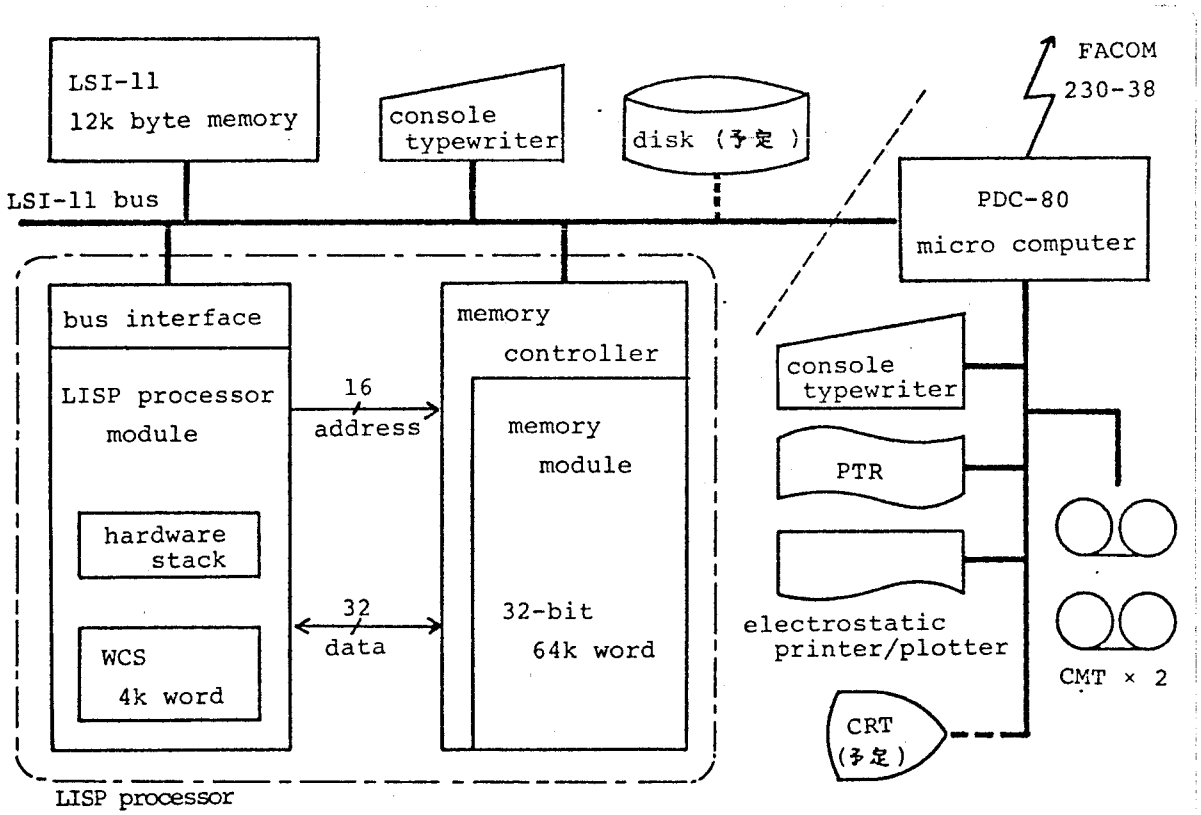
(5) システム規模を、約一年で試作の完了できる程度に制限すること。

したがって、LISPを高速で処理できるハードウェア、ファームウェア構造をもつ計算機システムの開発が本研究の中心テーマとなり、このことは、新しいアーキテクチャを持つ高級言語計算機の研究としても興味のあるものである。なお、(5)の制限から、大容量のメモリの制御は見送り、コンパイルドコードの実行と強力な入出力やファイル機能の実装は、次期の課題として見送ることにした。

Ⅱ.4 LISPマシンシステムの構成

Ⅱ.4.1 ハードウェアのシステム構成

本システムで採用したハードウェア構成を図Ⅱ・1に示す。すなわち、ＬＩＳＰプログラムの処理の中心となって働く、プロセッサモジュールとメモリモジュールを市販のＬＳＩミニコン（dec社のLSI-11）のバスに接続した構成である。ミニコンからはメモリモジュールへページモードでアクセスでき、またプロセッサモジュールのＣＭＲ（コマンドレジスタ）やＷＣＳ（writable control storage）に対しても同様にアクセス可能である。ハードウェア構成上は、ミニコンがマスターＣＰＵであり、独自の主記憶を持ち、LISPプロセッサと並行して動作できる。メモリモジュールは、２つのＣＰＵに対して共有メモリであり、ＣＭＲはミニコンに対しては、Ｉ／Ｏデバイスである。本構成は、保守やデバッグの容易さを考え、さらに既存システムとの適合性を考慮して決定したもので、前節の(5)の制限を満足するもの



図Ⅱ・1　ＬＩＳＰマシン システムのハードウェア構成

である。

ミニコンの仕事は2つあり、1つは始動時や保守時にマスターCPUとして、LISPプロセッサのプログラムロードやメモリ、レジスタの初期化を行う。もう1つは、実行時にソフトウェア上のスレーブCPUとして、LISPプロセッサからの割り込みを受け、入出力プログラムの一部と、入出力機器の制御を担当する。そのほか、時間監視や、外部からの緊急要求の受け付けも行う。また図Ⅱ・1に示すとおり、ミニコンは並列入出力インターフェースを通して、既存のマイクロコンピュータと中型計算機システム(FACOM 230-38)に接続されており、各種I/Oデバイスが利用できる。

メモリモジュールは、32ビット×64k語の構成で、高位ミニコン程度の容量があり、プロセッサモジュールは、処理巾が16ビット、アドレス空間も16ビット、メモリとのデータ受け渡しが、32ビット巾で行われる。即ち、メモリ1語中に、car部とcdr部を16ビットずつ持ち、同時に読み書きを行う。また書きこみは、バイト単位でも可能である。

Ⅱ.4.2 ソフトウェアのシステム構成

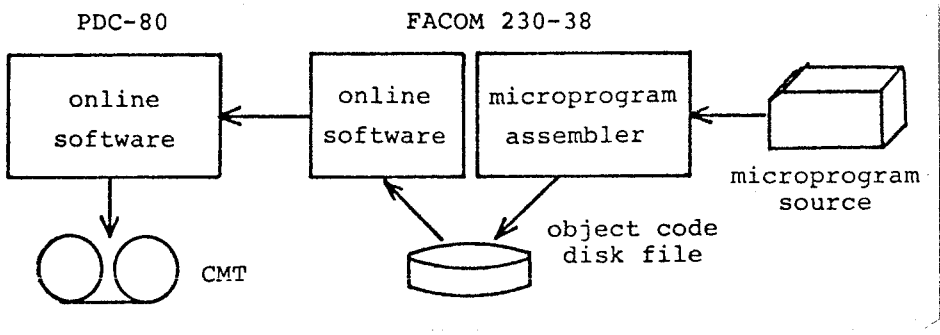
本システムのソフトウェアを分類すると3つの部分に分かれる。第一は、FACOM 230-38によるマイクロプログラムアセンブラ[4]と、そこで発生したマイクロコードを、PDC-80マイクロコンピュータのカセットMT装置まで転送する、オンラインソフトウェアである。第二は、カセットMT装置からLSI-11を経由して、LISPプロセッサのWCS、マッピングメモリ(Ⅲ.2.6参照)、主記憶へマイ

クロコードやその他のデータを転送する、イニシャルロードソフトウェアである。第三は、ＬＩＳＰプログラムの実行時に働くソフトウェア群であり、本システムの中心をなす。それらの構成を図Ⅱ・2 (a)～(c)に示す。

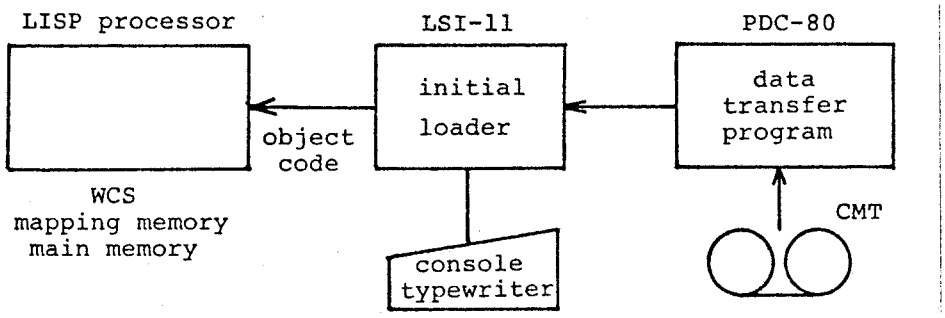
ＰＤＣ-80マイクロコンピュータのソフトウェアは、イニシャルロード時とＬＩＳＰプログラム実行時に共通の、データ転送ソフトウェアである。ＬＳＩ-11のソフトウェアは、イニシャルローダと、ＬＩＳＰプログラムの実行時に働くソフトウェアに分かれる。後者のうち、ＬＩＳＰモニタはＬＩＳＰプロセッサの初期化や実行、停止の制御機能、デバッグのための機能などを含み、ＲＥＡＤ、ＰＲＩＮＴルーチンはＬＩＳＰプロセッサからの要求により、ＲＥＡＤ．ＰＲＩＮＴ関数の処理の一部分を実行する。またタイマールーチンは1m sec毎のタイマー割込みにより、ＬＩＳＰプログラムの実行時間を計測するもので、そのほかＬＩＳＰプロセッサからの要求によって起動するタイマーの起動、停止ルーチンや、メッセージプリントルーチンなどがある。

ＬＩＳＰプログラム実行の中心をなすＬＩＳＰプロセッサのソフトウェアは、すべてマイクロプログラムで記述され、ＬＩＳＰインタープリタ、リード・プリントルーチン、組込み関数群、およびガーベージコレクションルーチンからなる。なかでもインタープリタ部分は、その制御構造とステップ数の大小により処理速度に大きな影響を与えるため、もっとも重要な部分ということができる。

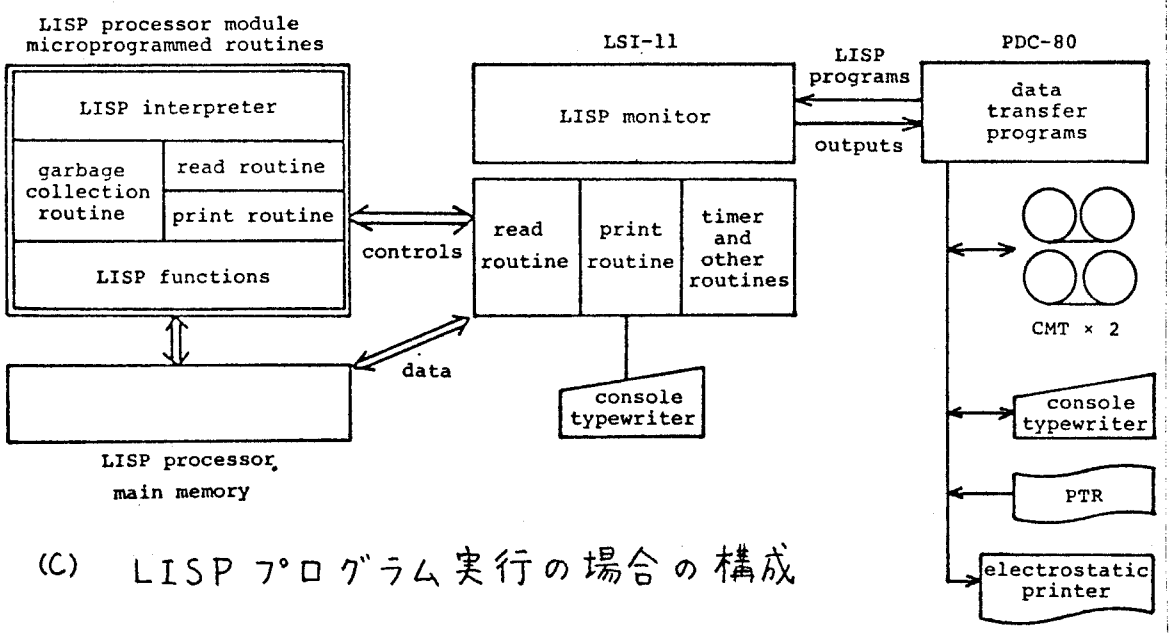
本システムは、マイクロプログラムレベルでの間接ジャンプ機能を有するため、主記憶上にマイクロプログラムのジャンプアドレスがデータとして置かれる場合がある。ま



(a) マイクロプログラムアセンブラとオブジェクトコード転送の場合の構成



(b) LISPプロセッサへのイニシャルロードの場合の構成



(C) LISPプログラム実行の場合の構成

図Ⅱ・2 LISPマシン システムのソフトウェア構成

たマッピングメモリと呼ぶ特殊なハードウェアがあり、それらの初期値設定についても、マイクロプログラムアセンブラがそのコード発生を行っている。したがって、マイクロプログラムがWCSレベルで閉じているシステムと比べて、マイクロプログラムアセンブラには、より多くの機能が必要とされる。

Ⅱ.5 結　言

複雑な階層構造を持つアルゴリズムを簡潔に表現できるLISP言語は、その反面において、実行時の処理負担が大きく、LISPマシンにはLISP処理向きのハードウェアが必要とされることを述べた。そして本LISPマシンの設計方針として、市販のLSIの使用とインタープリタのマイクロプログラム化が前提であること、ソフトウェアからの要求を極力ハードウェアに反映させることを述べた。また本研究の中心テーマが、LISPを高速で処理できるハードウェア、ファームウェア構造を持つ計算機システムの開発であることを示した。さらに、この方針のもとに決定したシステム構成を示し、本システムが、ミニコンピュータを入出力およびバックアッププロセッサとした計算機複合体であり、高位ミニコン程度のシステム規模を有することを示した。

第3章　ＬＩＳＰマシン システムのハードウェア

Ⅲ．1　緒　言

前章で述べたとおり本ＬＩＳＰマシンシステムの設計方針は、市販のＬＳＩの使用とインタープリタのマイクロプログラム化を前提として、ソフトウェアからの要求をハードウェアに取り入れ、ＬＩＳＰ処理に適した構造を持つシステムを構成することであった。本章では、この設計方針のもとで決定されたハードウェアのうち、まずその中心をなすＬＩＳＰプロセッサモジュールの構成を示し、モジュール内の各部の機能と特徴について述べる。つづいてそれらを制御するためのマイクロ命令について解説を加え、随所にＬＩＳＰ処理向きの工夫を有することを示す。さらに、ＬＩＳＰプロセッサとそのバックアップをするＬＳＩ－11の接続部分のハードウェアについて述べ、最後にハードウェア実装上の問題点と実際のハードウェアサイズを示す。

Ⅲ．2　ＬＩＳＰプロセッサのハードウェア

Ⅲ．2．1　ＬＩＳＰプロセッサモジュールの構成

ＬＩＳＰプログラムの高速処理に必要とされるハードウェア機能のうち、a．スタック操作の容易性、高速性．b．フィールド処理．ビット処理．C．条件ジャンプの多様性．などは、高級言語計算機のエミュレーター一般に、要求される機能ともいわれている。[6][10][11][18]　本ＬＩＳＰマシンは、市販のビットスライスプロセッサエレメントと、マ

イクロプログラムシーケンサＬＳＩを用いて、汎用エミュレータをめざすのではなく、インタープリタオリエンテッドなＬＩＳＰシステムの構成を目的とした。

インタープリタからの要求により、上記の３つの機能を含む、図Ⅲ・1のハードウェア構成をとった。即ち、高速かつ高機能のハードウェアスタック、フィールド抽出回路、ビットアドレッシング回路、マイクロプログラムレベルでのマルチウェイジャンプ、間接ジャンプ機能、豊富な条件テスト回路、メモリの番地による利用種別を３ビットのコードに変換するマッピングメモリ回路等である。また、マイクロプログラム用のＷＣＳは４k語を装備し、インタープリタ全体のマイクロプログラム化を行う。ハードウェアスタックは16ビット４k語の固定長である。

内部バス構成は、Ａバス、Ｂバス、Ｙバスの各16ビットの３バス構成である。Ａバスがデータのソースバスであり、Ｙバスがデスティネーションバス、またＢバスには、マイクロプログラム中の定数が与えられる。ＡＬＵでの演算は、

i) ＡＬＵ中のレジスタファイルどうし
ii) レジスタファイルと定数
iii) Ａバスデータとレジスタファイル
iv) Ａバスデータと定数

との間で、それぞれ可能である。また、マイクロプログラムシーケンサはＹバスとも結ばれていて、演算結果の番地へジャンプができる。このことにより、マルチウェイジャンプや、スタックに保存しておいた番地へのリターンが許され、マイクロプログラムレベルでの再帰呼び出しを可能にしている。また、ＡＬＵでの演算結果や、ビットアドレッシング回路からの信号は、フラグレジスタを介して、マ

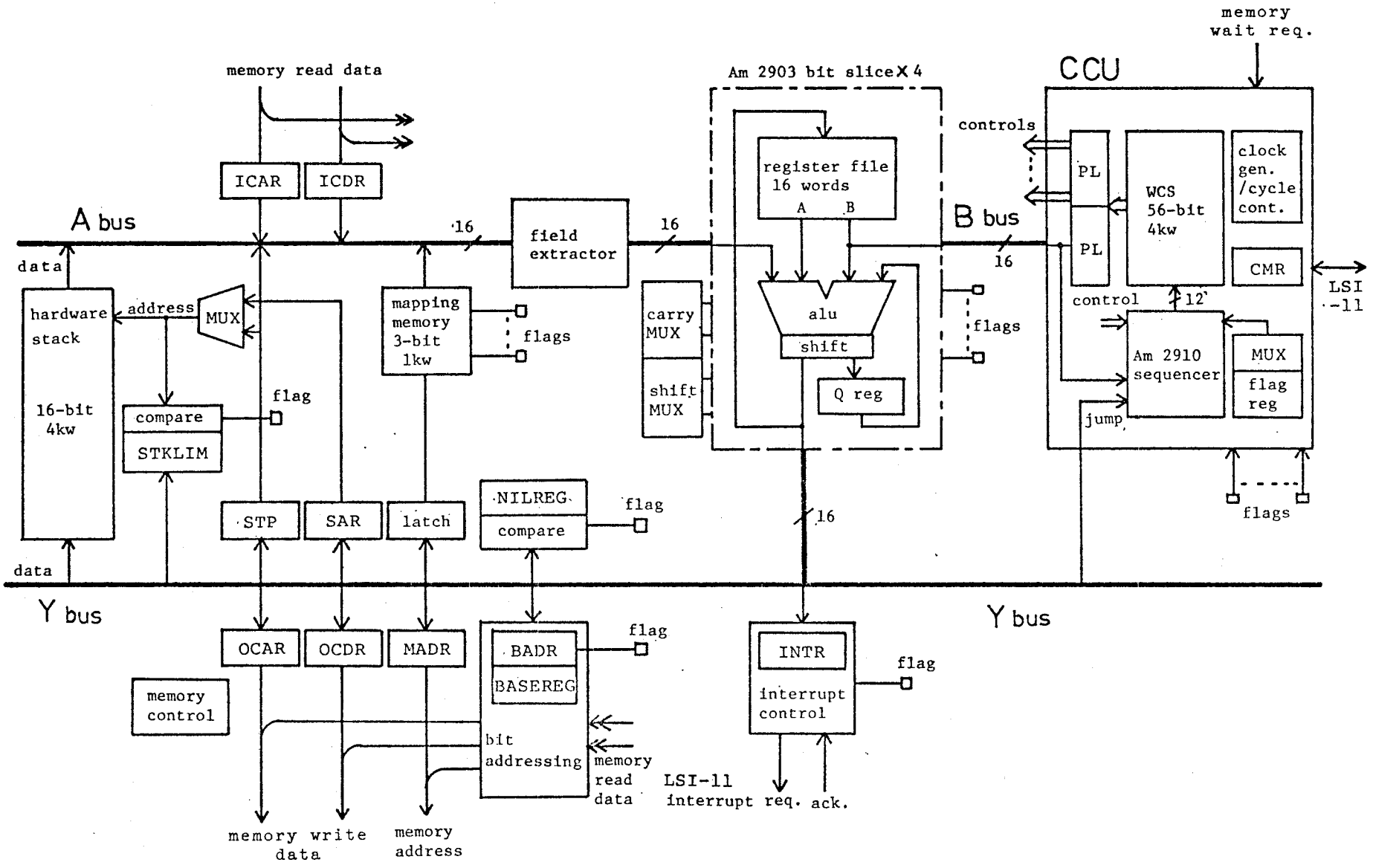


図 Ⅲ・1　プロセッサモジュールのハードウェア構成

イクロプログラムシーケンサに結ばれ、条件ジャンプに利用される。

このＬＩＳＰマシンは、再帰を含むすべての演算を、マイクロプログラムで記述し得るため、マシンコードを取り扱うためのハードウェアを特に準備していない。しかしながら、ＷＣＳの容量や、コンパイラとの関係からマシンコードを必要とすることもあり、その実現方法については、Ⅴ.3.3で述べる。

Ⅲ．２．２　ビットスライスプロセッサエレメント

ビットスライスプロセッサエレメントは、Advanced Micro Devices 社の４ビットスライスである、Am 2903[22]を、４個使用した。その特徴は、

i)　チップ上に16語のレジスタファイルを持ち、任意の２つをデータソースとして独立に指定できる。

ii)　ＡＬＵに対するソースデータの２つを、ともに外部から与えることができる（ＡバスとＢバス）

等である。ii)の機能は、ＡＬＵをレジスタファイルと切り離して利用できるために便利であり、Ａバスソースとマイクロプログラム上の定数との間の演算に利用している。最小サイクルタイムは、100n sec程度である。

zero、negative、overflow、carry のフラグ用の出力端子を備えており、条件ジャンプのテスト条件に用いている。

Ⅲ．２．３　コンピュータコントロールユニット　CCU

マイクロプログラム制御による計算機の制御部をＣＣＵ

と呼んでおり、Am 2910 マイクロプログラムシーケンサ[22]、WCS、PL（パイプラインレジスタ）、CMR、クロックジェネレータ/サイクルコントローラ、フラグレジスタから成り立っている。1レベルのパイプライン制御を行っており、ALUの実行中につぎのマイクロ命令の読出しを並行して行う。

Am 2910は、12ビット4096語のWCSを指定でき、μ-PC（マイクロプログラムカウンタ）の他に、サブルーチン用5段のスタック（ハードウェアスタックとは別）と、ループカウンタを持っている。マイクロ命令上の4ビットの指定によりオペレーションが決められるが、主なものはつぎのとおりである。

continue, conditional jump, conditional jump subroutine,
conditional return, repeat while counter≠0,
test end loop, etc.

WCSは、1語56ビットを4096語実装しており、素子はアクセスタイム150n secの4kビットMOSメモリである。

クロックジェネレータはシステムクロックを発生し、サイクルコントローラは、マイクロ命令やメモリからのwait要求により、クロックジェネレータに働きかけて、マイクロ命令の実行周期を変化させる。

CMRは、LSI-11から読み書きできる8ビットのレジスタで、LISPプロセッサの実行、停止、初期化の制御に用いられ、またattention 0,1の各ビットはフラグレジスタに結ばれていて、LSI-11からLISPプロセッサに対して、何らかの要求のあることを示す。

フラグレジスタは次のフラグを含み、すべて条件ジャンプに利用できる。

ＡＬＵ関係フラグ：

zero, negative, overflow, carry, S-shiftout, Q-shiftout

マッピングメモリ関係フラグ：

list, literal atom, number, atom

その他のフラグ：

nil, bit test, stack overflow, iack, attention 0,1

Ⅲ．２．４　ハードウェアスタック

70n secの高速メモリによる、4k語の固定長スタックである。スタックへのアクセスは、２つのスタックポインタレジスタＳＴＰ（スタックトップポインタ）とＳＡＲ（スタックアドレスレジスタ）により、ＳＡＲは主に、スタック内へのアクセスに用いる。いずれもカウンタタイプのレジスタで、スタックをソースとしたときは間接後自動減少、デスティネーションとしたときには自動増加後間接のモードを有する。これらにより、ＡＬＵとＳＴＰ、ＳＡＲ間のデータ転送の回数を減らすことができる。またＳＴＰとＳＡＲを利用して、スタックからスタックへのデータ転送を１マイクロサイクルで完了できる。ＬＩＳＰインタープリタにおけるframeの作成とリターンのとき、またSUBR関数中での引数の移動の際などに、スタックに対するデータ転送は多く、これらの機能の活用によってマイクロプログラムのステップ数を減少させることができる。

ＳＴＫＬＩＭ（スタックリミットレジスタ）にはあらかじめ値をセットしておき、この値を越えてスタックが伸びたときにはフラグが立って、スタックオーバフローを簡単に調べることができる。

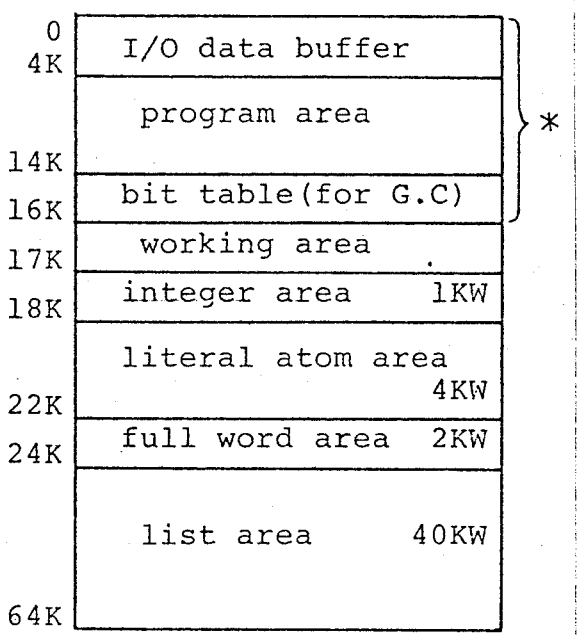
Ⅲ．2．5　フィールド抽出とマッピングメモリ

フィールド抽出回路は、ＡバスとＡＬＵの間にはいり、データのマスキングとシストを同時に行う。データの素通りを含めて４種類の固定パターンを有するにすぎないが、本ＬＩＳＰマシンは、タグマシンではなくマシンコードも規定しないため、フィールド抽出の多様性はあまり必要としない。抽出位置とビット数は、ジャンパ線により変更できる。フィールド抽出回路の実現には、８個の高速マルチプレクサＩＣを用いている。

マッピングメモリは３ビット×1k語あり、主記憶アドレスを入力として、表Ⅲ･1に示す３ビットのコードを出力する。すなわち、主記憶アドレスを64語毎に1024区画に分割し、各区画に、利用種別を表わす３ビットのタグを付

表Ⅲ･1　マッピングメモリによるコードと種別との対応

コード	種　別
0	小整数１
1	小整数２
2	小整数３
3	整　数
4	文字アトム
5	フルワード
6	フリーリスト１
7	フリーリスト２



図Ⅲ･2　メモリ割当て

けたと考えることができる。図Ⅲ・2がＬＩＳＰマシンのメモリ割当てであるが、マッピングメモリにより、アドレスから、利用種別のコードがただちに得られる。このコードやフィールド抽出回路の出力を定数（アドレス）と加算することにより、インデックスジャンプのような形のマルチウェイジャンプが実現できる。

マッピングメモリの出力は、Ａバスの他に、デコーダをとおしてフラグレジスタに接続されており、list、文字アトム、数値、アトムの判定が簡単に行える。

Ⅲ．２．６　その他のレジスタ

メモリとのデータ受け渡しは、ICAR(input car register), ICDR(input cdr register), OCAR(output car register), OCDR(output cdr register)を通して行う。ＭＡＤＲ（メモリアドレスレジスタ）にアドレスを書きこむと、メモリオペレーションは自動的に始まる。

メモリオペレーションは、readとread while writeの２種である。read while writeとは、メモリセルの内容の読み出しと同時に、別のデータの書きこみが行われるモードである。最近のダイナミックＲＡＭにおいて可能となったもので、アトムのvalue cellの書きかえに有効である。

ＬＩＳＰインタープリタにおいて、＝ＮＩＬを判定する機会は多く、演算結果がＮＩＬであることをフラグに出力する回路を設けた。ＮＩＬレジスタがそれである。

また、ＬＳＩ-11に割り込みを発生させるためのレジスタとして、ＩＮＴＲ（インタラプトレジスタ）がある。INTRにデータを書きこむと、その値がベクタアドレスとなって

LSI-11に割り込みを発生し、受け付けられると、フラグレジスタにiackフラグが立つ。

Ⅲ．2．7　ビットアドレッシング回路

主記憶上のビットテーブルに対して、ビットテストとビットセットを行う。ガーベージコレクション時に、スタックを用いたマーキングを行うので、1語（リストエレメント）に対し1ビットのマークビットを用意すると、64kビット、すなわち2k語のビットテーブルとなる。ガーベージコレクションルーチンでは、テーブルをすべてクリアしたあと、BASEREG（ベースレジスタ）にテーブルの先頭アドレスを書き込む。以後マーキングアルゴリズムに移り、たとえば8000(16)番地のリストエレメントに対し、マークの有無が知りたければ、BADR（ビットアドレスレジスタ）に、TEST指定で8000(16)を書きこむ。するとメモリ参照が起こり、2つあとのマイクロ命令サイクルで、フラグレジスタにマークの有無が現れる。ビットセットの場合にも同様に、SETの指定でBADRに書きこめば、対応ビットがセットされる。こうして、ビットテーブルへのアクセスが簡単になり、ガーベージコレクションが高速化される。

なおBASEREGに書きこむことのできるアドレスは、2k語境界（下位11ビットがすべてゼロ）の値のみである。

Ⅲ．2．8　メモリモジュール

メモリモジュールは、メモリバスを中心にして次に示す

4つのユニットが接続された構造である。

i) メモリコントローラ

ii) メモリセルユニット

iii) LSI-11インタフェース

iv) LISPプロセッサモジュールインタフェース

iv)の中には、図Ⅲ·1に示したMADR、OCAR、OCDR、ICAR、ICDR、BASEREG、BADRなどのレジスタ群が含まれる。またメモリコントローラには、メモリサイクルのタイミング発生回路、メモリバス使用権の先着優先回路、リフレッシュ制御回路が含まれている。

本メモリモジュールの特徴は、先着優先回路の働きでメモリ参照がまったく非同期的に行えることで、もしメモリ参照要求が重なると、あとから要求を出した方が待たされることになる。メモリ参照要求を出す可能性のあるのは、LISPプロセッサモジュール、LSI-11、リフレッシュ制御回路の三者である。メモリ参照が非同期的に行えることは、高速で動作しているマイクロ命令コントロールサイクルに合わせて、メモリ参照が可能になるということで、LISPの高速化に役立つことが期待できる。

Ⅲ.3 マイクロ命令とコントロールサイクル

Ⅲ.3.1 マイクロ命令フォーマット

ビットスライスプロセッサエレメント、マイクロプログラムシーケンサ、外付けのスタックやレジスタ類を制御するために、図Ⅲ·3に示す56ビットのマイクロコードを使用する。

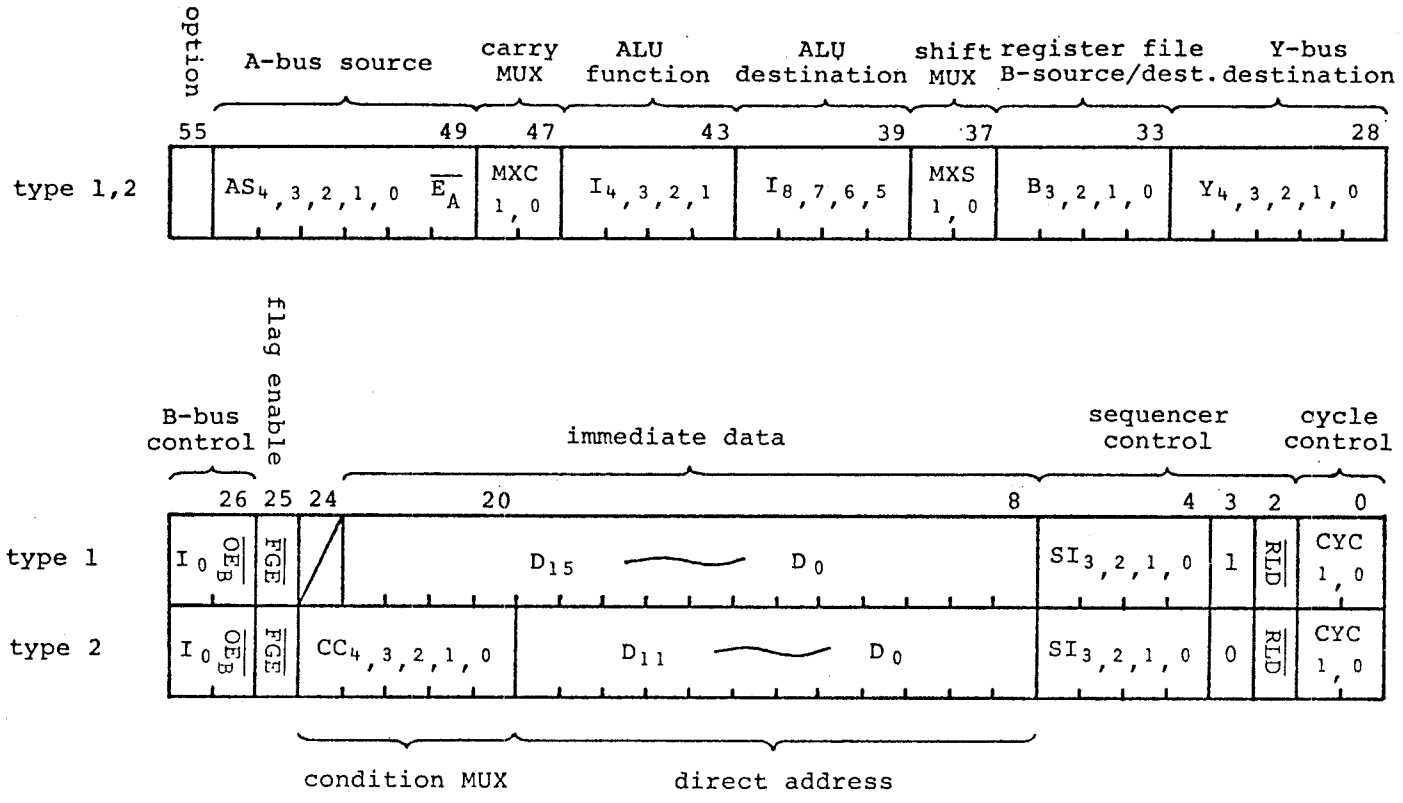


図 Ⅲ・3　マイクロ命令フォーマット

マイクロ命令は2つのタイプに分かれ、ビット3が1のときをタイプ1、ビット3が0のときをタイプ2と呼ぶ。タイプ1はALUに対して定数を与えることのできるタイプ、タイプ2はフラグをテストして条件ジャンプのできるタイプである。ALU動作については、いずれのタイプでも同様に指定でき、条件ジャンプをしながら並行してALU動作が行えるといった特徴を有する。以下にフィールドごとについての説明を簡単に加える。

ビット49～54はAバスソース指定、ビット28～32はYバスデスティネーション指定である。詳細をそれぞれ表Ⅲ·2、Ⅲ·3に示す。スタックアクセスに対するモード指定は、これらのフィールドにて行う。タイプ2のビット20～24は、ジャンプ条件の指定であり、Ⅲ.2.3節に示したフラグがジャンプ条件として、指定可能である。ビット39～46は、Am 2903のオペレーション指定、ビット37. 38. 47. 48はシフトマルチプレクサとキャリー入力に対する指定、ビット25はフラグレジスタのenable指定、ビット2～7はAm 2910に対するオペレーション指定である。[22]

Ⅲ．3．2　マイクロ命令コントロールサイクル

マイクロ命令の実行サイクルは、基本的に300n secであり、またALUおよびシーケンサLSIは、1相クロックで制御されるように設計されているが、システムクロックとしては、図Ⅲ·4に示す4相クロックを用いている。これは主にスタックコントロールに用いられる。AバスソースやYバスデスティネーションにスタックを指定した場合、$\overline{\phi_0}$と$\overline{\phi_1}$がスタックからの読出しフェーズ、$\overline{\phi_3}$がスタックへ

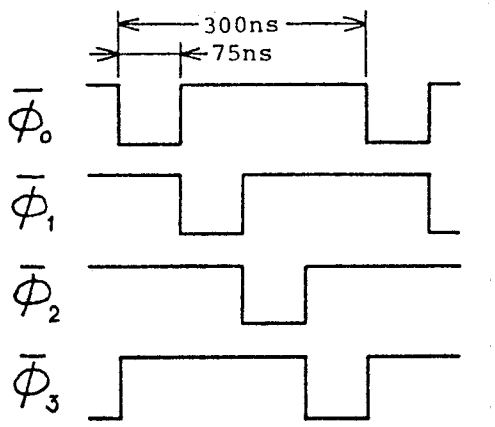


図 Ⅲ・4 システムクロック

表 Ⅲ・4 マイクロコードによる $\overline{\Phi}_1$ のコントロール.

CYC_1	CYC_0	$\overline{\Phi}_1$ のパルス巾 (ns)
0	0	75
0	1	150
1	0	225
1	1	wait 要求がなくなるまで

$AS_{4,3,2,1,0}$	$\overline{E_A}$	A bus source	
X	A_3 A_2 A_1 A_0	0	register file	1)
0	0	0 AF_1 AF_0	1	ICAR	2)
0	0	1 AF_1 AF_0	1	ICDR	
0	1	0 AF_1 AF_0	1	STP	
0	1	1 AF_1 AF_0	1	MAPPING MEMORY	
1	0	0 AF_1 AF_0	1	(STP)	3)
1	0	1 AF_1 AF_0	1	(STP)-	4)
1	1	0 AF_1 AF_0	1	(SAR)	
1	1	1 AF_1 AF_0	1	(SAR)-	

表 Ⅲ・2 Aバスソース指定

1) A_3〜A_0 はレジスタファイルのアドレス指定
2) AF_1, AF_0 はフィールド抽出回路の指定
3) (STP) はスタックポインタによる スタックの間接アドレス指定
4) (STP)- は間接後スタックポインタを自動減少させる指定

Y_4 Y_3 Y_2 Y_1 Y_0	Y bus destination	
0 0 X X X	未使用	
0 1 0 0 0	OCAR	
0 1 0 0 1	OCDR	
0 1 0 1 0	STP	
0 1 0 1 1	SAR	
0 1 1 0 0	BASEREG	
0 1 1 0 1	STKLIM	
0 1 1 1 0	NILREG	
0 1 1 1 1	INTR	
1 0 0 0 0	MADR R	1)
1 0 0 0 1	MADR W CAR	2)
1 0 0 1 0	MADR W CDR	3)
1 0 0 1 1	MADR W BOTH	4)
1 0 1 0 X	BADR TEST	5)
1 0 1 1 X	BADR SET	6)
1 1 0 0 X	(STP)	
1 1 0 1 X	+(STP)	7)
1 1 1 0 X	(SAR)	
1 1 1 1 X	+(SAR)	

表 Ⅲ・3 Yバスデスティネーション指定

1) メモリリード
2) メモリライト, car部のみ
3) メモリライト, cdr部のみ
4) メモリライト, car, cdr両方
5) ビットテーブルのテスト
6) ビットテーブルのセット
7) スタックポインタの自動増加後間接アドレス指定

の書きこみフェーズになる。またＬＳＩへのクロックをはじめ、各レジスタに対する書きこみクロックには$\overline{\phi_3}$を用いる。

また$\overline{\phi_1}$のパルス巾は、マイクロコードのビット０、１の指定により、表Ⅲ･４のように75n secの整数倍だけ延長される。たとえば、ＡＬＵの演算結果をジャンプアドレスに使うようなときには、パルス巾の延長を行い、ＷＣＳにnext addressが与えられるのが遅れた分だけ、サイクルを延ばしてやる。またメモリオペレーション時には、wait要求がなくなるまで、$\overline{\phi_1}$パルスが延長されてメモリに同期する。レジスタ、ＡＬＵ、バス相互間の伝搬遅延と、サイクルの延長時間との関係は、付録１.に示す。

Ⅲ．３．３　メモリコントロール

メモリオペレーションは、表Ⅲ･３に示したとおり、Ｙバスデスティネーション指定によってＭＡＤＲまたはＢＡＤＲにアドレスを書きこんだ時点から開始される。ＭＡＤＲ等への書きこみは$\overline{\phi_3}$の立ち上りで行われ、ただちにメモリrequestが発せられる。次の$\overline{\phi_1}$は、メモリからのacknowledge信号が帰るまで（wait信号がなくなるまで）延長され、メモリからのリードデータは、次の$\overline{\phi_3}$の立ち上りでＩＣＡＲ.ＩＣＤＲレジスタにラッチされる。こうしてメモリオペレーションは、ＭＡＤＲまたはＢＡＤＲへの書きこみの、次のサイクル中に完了し、読み出したデータは、その次のサイクルで利用可能となる。メモリ動作が進行している間の１サイクルは、メモリに関係のあるレジスタを除いて、普通のＡＬＵオペレーションが可能であるため、メモリ参照

時のむだ時間を減少させることができる。

Ⅲ．4　ＬＳＩ－11との接続とメンテナンスパネル

Ⅲ．4．1　ＬＳＩ－11から参照できるレジスタ

ＬＩＳＰプロセッサの中にあってＬＳＩ-11から参照できるレジスタは、次の３つである。

i)　ページレジスタ

ii)　コマンドレジスタ（ＣＭＲ）

iii)　マイクロアドレスレジスタ

図Ⅲ･1では、これらのうちＣＭＲだけが示されていたが、以下順に機能説明を行う。

ページレジスタは８ビットの読み書き可能なレジスタで、ＬＳＩ-11からＬＩＳＰプロセッサの主記憶、ＷＣＳ、マッピングメモリのどれかにアクセスしようとするとき、対応するページを書きこむためのものである。ＬＳＩ-11のアドレス空間は、16k語（32kバイト）から24k語（48kバイト）までの部分が多重化されており、ページレジスタで指定したページ（１ページが８k語）のみがアクセス可能となる。０～$0F_{(16)}$ページが主記憶、$F0$、$F1_{(16)}$ページがＷＣＳ、$F8_{(16)}$ページがマッピングメモリに対応する。

ＣＭＲはＬＩＳＰプロセッサの動作モードを規定したり、実行・停止の制御をしたり、ＬＳＩ-11からのフラグ情報を伝えたりする重要なレジスタである。各ビットのうち分けを表Ⅲ･5に示し、その機能を以下に述べる。

bit 2：　メンテナンスビット―1を書きこむことによりＬＩＳＰプロセッサはメンテナンスモードとなる。メン

表Ⅲ・5 CMRのビット割当て

bit	name	access mode	initial state
0	STEP	write only(pulsed)	0
1	INITIALIZE	write only(pulsed)	0
2	MAINTENANCE	read/write	1
3	RUN / HALT	read/write	0
4	unused	———	0
5	unused	———	0
6	INTERRUPT ENABLE	read/write	0
7	RUN FLIPFLOP	read only	0
8	ATTENTION 0	read/write	0
9	ATTENTION 1	read/write	0
10~15	unused	———	0

テナンスモードはＬＩＳＰプロセッサが停止し完全に受動状態になるモードで、ＬＳＩ-11からＷＣＳやマッピングメモリへの読み書きのために、データ経路が切りかえられた状態である。

bit3： ＲＵＮ／ＨＡＬＴビット ― マイクロ命令の実行停止を制御するビットで、１を書きこむと実行状態、０を書きこむと停止状態となる。bit２が０のときのみ有効である。実行状態とするためには、メンテナンスパネルのＲＵＮ／ＨＡＬＴ／ＳＴＥＰスイッチが、ＲＵＮ状態でなければならない。

bit0： ステップビット ― 停止状態の時に有効で、１を書きこむとマイクロ命令を１ステップだけ実行する。デバッグ時に用いる。なおメモリ参照命令の実行時のみ２ステップの実行となる。

bit1： イニシャライズビット ― 実行時、停止時に有効で、１を書きこむことにより、マイクロ命令の実行の流れを、強制的に０番地へジャンプさせる。

bit6： 割込み許可ビット ― このビットが1であると、マイクロプログラムでＩＮＴＲへ書きこみを行ったときに、ＬＳＩ-11に割込みを発生する。

bit7： ＲＵＮフリップフロップビット ― ＬＩＳＰプロセッサの状態を監視するためのビットで、通常は、bit3に等しい。メンテナンスパネルからシングルクロックでハードウェアのテストをするときのみ、マイクロ命令サイクルが終わるまで1を保持する。

bit8,9： アテンションビット ― ＬＩＳＰプロセッサのフラグレジスタに接続され、ＬＩＳＰプログラムの実行時に、ＬＳＩ-11からの要求を伝える。

マイクロアドレスレジスタは、マイクロプログラムシーケンサAm 2910の発生するマイクロ命令アドレスを、ＬＳＩ-11側から読み出すためのレジスタで、デバッグ用のブレイクポイント機能を実現するために用いる。

Ⅲ．4．2 ＬＳＩ-11のアドレス割当て

ＬＳＩ-11のハードウェアのアドレス割当ては次のとおりである。

0～77777(8)： ＬＳＩ-11の主記憶。

100000(8)～137777(8)： アドレス多重化領域で、ＬＩＳＰプロセッサの主記憶、ＷＣＳ、マッピングメモリが割当てられている。0ページの100000(8)番地がＬＩＳＰプロセッサ主記憶の0番地下位16ビットに対応し、同じく100002(8)番地が、上位16ビットに対応する。F0(16)ページの100000(8)番地から100006番地がWCS

の0番地の下位ビットから上位ビットに対応する。

140000(8) ： ページレジスタ。

140002(8) ： CMR.

140004(8) ： マイクロアドレスレジスタ。

172540(8) ： クロック割込み回路CSR.

177550(8) ～ 177556(8) ： PDC-80インタフェースレジスタ。

177560(8) ～ 177566(8) ： コンソールタイプライタインタフェースレジスタ。

Ⅲ．4．3 メンテナンスパネルの機能

LISPプロセッサモジュールには、メンテナンスパネルを用意しており、状態監視や保守デバッグに利用できる。

メンテナンスパネルは、スイッチ部と表示部に分かれ、表示部は、Aバス、Yバス、スタックアドレス、マイクロ命令アドレス、マイクロ命令コード、CMRの内容、クロックの各相を表示する。スイッチ部は、RUN/HLT/STEPの制御スイッチ、シングルクロックの制御スイッチと、LISPプロセッサの停止時に、レジスタファイルやAバスに接続されたレジスタの内容を見るためのレジスタ指定スイッチからなる。レジスタ指定の方法は、マイクロ命令のAバスソース指定フィールドと同様である。

Ⅲ．5 ハードウェアの製作

Ⅲ．5．1 ハードウェア実装上の問題点

ＬＩＳＰプロセッサモジュールをハードウェアの実装という観点からみた特徴は、比較的高速度のＩＣ、ＬＳＩが、ビットスライスプロセッサ、ＣＣＵ、ハードウェアスタック部分を中心にして相当個数使用されることである。このことから、いくつかの実装上の問題が生じたので、以下に列挙する。

まず第一は、ショットキーＴＴＬ・ＩＣの高速性に起因するものである。このＩＣはスイッチング時のスパイク雑音が、標準ＴＴＬの３倍も大きく、したがって電源のバイパスコンデンサの強化と太く短い接地ラインを必要とする。また、立上り立下り時間が短いため配線のインピーダンスがよけいに影響し、特に容量性の負荷を接続した場合には、大きな減衰振動波形を発生する。スイッチング時の雑音とこの減衰振動波形が重なり合うと、場合によりＩＣの雑音余裕度を越えて、誤動作の原因となる。対策としては、配線を短くすることと、長い配線の場合には速度を犠牲にしても標準ＴＴＬ・ＩＣを用いること、また接地ラインを強化すること等である。

第二は基板分割に関する問題である。システムが大きくなると使用ＩＣ個数が増加し、高速動作を要求すると短い配線が必要となる。そうすると基板当たりのＩＣ実装個数を増さねばならないが、それには限度があり、また基板コネクタのピン数の制限もある。許された実装方法のもとで実装密度を上げ、配線を短くし、基板間の配線本数を最小化するように、各基板に回路を分割配置することがたいへん難しくなる。本システムでは、ＷＣＳとビットスライスプロセッサまわりでほぼ限界の実装状態であり、ＷＣＳ容量を増加させたり、16ビットの処理幅を32ビットに変更し

ようとすると、実装方法自体を変更しなければならなくなると考えられる。

第三は発熱の問題である。実装密度を上げると当然のことながら、面積当たりの電力密度が上がり、自然空冷では間に合わなくなる。本システムでも特にＷＣＳの部分での電力消費が大きく、小型ファンによる空冷を行っている。

以上の問題点はいずれもシステム規模に密接に関係しており、本システムよりも１クラス上のＬＩＳＰマシンシステムを製作しようとすると、実装設計が格段に難しくなることが予想される。

Ⅲ．５．２　ハードウェアサイズ

本システムのハードウェアサイズはおおよそ次のとおりである。

まずＬＩＳＰプロセッサ部分は、17cm×22cmの万能基板（コネクタピン数は100ピン）15枚を用いた構成で、電源を含めて３段の19インチラックに収納している。そのうち分けは、プロセッサモジュールが７枚、メモリモジュールが８枚で、はんだ付けとワイヤラッピングにより配線を行っている。総ＩＣ個数は約600個で、電源容量は５Ｖ－30Ａ、12Ｖ－４Ａである。冷却には12個の小形ファンを用いている。また基板コネクタは100ピンでは不足であり、基板端にフラットケーブルコネクタを増設して多用している。

ＬＳＩ－11側では、タイマー割込み回路、ＰＤＣ－80インタフェース回路、バス拡張回路を約70個のＩＣを用いて３枚の基板上に構成し、ＬＳＩ－11の筐体内に収納してい

る。

3段の19インチラックとＬＳＩ－11は、まとめて1つのキャビネットに収納する。システム全景の写真を付録.2に示す。

Ⅲ.6 結　言

本ＬＩＳＰマシンシステムのＬＩＳＰプロセッサ部は、強力なハードウェアスタック、フィールド・ビット処理回路、マッピングメモリ、多様な条件ジャンプのための回路等、ＬＩＳＰ処理に適するハードウェアから構成されることを示した。またそれらを制御するマイクロ命令も、ALU動作と条件ジャンプの並列実行ができるなど、ＬＩＳＰ処理の速度向上に役立つ特徴をもつことを述べた。また、ＬＳＩ－11から見たＬＩＳＰプロセッサ内のレジスタの仕様、メモリ割当てを示した。またハードウェアの製作上の問題点と本システムの実装設計が限界に近いことを述べ、実際のハードウェアサイズについて触れた。

第４章　ＬＩＳＰマシンシステムのソフトウェア

Ⅳ．１　緒　言

本ＬＩＳＰマシンシステムのソフトウェアの中心となるのはマイクロプログラム化されたインタープリタであり、その制御構造とステップ数の大小が、ＬＩＳＰの処理速度に大きな影響を与えることは第２章で述べたとおりである。本章では、始めにＬＩＳＰ言語の仕様を示したあと、LISPインタープリタが走り始めるまでの手続きと、ＬＳＩ－11が関与する入出力処理について述べる。つづいて、本LISPインタープリタの特徴と高速化の要点を、ＬＩＳＰ言語の制御構造との関連において述べ、具体的なインタープリタ内における処理手順を示す。さらにＬＳＩ－11のモニタとデバッグ機能について触れ、実際に作成したソフトウェアのサイズについても言及する。

Ⅳ．２　ＬＩＳＰ言語の仕様

Ⅳ．２．１　データの種類とその表現

本処理系で現在用意しているデータ型としては、小整数、整数、文字アトム、リスト要素の４種類がある。これらの表現を図Ⅳ･１に示す。

文字アトムをＬＩＳＰ1.5流にリストを用いて表現しないで、連続した領域に情報を置くことにより、属性のチェック、属性値の取り出しを容易にした。[4][9]　また、変数値はスタック上にold-valueを退避するshallow-binding

を採用しているので、自由変数の値の取り出しについては速度の向上が期待できる。また、図Ⅲ・2で示したように主記憶上の領域をデータ型によって分割し、マッピングメモリ回路により高速に型判別を行っている。

データ型	語数	表　現
小整数	0	ポインタの値自身　$-8192 \leq n \leq 8191$
整数	1	31 … 0：2の補数表現で整数エリアに格納
文字アトム	4	31 … 16 15 … 12 11 … 0：関数本体 / 属性 / 属性別飛び先；flag / 引数個数；top-level-value；property-list；print-name；go-label-value；print-name-hash
リスト要素	1	31 … 16 15 … 0：cdr / car

図Ⅳ・1　データ表現

Ⅳ．2．2　関数型の種類

関数型は、ＥＸＰＲ、ＦＥＸＰＲと、システム関数用としてマイクロコードで記述されたMSUBR、MRSUBR、MRFSUBRを用意した。ＭＳＵＢＲだけは再帰呼出しを許さない関数型である。[4] なおマシンコードで記述されたＳＵＢＲ、ＦＳＵＢＲ型関数は、コンパイラの実装時に用意する予定である。主な関数を次に示す。

```
MSUBR  : CAR, CDR, CONS, EQ, NULL, ATOM, ADD1,  etc.
MRSUBR : EVAL, APPLY, EVLIS, EQUAL, MAPCON,  etc.
MRFSUBR: COND, PROG, AND, OR, LIST, DE, SETQ,  etc.
```

Ⅳ．3　システムの初期化と入出力処理ソフトウェア

Ⅳ．３．１　ＬＩＳＰプロセッサの初期設定

ＬＩＳＰインタープリタが動き出すまでに必要なＬＩＳＰプロセッサの初期設定としては、

i)　ＷＣＳへのマイクロプログラムの書込み。

ii)　組込み関数のための文字アトムとマッピングメモリデータの作成。

iii)　各レジスタの初期設定。

iv)　自由リストの作成と整数領域の初期化およびそれらを指すポインタの値のセット。

を上げることができる。

i)、ii)はマイクロプログラムアセンブラによりあらかじめ準備してＣＭＴに格納してあるものを、ＬＳＩ－11が読み出してそれぞれの領域へ書き込むことにより行っている。iii)、iv)はＬＩＳＰプロセッサ自身のマイクロプログラムの働きにより行われるもので、ＬＳＩ－11からＣＭＲへビット書きこみすることにより、ＬＩＳＰプロセッサが動作を開始して実行される。

Ⅳ．３．２　式の読み込みと結果のプリント

プログラムは、式formの読み込み、評価、結果のプリントのサイクルを繰り返すことにより実行される。式の評価はＬＩＳＰプロセッサが実行するが、式の読み込みと結果のプリントはＬＳＩ－11とＬＩＳＰプロセッサが相互通信を行いながら実行している。

式の読み込みはＲＥＡＤ関数によって実行される。LISPプロセッサは入力の要求が生じると、ＬＳＩ－11に対して

ＲＥＡＤ要求の割込みを発生させ、ＬＳＩ-11側でのREAD処理終了の信号を待つ。ＲＥＡＤ要求を受けたＬＳＩ-11はＣＭＴより式を読み込み、括弧記号とドットとアトムとを分離し、形式のチェックを行い、各アトムを、

i) 登録済みのアトムの場合にはそのアトムへのポインタ。

ii) 未登録のアトムの場合には新しくアトムを作成し、それへのポインタ。

iii) 小整数の場合には値自身。

のように変換し、入力ストリングを括弧、ドット、ポインタからなる入力ストリングに変換する。

変換が終了すると、ＲＥＡＤ処理終了の信号をＬＩＳＰプロセッサに送る。信号を受けたＬＩＳＰプロセッサはストリング列から二進木を作成してＥＶＡＬ処理に制御を移す。

結果のプリントはＰＲＩＮＴ関数によって実行される。その手順は、

i) 結果をプリント順に並んだ括弧、ドット、アトムへのポインタ列に変換。

ii) ＬＳＩ-11にプリント要求の割込みを出し、LSI-11からのプリント終了信号の到着を待つ。

iii) ＬＳＩ-11は、括弧、ドット、アトムへのポインタ列を出力文字ストリングに変換しながら出力を行う。

iv) プリント終了の信号をＬＳＩ-11はＬＩＳＰプロセッサに送る。

v) ＬＩＳＰプロセッサはプリント終了の信号を受けると、次の式の読み込み要求を出す。

となる。

なおＬＳＩ-11からＬＩＳＰプロセッサへ送られる処理終了合図の信号には、ＣＭＲ内のアテンション１のフラグビットが用いられる。

Ⅳ．４　ＬＩＳＰマシンのインタープリタ

Ⅳ．４．１　インタープリタのマイクロプログラム化

インタープリタの全体をマイクロプログラム化するに当たり、次の各点に注意して、処理速度の向上に努めた。

i)　ＥＶＡＬルーチンの改良

インタープリタ関数ＥＶＡＬの外部仕様はalistを用いないことを除いて、ＬＩＳＰ1.5に従っているが、その内部構造は、既存の定義にとらわれることなく次の示すアイデアのもとに設計しようとしている。すなわち、a. LISPプログラムのＳ式を左から右端までたどるのと全く同様の手順で、その内部表現である２進木リストを最後までたどり終えると、式の評価が完了するような、インタープリタを作ること。b. リストのcar部に現われるデータ（リスト、文字アトム、数値アトムなど）をあたかも機械語であるかのようにfetchし、解釈実行できるような制御構造をもたせること、である。したがって、ＥＶＡＬ中でのプログラムカウンタＰＣの概念は、命令をシーケンシャルに指定するものではなく、２進木リストのエレメントを指定するものであり、またＥＶＡＬルーチンの処理の流れも、ＬＩＳＰ1.5とは異なる部分がある（Ⅳ.4.3参照）。

ii)　ＡＰＰＬＹルーチンの改良（Ⅳ.4.4参照）

iii)　ハードウェアの活用

if then else を繰り返すタイプの条件判断を避け、マルチウェイジャンプ、間接ジャンプを多用する。マルチウェイジャンプは、マッピングメモリ出力の3ビットコード、またはフィールド抽出回路によって取り出された4ビットコードと、定数アドレスとの和の番地へ、1マイクロサイクルでジャンプするもので、データ型や文字アトムの属性の種類により多分岐するときに用いる。

また、演算結果がnilであるか、アトムか、リストか、数値か、文字アトムであるかといった判定は、専用のフラグをテストすることにより、1ステップで済ませる。さらに、無条件ジャンプの際には、ALU関係の処理を並行して行い、ステップ数を短縮する。

また、2つのポインタレジスタを持つ、強力なハードウェアスタックにより、スタックマシンのようにレジスタを使わないで処理が可能なほどであるが、Am 2903中の16個のレジスタを並用することによって、より効率向上をはかる。

iv) 主記憶アクセス回数の低減

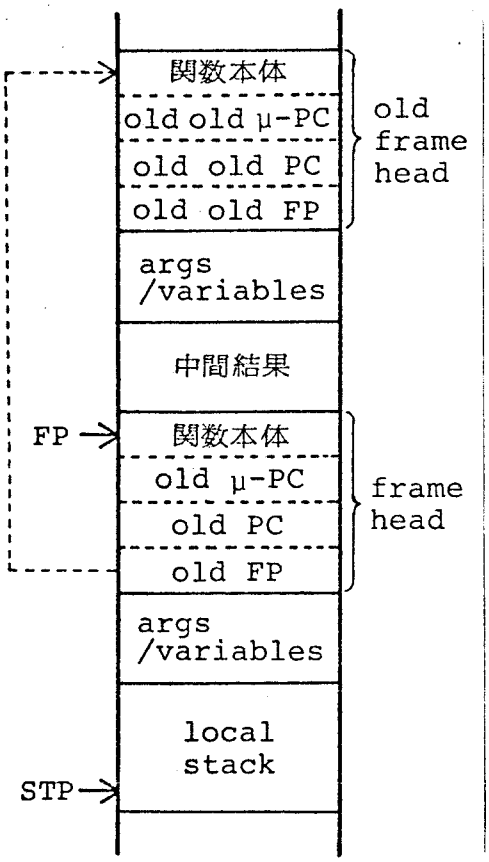
主記憶を参照してデータをスタックトップまで移動するのに、最低3マイクロサイクルかかるため、主記憶の参照回数は最低限におさえる。EVAL、APPLYルーチンの改良とともに、主記憶からの読み出し幅が32ビットであることを利用する。すなわち、リストのcar部、cdr部のうち、すぐに利用しないものはレジスタやスタックに保存しておいたり、文字アトムの場合にも各フィールドの配置を工夫して、レジスタに保存しておいてあとで使えるようなフィールドを隣どうしに置く。

Ⅳ．4．2　スタックの構成と制御構造

i)　スタックの構成

スタック上には、関数の実行に対応して、frameというものが作られる。frameの構造を図Ⅳ·2に示す。

frame headには関数本体へのポインタ、PCレジスタ、MPCレジスタを保存する領域（old PCとold μ-PC）及び1つ前のframe headを指すポインタold FPが含まれる。local stackには、関数本体の実行時には、計算の中間結果やマイクロサブルーチンへの引数などが置かれる。FPはframe pointerでframe headの位置を、STPはstack top pointerで、現在のスタックの先頭を示す。上記のうち、PC、MPC、FPの各レジスタは、Am 2903のレジスタファイル中にとられる。



図Ⅳ·2　frameの構造

PCは式の評価の際、ユーザープログラムの2進木リストを指して順にたどってゆくときに用いる。また、マシンコード実行のときには、普通の意味でのプログラムカウンタとしても用いる。MPCはマイクロプログラムカウンタの意味であるが、Am 2910中の真のマイクロプログラムカウンタへロードする値を保持するのに用いる。

Am 2910はサブルーチン用の5段のスタックを有するが、スタックの中味を外部に取り出せないため、再帰呼出しには利用できない。そこで、再帰を含むサブルーチンの呼出しには、あらかじめ戻り番地をAm 2910の外部の決められた格納場所に置いてジャンプし、リターンのときには間接ジャンプを用いて戻ってくる。MPCは、この戻り番地の格納場所である。したがって、再帰のときには、PCやMPCをスタックに保存する。

以後、MPCに戻り番地を置く場合をサブルーチンコールと呼び、frame headのμ-PC領域に直接に戻り番地を置いて、スタック操作をともなう場合をリカーシブコールと呼ぶことにする。またAm 2910のスタックを用いる場合を特にマイクロサブルーチンコールと呼ぶ。

ii) 制御構造

2引数のMRSUBR関数を例にとって、関数本体への制御の渡し方、関数本体での処理の流れ、リカーシブコール、リターンの仕方について、frameの取扱いと関係づけながら説明する。

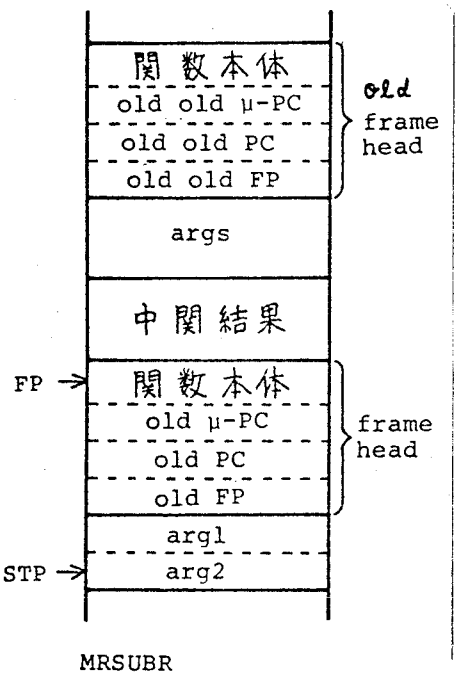
MRSUBR関数の本体は、マイクロプログラムで記述され、再帰呼出しを含んでいる。関数本体へ制御が移される前には、あらかじめスタック上に図Ⅳ·3のように、frame headと評価済のargsが並べられて、その後、関数本体へマイクロプログラムジャンプする。またframe headには、マイクロプログラムの戻り番地と呼び出し元のPCレジスタの内容が、保存されている(old μ-PCとold PC位置)。

関数本体での処理の流れの例を図Ⅳ·4に示す。スタート点へジャンプしてきたときのスタックの状態は①である。

まず、argsを用いて計算が行われ、終了条件が成立すると、呼び出し元へ関数値として返す値を計算し、RETURNオペレーションに移る。RETURNオペレーションの前のスタックの状態は④である。

終了条件が成立しないときは、再帰のためのargsを計算し、リカーシブコールに移る。スタックが②の状態でスタート点へジャンプすると、new frame headが①のframe headに対応する。

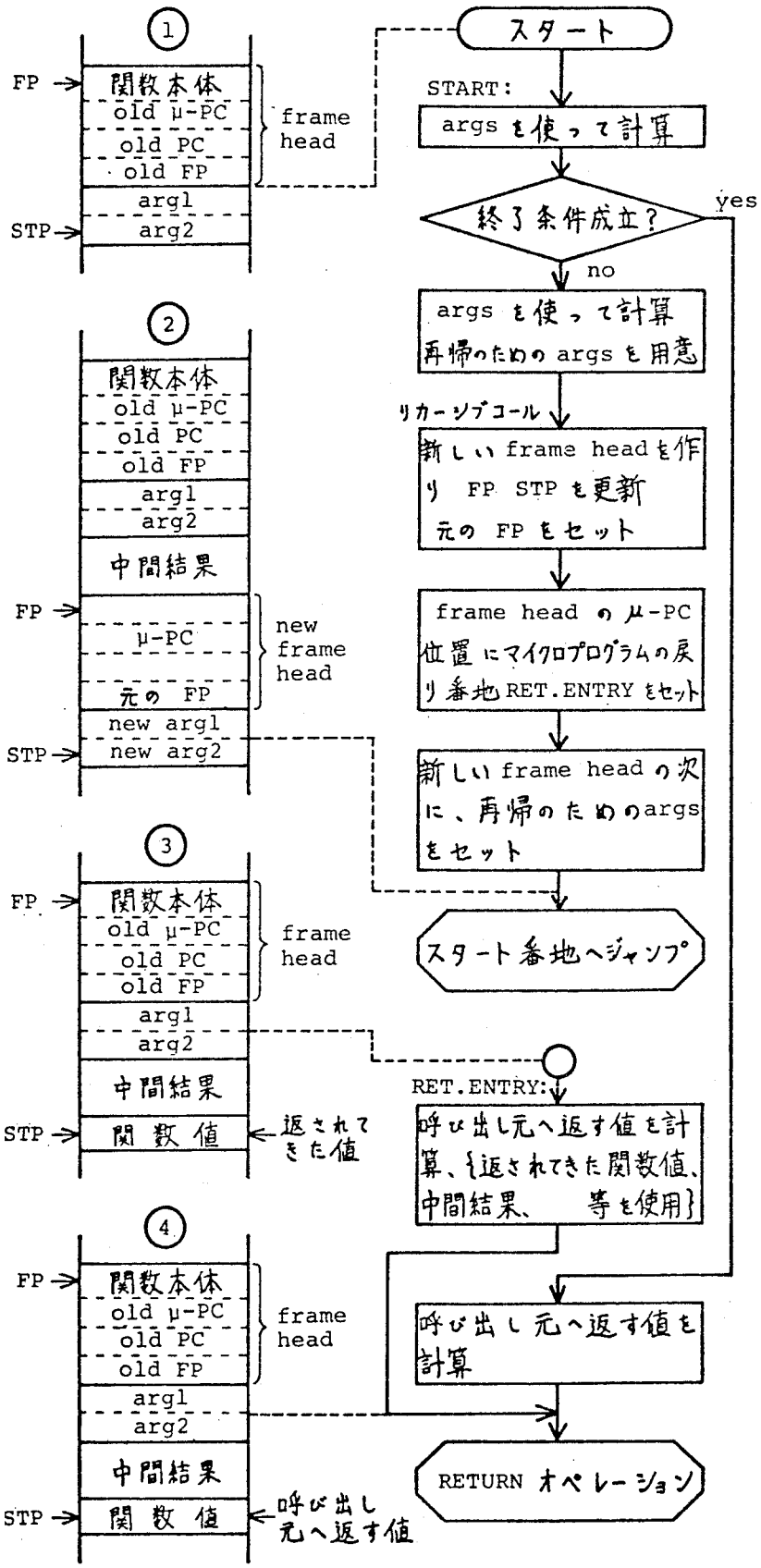
RETURNオペレーションは、関数値として呼び出し元へ返す値を呼び出し元のstack top(現FP位置)に置き、現frameを消滅させ、old μ-PCに保存してあった戻り番地へジャンプする操作である。スタックの状態で示すと、②のスタックの下側に中間結果と返すべき関数値が積まれた状態から、③の状態へと変化させる操作に対応する。RETURNオペレーションはすべての再帰関数に共通で、スタックに保存されていたPCの復元も行う。



図Ⅳ・3 MRSUBRが呼ばれるときのスタックの状態

MRSUBR関数の本体とは、frame headのold μ-PCフィールドを戻り番地の格納場所とする、マイクロプログラムの再帰的なサブルーチンと考えることができる。

Ⅳ.4.3 関数EVALと式の評価



図Ⅳ・4 MRSUBR関数本体における処理の流れ

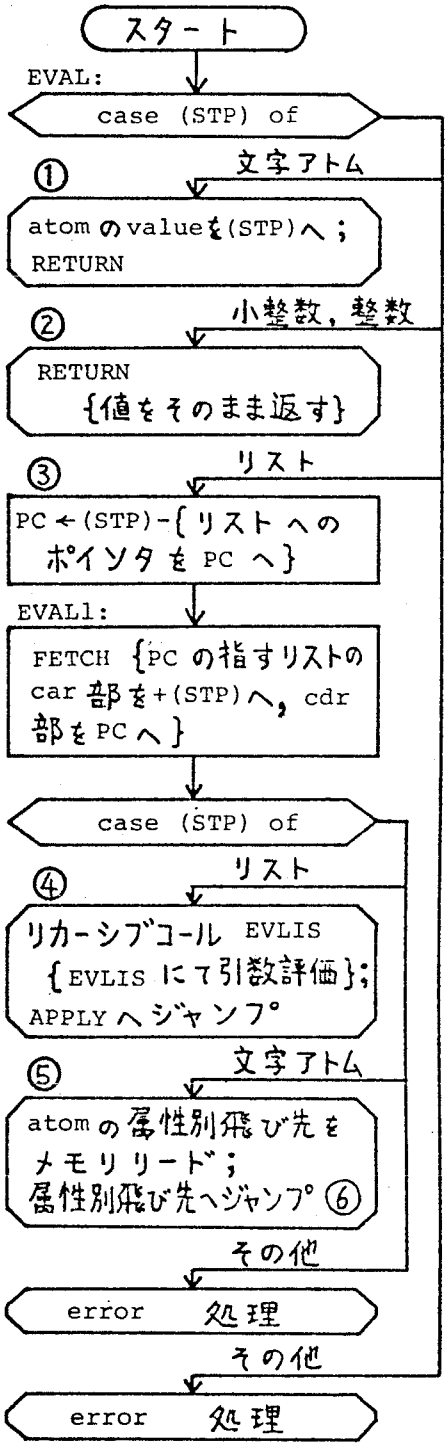


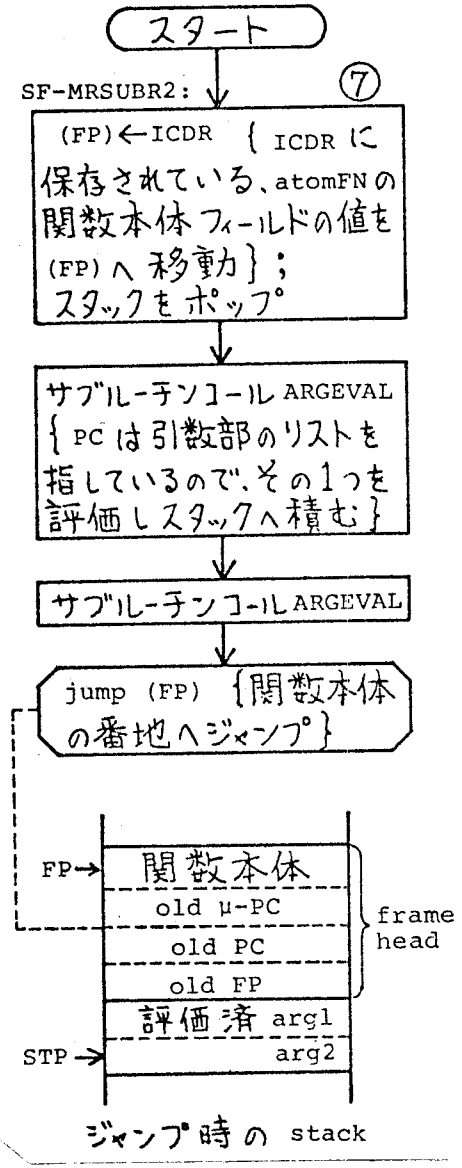
図 Ⅳ・5 EVAL ルーチン

```
eval[form;a]
    =[atom[form]→[numberp[form]→form;
                                    ②
          get[form;APVAL]→car[apval];
                          ①
          t→cdr[sassoc[form;a;error]]];
          ①
     atom[car[form]]
       →[get[car[form];EXPR]
     ⑤       →apply[expr;evlis[cdr[form];a];a];
             ⑥
         get[car[form];FEXPR]
               →apply[fexpr;list[cdr[form];a];a];
               ⑥
③       get[car[form];SUBR]
                 ⑦spread[evlis[cdr[form];a]];
              →{ $ALIST :=a;                    };
              ⑥{ call subr
         get[car[form];FSUBR]
                 { A1 :=cdr[form];
              →{ A2 :=$ALIST:a;                 };
              ⑥{ call fsubr
         t→eval[cons[cdr[sassoc[car[form];a;
              ⑥                  error]];cdr[form]];a]];
      t→apply[car[form];evlis[cdr[form;a];a]
    ]. ④
```

図Ⅳ・6 LISP 1.5 における EVAL の定義

```
apply[fn;args;a]
    =[null[fn]→nil;
      atom[fn]→[get[fn;EXPR]→apply[expr;args;a];
                ----------------------          ];
      eq[car[fn];LAMBDA]
        →eval[caddr[fn];pairass[cadr[fn];args;a]];
        ⑧
```

図Ⅳ・9 LISP 1.5 における APPLY の定義



図Ⅳ・7 SF-MRSUBR2 ルーチン

インタープリタ関数ＥＶＡＬは評価したい式を引数とするＭＲＳＵＢＲ関数であり、その本体の処理の流れを図Ⅳ･5に示す。また、図Ⅳ･6はＬＩＳＰ1.5におけるＥＶＡＬの定義（[ ]による）であり、図中の番号は、両者の対応する点を示している。図Ⅳ･5中、case（ＳＴＰ）ofで示した部分は、ＳＴＰの指すスタックの内容のデータ型により、マッピングメモリを用いて、マルチウェイジャンプすることを示している。

ＥＶＡＬルーチン、及びそれに続くルーチンの特徴をあげると、次のようになる。

i) もっとも頻繁に出現するルーチンを中心に高速化している。

ii) 関数が文字アトムのとき、引数評価にＥＶＬＩＳを使わず、スタックを利用して引数の受け渡しを行う。したがって、むだな自由リストの消費がなく、ガーベージコレクションも起こりにくい。

iii) むだなframeを作らず、必要に応じ、ＥＶＡＬやＣＯＮＤのframeは、評価しようとする関数やＡＰＰＬＹのframeに転用する。

iv) 関数を含む式の評価のとき、関数に対するframeは引数評価の前に作られる。したがって評価済の引数は、それを与えるべき関数のframeの、local stack位置へ積んでゆくことができる。

式の中で、もっともよく現われる形（関数名　引数　引数 ･･･）を評価する場合には、ＥＶＡＬルーチンでの処理の流れは、図Ⅳ･5の③から⑤に至る。そして⑥で、文字アトムの属性により、決められたルーチンへジャンプする。ここで、上述のii)を示すために、関数が2引数のMRSUBR

の場合の式（ＦＮ　ＡＲＧ１　ＡＲＧ２）の評価を例にとって、⑥以後のルーチンを説明する。

その場合、飛び先は、ＳＦ－ＭＲＳＵＢＲ２ルーチン（図Ⅳ･７）となり、２個の引数の評価と、評価後、関数本体へのジャンプが行われる。評価済の引数は、スタックに積んで関数本体に渡すため、図Ⅳ･６のspread〔evlis〔…〕〕に相当する関数、ＡＲＧＥＶＡＬ（図Ⅳ･８）をサブルーチンコールしている。また、ＳＦ－ＭＲＳＵＢＲでは、ＦＰの示す位置へ関数本体を書き込み、ＥＶＡＬのframeから評価しようとする関数のframeへと転用をはかっている。

ＡＲＧＥＶＡＬルーチンは前述のi)により、引数評価専用に、ＥＶＡＬルーチンを変形したもので、本インタープリタの持つ特徴的なルーチンである。そして、引数として再び（関数名　引数　…）の形が現われた場合には、その評価のために、ＥＶＡＬルーチンのリカーシブコールをも行う。この場合高速化のために、ＥＶＡＬのスタート番地ではなくて、ＥＶＡＬ１へジャンプし、引数もＰＣに置く変則リカーシブコールとなる。新たに現われた関数のためのframeは、ＥＶＡＬで引数評価をする前に、この部分で作られていることになる。

Ⅳ．４．４　ユーザ定義関数の実行とＡＰＰＬＹ

ユーザ定義関数を実行するときには、実引数と仮引数のbindのためＥＶＡＬルーチンからＡＰＰＬＹを呼び、さらにＥＶＡＬを呼ぶ手順が必要となる。これらの処理を高速化するために、次のような改良を加えた。

i) ＡＰＰＬＹの中でもっとも頻繁に現われるルーチンは、図Ⅳ･9では⑧であり、⑧に相当する部分を中心にＡＰＰＬＹルーチンを設計する。

ii) ＡＰＰＬＹルーチンへの入口を⑧に相当する部分にも設け、高速化をはかる。

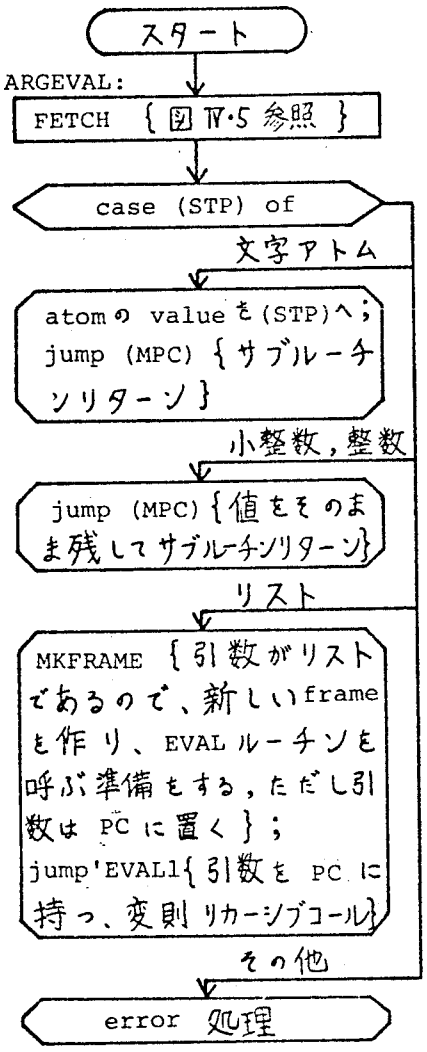
iii) bindしようとする実引数をＡＰＰＬＹへ渡す際、リストの形でなく、スタックに積んだままの形で渡すことを可能にする。

iv) shallow-bindingと、アトムのold-valueをスタックに残す処理ルーチンを、ハードウェアの特徴（2個のポインタレジスタを持つスタック、メモリのread while write 機能〔/〕等）が生かせるよう設計する。

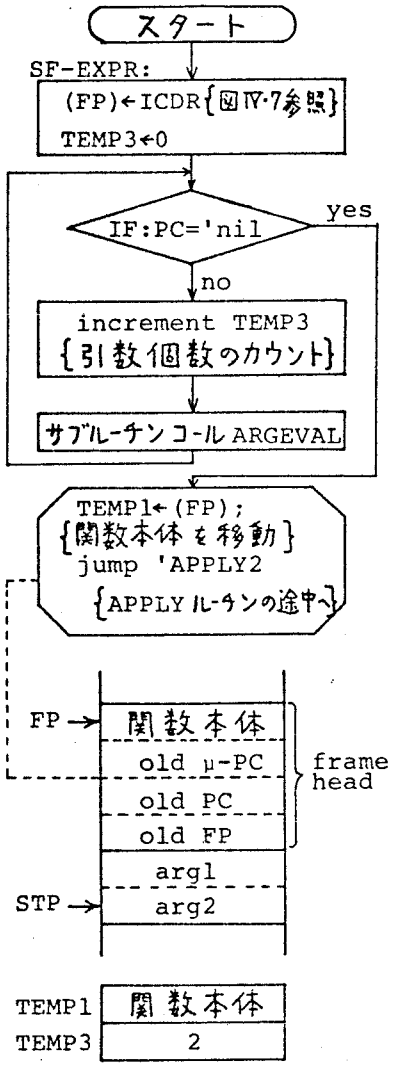
ユーザ定義関数は、関数定義関数ＤＥにより次のように定義される。（ＤＥ ＵＳＲＦＮ（Ｘ Ｙ）（ＣＯＮＤ---））その結果、アトムＵＳＲＦＮの関数本体フィールドには、（（Ｘ Ｙ）（ＣＯＮＤ---））というリストへのポインタが、また引数個数フィールドには2がセットされる。

式（ＵＳＲＦＮ ＡＲＧ１ ＡＲＧ２）の評価の場合、ＥＶＡＬルーチンの⑥から、ＳＦ-ＥＸＰＲルーチン(図Ⅳ･10)にジャンプする。ここでは、引数を評価してスタックへ積むとともに、引数個数をカウントして、スタックとレジスタを介してＡＰＰＬＹ２へ渡す。ＡＰＰＬＹ２とは、前述の⑧に相当する入口である。

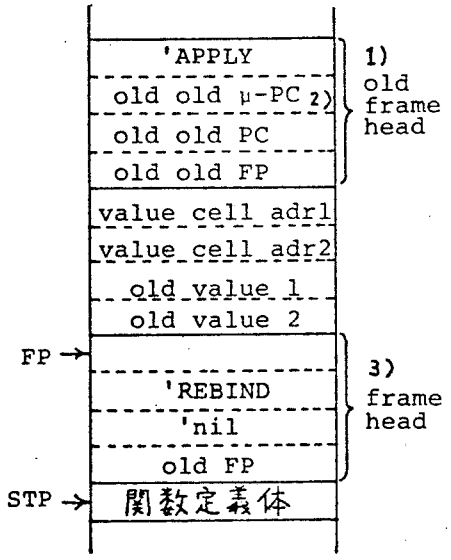
ＡＰＰＬＹ２では、shallow-bindingを行ったのち、アトムのold-valueとその番地の対をスタックに残し、関数定義体（ＵＳＲＦＮの本体のうち（ＣＯＮＤ---）の部分）を引数として、ＥＶＡＬをリカーシブコールする。戻り番地としては、old-valueを書き戻すための、REBIND



図Ⅳ·8 ARGEVAL ルーチン



図Ⅳ·10 SF-EXPR ルーチン



図Ⅳ·11 old value を保持するスタック

ルーチンの入口がセットされる。このときのスタックの状態を図Ⅳ.11に示す。

MRSUBR関数APPLYの外部仕様は、shallow-bindingであること、FUNARG機能を持たないことを除いてLISP1.5と同じである。

Ⅳ.5 LSI-11のソフトウェア

LSI-11は、システムの初期化の仕事と入出力処理の一部を担当することをⅣ.3節において述べたが、初期化の段階ではLISPプロセッサに対してLSI-11がマスターであり、入出力の時には逆にスレーブになるという特徴を有する。LSI-11のLISPモニタは、これらの、レベルの異なった仕事を処理することになる。

モニタの機能を大別すると、

(1) JOBの実行開始、停止、終了の管理

(2) インタープリタからの要求による入出力処理、タイマー制御処理(割込み処理)

(3) マイクロプログラムのデバッグ機能

の3種類となるが、(2)がスレーブとして働く場合であり、割込みにより対応する処理ルーチンが起動される。

(1)については、モニタは以前のJOBの実行によって使用されたアトムヘッダ領域、フルワード領域等を初期化したのち、インタープリタ中の初期化ルーチンから実行を開始するようにLISPプロセッサに起動をかける。LISPプロセッサを起動したのちは、JOBの停止、終了のコマンド待ちとなる。一方インタープリタは、初期化の仕事を

終了後、トップレベルごとの実行をくり返すが、この中で、LSI-11に出された要求は、前述のとおり割込みにより異なったレベルで処理される。これが(2)である。

(2)の割込み処理を分類すると

i) 入出力処理

ii) タイマ制御処理

iii) メッセージ出力処理

の3つに分かれ、入出力処理ではⅣ.3節で述べた以外に、括弧の対応のチェック、アトム名(プリントネーム)の長さのチェック、数値アトムの変換やレンジチェック、括弧の対応レベルのプリントなどが行われる。ii)のタイマ制御には、タイマカウンタの始動、停止、待避、復帰があるが、待避、復帰は、トップレベルの実行中にREAD、PRINT関数やガーベージコレクションにはいったときに要求される。iii)のメッセージ出力については、

(a) トップレベルの実行開始

(b) トップレベルの実行終了

(c) トップレベルの実行時間

(d) ガーベージコレクションの収集セル数と実行時間

(e) エラーメッセージ

等がある。

最後に(3)のデバッグ機能について述べる。マイクロプログラムのデバッグを行う際、LISPプロセッサ側にそれを助ける機能を盛りこむことは困難であり、したがってLSI-11側のデバッグ援助機能がたいへん重要となる。

それらを列挙すると、

(a) WCS、マッピングメモリ、主記憶の各メモリセルの読み書き

(b) CMR、ページレジスタの読み書き

(c) マイクロプログラムの1ステップ実行

(d) ブレイクポイント設定による任意アドレスでの実行停止、再設定と再実行

となる。(a)の機能により、マイクロプログラムコードの修正等が速やかに行え、また(d)の機能により、まずブレイクポイントを設定して実行することによって、任意のアドレスでマイクロプログラムの実行を停止でき、その時のメモリ内容等を知ることができる。またそのときのプロセッサモジュール内のレジスタの内容は、メンテナンスパネル側から知ることが可能である。本システムでは、デバッグのためにシュミレータを用意することをしなかったため、これらのデバッグ機能が非常に有用となった。

また、連続ステップ動作を利用して、マイクロプログラムの実行総ステップ数を計数する機能を付加しており、プログラムの動特性の測定時に利用している。

Ⅳ. 6 ソフトウェアの作成

実際に作成されたソフトウェアのサイズは、おおよそ次に示すとおりである。

i) マイクロプログラム： 1369総ステップ。

インタープリタ — 539 ステップ
READ関数 — 115 ステップ
PRINT関数 — 67 ステップ

ガーベージコレクション ― 118 ステップ
システム関数群 ― 212 ステップ

ii) LSI-11のモニタ： 3.3k語（1語16ビット）。
モニタの中核部 ― 1.3k語
割込み起動のルーチン群 ― 2k語

iii) マイクロプログラムアセンブラ： FORTRANにて1300総ステップ。

なおマイクロプログラムは、システム関数が55個実装された場合のものである。

またソフトウェア上の問題として、マイクロプログラムのコーディングがたいへん煩わしいこと、インタープリタに実装されているLISPプログラムのためのデバッグ機能がまだ不十分であること、それに付随してマイクロプログラムを任意の番地から実行させる機能がまだないこと、などが上げられ、多くの改善の余地を残している。

Ⅳ.7 結 言

本システムで採用したLISP言語の仕様のうち文字アトムの構造が処理の高速化に適することを示し、またマイクロプログラムで記述される関数の関数型の中で再帰呼出しを許すものがあることをまずはじめに示した。つづいてシステムの初期化の仕事と、入出力処理について述べ、入出力においてはLISPプロセッサとLSI-11が処理分担をしていることを示した。そのうえで、本システムの中心であるマイクロプログラム化されたインタープリタに言及し、制御構造の改良とハードウェアの活用という二面から、インタープリタの高速化が図られていることを示した。

また関数ＥＶＡＬの中でのプログラムカウンタの概念はリストをたどってゆくものであること、引数評価の高速化のためにＡＲＧＥＶＡＬというルーチンを導入したことを述べた。さらにＬＳＩ－11のＬＩＳＰモニタ中のデバッグ機能が、マイクロプログラムデバッグにたいへん有効であることを示し、最後にマイクロプログラムの総ステップ数が1400ステップ足らずでかなり小さいことを示した。

第5章　ＬＩＳＰマシン システムの評価

Ⅴ．1　緒　言

前章までに述べように、本ＬＩＳＰマシンシステムは、プロセッサモジュールのハードウェア構成とインタープリタの制御構造の双方に、十分な高速化のための努力が払われており、それらが協調し合って、高速のＬＩＳＰ処理を実現することが期待されるわけである。

本章では、完成したシステムの上で実際にいくつかのプログラムを走らせたときの処理時間を示し、従来の大型計算機上に構成されたシステムとの比較を行う。さらに数個の例題を用いてＬＩＳＰインタープリタの動的特性を測定し、その中でのハードウェア各部の利用状態を定量化し、それぞれに対して評価を加える。最後にそれらの結果から、全体的な評価と考察を行い、本システムの成功した点と問題点を明らかにする。

Ⅴ．2　ＬＩＳＰプログラムの実行時間

1978年に行われた第２回ＬＩＳＰ性能コンテスト[7]に出題されたプログラムを用いて、本システムにおける実行時間を測定した。その結果を、他の処理系の実行時間と合わせて表Ⅴ･1に示す。他の処理系の実行時間は、文献[7]によるものであり、またプログラムのリストを付録.4として添付する。表中の数値は実行時間を m sec の単位で表わしたものであり、ガーベージコレクション時間は含まない。数値に添えられた＋、＋＋の記号はガーベージコレクショ

	-----INTERPRETER-----						-----COMPILER-----			
	FAST-LISP KOBE UNIV.	HLISP	OLISP	TOSBAC-5600 LISP1.9	LIPQ		HLISP	OLISP	TOSBAC-5600 LISP1.9	LIPQ
BITA-4	6	6	14	24	90	*	2	4	7	18
BITA-5	17	22	46	76	300	*	9	15	22	63
BITA-6	55	71	155	249	1,010	*	30++	50	71	219+
BITA-7	188	249	531	936	3,460	*	108+	176	248	775+
BITA-8	652	869	1,849	2,925	---	*	377++	622	499	---
BITA-9	STACK OVER	3,111+	6,498++	15,236++	---	*	1,359++	2,229	2,705	---
BITB-4	3	6	5	11	45	*	1	1	4	9
BITB-5	6	14	13	23	104	*	3	3	6	25+
BITB-6	13	40	34	59	282	*	10++	9	14	75+
BITB-7	37	118+	104	172	870	*	33	28	39	258+
BITB-8	120	394+	338	549	3,070+	*	105++	91	126	1,080++
BITB-9	397	1,318+	1,130	1,832	---	*	356++	307	712	---
BITB-10	1,355(+)	4,542++	---	---	---	*	1,230++	---	---	---
SORT-20	73	106	287	505	1,400	*	12	18	31	23++
SORT-40	261	374	1,109	1,415	4,900	*	43	69	108	85++
SORT-60	415	---	2,459	2,061	7,800	*	---	153	176	140++
SORT-80	626	568	4,337	3,121	11,900	*	65	271	487+	210+
SORT-100	804	1,134	6,753	6,154	15,000	*	135	423	605+	270+
TARAI-3	55	78	197	277	900	*	17	36	87	13
TARAI-4	1,013	1,443	3,727	5,124	16,000	*	330	670	1,604	255
TARAI-5	27,538	39,068	100,734	138,819	437,000	*	8,905	18,934	43,609	6,900
TARAI-6	1,011,732	---	---	---	---	*	322,347	---	1,603,910	255,000
TPU-1	904	4,790	2,262	4,403+	12,000+	*	1,029	658	1,591	2,200+
TPU-2	3,426	13,411	10,262	21,148++	55,200+	*	2,871+	2,064	6,764+	6,100+
TPU-3	1,365	5,684	3,932	7,447+	21,200++	*	1,227++	850	2,798+	2,600+
TPU-4	1,815	8,025	5,042	10,958+	27,400++	*	1,707	1,161	3,250+	3,700+
TPU-5	238	866	759	1,461+	4,000++	*	181	132	555	400
TPU-6	6,728(+)	22,849	20,241	66,502++	118,000++	*	4,893+	3,617	27,049++	10,500+
TPU-7	1,311	5,078	4,179	8,350+	21,800++	*	1,053	776	3,189+	2,300+
TPU-8	1,187	3,459	3,987	5,459+	21,000++	*	724+	596	1,163+	1,600+
TPU-9	819	2,663	2,716	4,133+	14,200++	*	545	424	964+	1,100+
PLISP-3	7,717	3,096	46,061	49,125	---	*	303	1,876	2,933	2,240
PLISP-3	163	132	575	871	2,150	*	132	575	871	2,150
RATIO	47.3	23.5	80.1	56.4	---	*	2.3	3.26	3.4	1.04
PLISP-4	170,471(+)	56,877	1,096,786	1,118,014	---	*	4,998	44,910	66,722	48,500
PLISP-4	3,059(+)	2,020	11,384	18,119	44,000	*	2,020	11,384	18,119	44,000
RATIO	55.7	28.1	96.3	61.7	---	*	2.5	3.95	3.7	1.1
PLISP-5	STACK OVER	---	---	---	---	*	125,290	---	2,015,495	---
PLISP-5	83,321	46,431	---	503,999	1,249,000	*	46,431	---	503,999	1,249,000
RATIO	---	---	---	---	---	*	2.7	---	4.0	---

表 V·1　LISPコンテストのプログラムの実行時間（msec）

ンが発生したその時間が、それぞれ実行時間の10%～20%、20%以上であったことを示している。本システムにおいては10%を越える場合がなかったので(+)で示した。

比較の対象として選んだ他の処理系の特徴を以下に簡単にまとめておく(文献[7]参照)。

i) HLISP: LISPコンテストにおいてもっとも速かったシステムで、HITAC-8800上に構成され、インタープリタの制御構造に他と異なる特徴をもつ。HITAC-8800は、50n secのCPUサイクルタイムと100n secの主記憶バッファメモリをもつ高速の超大型計算機システムである。

ii) OLISP: LISPコンテストにおいてHLISPと肩をならべて速かったシステムで、ACOS-800上に構成されたシステムである。

iii) TOSBAC-5600 LISP1.9: TSS向きのLISPシステムでTOSBAC-5600のほか、ACOSシリーズの計算機に移植でき、著者の研究室においても利用している。

iv) LIPQ: ミニコンピュータ上に構成された高速のシステムで、システム規模が本LISPマシンに近いので取り上げた。ミニコンピュータはPDP11/55で、CPUサイクルタイムが300n sec、メモリサイクルタイムが同じく300n secで、本システムと同等である。

以上のことを念頭において表V-1をみると、まず本LISPマシン(表中FAST-LISP)における実行時間が、他のどの大型計算機よりも短いことがわかる。本システム

のサイクルタイムが300n secであるから、ＨＬＩＳＰと単純に比べると6倍遅いことになるが、ハードウェア、ソフトウェア両面での改良の効果が、十分に現われたと考えることができる。

測定に利用したプログラムは、ＢＩＴＡ、ＢＩＴＢ、ＳＯＲＴ、ＴＡＲＡＩ、ＴＰＵ、ＰＬＩＳＰの6種類であるが、他システムと実行時間の比をとると、プログラムごとに似た値となる。たとえばＨＬＩＳＰと比較すると、ＢＩＴＡでは1.3倍、ＢＩＴＢでは3倍、ＳＯＲＴでは1.4倍、ＴＰＵでは4倍前後、本システムの方が速いことがわかる。プログラムごとにこの値が異なるのは、システムによって、インタープリタの動作や関数の実行速度に差があるためと考えられる。

表Ⅴ·1には、他システムのコンパイラによる実行時間も示してある。ＬＩＳＰコンパイラを用いると、ＬＩＳＰのソースプログラムが機械語列に変換されて直接実行できる形になり、組込み関数と同等になる。すなわち、インタープリタが「解釈」する時間が不要になって高速化が図られる。本システムはコンパイラの機能をまだ持たないが、他システムのコンパイラによる実行時間との比較を行ってみても、たいして劣らないことがわかる。この高速性の理由は、十分に検討する必要があり、次節以降にて考察する。

Ⅴ．3　インタープリタの動的特性の測定と各部の評価

Ⅴ．3．1　インタープリタの動的特性の測定

ハードウェア各部の評価を行うためには、それらの利用

状況を的確につかむ必要がある。そのために、評価用のプログラムを実行させてそのときの、インタープリタルーチンやシステム関数の呼ばれた回数を測定するという方法をとった。そして、あらかじめそれぞれのルーチンについてステップ数と各ハードウェア機能の利用された回数を求めておき、ルーチンの呼ばれた回数との積をとる。そうすると、プログラム実行時のマイクロプログラムの総実行ステップ数と、ハードウェア機能の利用された回数が求められる。

表Ⅴ·2, Ⅴ·3はＴＡＲＡＩ－３とＴＰＵ－６を実行させたときの、総ステップ数、ならびに各ルーチンの実行ステップ数の合計が全体にしめる割合である。この表は、ステップ数の比率を示しているが、それはほぼそのまま、各ルーチンの実行時間が全実行時間にしめる割合に対応する。なおルーチンの呼ばれた回数等の生データは、付録.5に示す。ＴＡＲＡＩ－３、ＴＰＵ－６のプログラムの特徴は、前者が外部定義関数の再帰呼び出しばかりで構成される評価試験用のプログラムであるのに対し、後者は定理証明のプログラムで、ＰＲＯＧ形式を多く含む実際的な形をしている。プログラムの特徴は表Ⅴ·３に現われていて、TARAIではインタープリタ関数やルーチンのステップ数が85.2%と圧倒的に多いのに対し、ＴＰＵでは逆にシステム関数のステップ数が56.3%と多い。表にはないが、システム関数のステップ数のうち６%は関数からのリターンに使われるステップである。２つの測定結果に共通していえることは、ＡＲＧＥＶＡＬルーチンのしめる割合が大きいことで、これは関数の引数評価用のルーチンであり高速化に特に努力を払ったところである（Ⅳ.4.3参照）。すなわち、高速化

表 V·2　LISP プログラムの実行時間とマイクロプログラムの実行ステップ数

プログラム	実行時間 (m sec)	総実行ステップ数	1ステップの平均実行時間(n sec)
TARAI-3	55	154,763	355
TPU-6	6,726	19,457,963	346

表V·3 (a)　TARAI-3実行時の各関数、ルーチンのステップ数にしめる割合

関数またはルーチンの名前		比率(%)
interpreter functions / routines (85.2%)	EVAL	13.9
	ARGEVAL	28.0
	APPLY	26.5
	EXPR	7.4
	COND	9.2
system functions (14.8%)	GREATERP	9.3
	SUB1	5.5

表V·3 (b)　TPU-6実行時の各関数、ルーチンのステップ数にしめる割合

関数またはルーチンの名前		比率(%)
interpreter functions / routines (43.7%)	EVAL	9.3
	ARGEVAL	13.0
	APPLY	4.2
	EXPR	1.2
	PROG	9.7
	GO	1.4
	RETURN	0.5
	COND	4.4
system functions (56.3%)	SUBST	17.3
	EQUAL	11.3
	SETQ	6.3
	MEMBER	4.8
	CAR	3.2
	NULL	2.1
	ATOM	1.6
	CDR	1.6
	OR	1.5
	CADR	1.2
	APPEND	0.8
	LENGTH	0.7
	CADDR	0.7
	LIST	0.5
	EQ	0.4
	CONS	0.4
	others	1.9

した部分のステップ数比がなお高いということで、それだけルーチンの改良の効果が大きく現われていることが考えられる。なおTPUにおいては、SUBSTという関数のステップ比がもっとも大きく、これはSUBST内での再帰が非常に多いからであるが、この関数は他のプログラムで用いられることは多くなく、したがってTPUは、評価用のプログラムとしてはあまり適しているとはいえない。

つぎに、各ルーチンや関数に含まれるハードウェア機能の利用回数の比率を集計したものを表V·4に示す。ここでいう比率とは、各ハードウェアオペレーションを含むステップの総和が、全実行ステップ中にしめる割合のことで、当然ながら1つのステップに複数のオペレーションを含むことは多く、比率の合計をとっても100%にはならない。表V·5は、TPU-6におけるメモリオペレーション数の合計を100として集計しなおし、そのうち分けを示したものである。ただしread while writeオペレーションの実行時間はwriteオペレーションの中に含まれる。表V·6はTPU-6の中の直接ジャンプ(表V·4(b)の条件ジャンプと無条件ジャンプの和)オペレーションを集計しなおして、うち分けを示したものである。jump with other op.とはジャンプしながらその他のオペレーションを並列実行することである。

これらの表からまずわかることは、ジャンプとスタックオペレーションの比率がきわめて高いことであり注目に値する。その原因としてはLISPがインタープリティブなシステムであり関数呼び出し毎にスタックをのばしたりちぢめたりすること、データ型を判断して対応する処理ルーチンに飛ぶ操作が多いことなどが考えられる。表V·4～

表V・4(a) TARAI-3 における
ハードウェア機能の利用の割合*

オペレーション		比率(%)
MEMORY ACCESS (15.4%)	read	12.8
	write	2.6
JUMP (41.0%)	conditional	17.0
	un-conditional	13.0
	indexed / indirect	11.1
STACK operation (38.0%)	source	19.4
	destination	21.0
	( )−	13.0
	+( )	12.7
	stack to stack	1.1
直接数値演算		22.2
ALU オペレーション		20.2
FEX オペレーション		1.3
マッピングメモリ直接		7.0
IDLE		1.0

* 総ステップ数 154,763 中にしめる
各オペレーションを含むステップ数の割合

表V・4(b) TPU-6 における
ハードウェア機能の利用の割合*

オペレーション		比率(%)
MEMORY ACCESS (11.4%)	read	10.2
	write	1.2
JUMP (38.3%)	conditional	15.0
	un-conditional	13.4
	indexed	3.9
	indirect	6.1
STACK operation (49.0%)	source	26.7
	destination	25.7
	( )−	18.9
	+( )	17.8
	stack to stack	3.4
直接数値演算		19.1
ALU オペレーション		20.3
FEX オペレーション		0.6
マッピングメモリ直接		3.9
IDLE		0.4

* 総ステップ数 19,457,963 中にしめる
各オペレーションを含むステップ数の割合

表V・5 TPU-6 における メモリ
オペレーションの うち分け

オペレーション	比率(%)
READ carのみ利用	34.0
READ cdrのみ利用	19.2
READ car,cdr利用	36.2
WRITE car	7.5
WRITE cdr	2.1
WRITE both	1.0
read while write	7.0

表V・6 TPU-6 における 直接
ジャンプのうち分け

オペレーション	比率(%)
un-conditional	47.3
MAP - flag	17.3
NIL - flag	12.1
Stack over-flag	10.3
ALU - flag	13.0
jump with other op.	70.6

V·6により、ハードウェア機能の利用状況がかなり正確につかめるので、次節においてハードウェア各部の評価を行う。

V．3．2　測定結果によるハードウェア各部の評価

表V·4～V·6に示した比率から、各オペレーションに関係するハードウェアが役に立っているかどうか、またハードウェアの改良に努力した部分についてそれが有効であったかどうかの評価を加えてゆく。

i)　ハードウェアスタック

スタックオペレーションの比率がたいへん高いことが分かったが、以下3点について考察を加える。

第一は、スタックアクセスタイムという観点に立ってハードウェアスタックを見た場合である。本システムではスタックアクセスの時間は、完全に1つのマイクロプログラムサイクルに組みこまれてしまう。したがってスタックアクセスタイムは、スタックをデスティネーションとした場合にはゼロ、ソースとした場合にはサイクル延長分の75n sec（付録.1参照）のみとなる。このことはスタックオペレーションの比率からみて十分高速化に貢献しているといえる。スタックアクセスの時間を不要にしているのは、ハードウェアスタック、ALU、バス相互のデータ経路の選び方と、スタックポインタレジスタの使用によるもので、成功であったといえる。

第二は、スタックオペレーションのうち間接後減少、増加後間接（表中（　）－，＋（　）で示したもの）の利用である。表V·4(b)によると、（　）－のsourceにしめる割合と

十( )のdestinationにしめる割合は共に約70%で、もし自動減少、増加の機能がないとそのためによけいなステップを必要とすることになる。自動増減の機能を持たせたこと、すなわちスタックポインタレジスタにカウンタタイプのレジスタを採用したことは成功であった。

第三は、スタックからスタックへ1マイクロプログラムサイクルでデータ転送ができる機能についてである。表V·4(b)にはstack to stackで示してあるが、3.4%しか利用されておらず、かならずしも必要ないと言えそうである。しかしながらこの機能は、プロセッサモジュールのデータ経路を決定したときに副次的に生まれたもので、より高速のプロセッサを再設計するときには省略して良いかもしれない。

以上から、ハードウェアスタックを実装したこと、強力な機能を持たせたことは成功であったといえる。

ii) ジャンプに関するハードウェア機能

表V·4からジャンプオペレーションの比率もたいへんに高いことが分かった。以下3点について考察を加える。

第一は、もっとも比率の高い条件ジャンプに関するもので、そのうち分けは表V·6から分かる。マッピングメモリ関係フラグ、NILレジスタフラグ、スタックオーバフローフラグ、ALU関係フラグのいずれも10%台の利用率で、大きな片寄りはない。もし前三者のフラグをなくした場合には、定数との比較の後ALUフラグジャンプということになり、1ステップないしパイプライン処理を乱すときには2ステップのオーバヘッドとなる。したがって、マッピングメモリ出力にデコーダを用意してフラグテストができるようにしたこと、NILレジスタを用意したこと、スタ

ックオーバフローの検出回路を設けたことは、いちおう成功だったといえそうである。なおＮＩＬレジスタについては、ＩＣＡＲレジスタ、ＰＣレジスタに密着させる方が、より使い易いという意見があった。

第二は、表Ⅴ･6のjump with other op.に関することである。これは、ジャンプをしながら他にスタックオペレーションやＡＬＵオペレーションその他を並行して行うことで、無条件ジャンプについては飛び先での仕事の先まわり実行、条件ジャンプでは1つ前のステップ条件によりジャンプするのでパイプライン処理の形になる。この機能は直接ジャンプのうちの70％に利用されており、マイクロ命令コードを長くとって並列実行を可能にしたことが、成功であったといえる。

第三は、インデックスジャンプと間接ジャンプの、いわゆるＹ-バスジャンプに関することである。これも表Ⅴ･4(b)では合計で10.0％の比率を持ち、ジャンプ中にしめる割合は26％である。この二者はほとんどインタープリタ中で使用され、マイクロプログラムの再帰呼び出しを許したりマルチウェイジャンプを可能にしたりする。この二者を除くと、インタープリタの構造自体を変更せねばならず、効率の低下は大きいと考えられる。したがって、ジャンプアドレスとしてＹ-バスデータが利用できるというバス構造は、なくてはならないものということができる。

以上、ジャンプオペレーション関係のために盛りこんだハードウェア機能は、いずれも成功であったといえる。

iii) メモリオペレーションに関する機能

表Ⅴ･4(b)によると、メモリアクセスのためのステップ比率は11.4％で一見少ないように見える。しかしながら、10

ステップに一度はメモリ参照があると考えるとやはり多いといえる。本システムにおける特徴は、第一にメモリアクセスタイムが短く待ち時間が1サイクル分しか必要でないこと、第二にメモリ待ちのサイクル中に別のオペレーションが可能でありむだ時間にならないこと、第三に読み出し幅が32ビットで、car部、cdr部を同時に読み出し、メモリアクセス回数を減らせること、第四にread while write機能があることである。

第一点については、メモリアクセス数が多い（TPUでは全ステップの一割強で200万回）ことから待ち時間が短いのが特に有効といえる。第二点は、表中のIDLEで示されるところが何もしない純粋なメモリ待ちであることから、TPUの場合には、全メモリ参照のうち$(11.4-0.4)\div 11.4\times 100 \fallingdotseq 96\%$までが、メモリ待ちのサイクル中に別の仕事をしていることになる。すなわち、本システムにおいては、メモリ参照はもう効率を落とすネックにはなっていないといえる。第三点の読み出し幅については、表V·5より、car, cdrの両方を利用するためのREADがメモリアクセス中36.2%をしめ、比較的よく利用されていることが分かる。第四点については、read while write機能の利用は、メモリアクセス中の7%、WRITEの中の$7\div(1+2.1+7.5)\times 100 \fallingdotseq 66\%$をしめることであるが、もしこの機能をなくすると、読み出してどこかに保存したあと書きこみを行う手順が必要となり、1回当り最低3ステップのオーバヘッドとなる。したがってTPUでは、全ステップに対して2.4%程度のオーバヘッドが生じることになるが、この程度を許すならば、read while write機能は除去できる。

以上から、メモリアクセス関係のハードウェア構成はほ

ぼ成功したと言え、またメモリアクセスがＬＩＳＰ処理の速度低下を招かないようになったことも大きな進歩といえる。

ここで本システムのメモリアクセスとスタックアクセスを比較するため、読み出しシーケンスを記述すると、

メモリ：　アドレス→ＭＡＤＲ；
　　メモリ待ち；
　　ＩＣＡＲ（またはＩＣＤＲ）→データ利用；

スタック：
　　（ＳＴＰ）→データ利用；　など.

となり、２ステップの差のあることがわかる。この差の意味については、別途検討の余地がある。

また表Ⅳ･4(b)から、ＴＰＵにおいては、約３μsecに1度のメモリアクセスがあることが分かるが、これは低速のミニコンピュータが、機械語を実行するときのメモリ参照頻度に近いことを付記しておく。

iv)　直接数値演算機能

５ステップに１度は利用されており、マイクロ命令コードを56ビットと長くしてこの機能を持たせたことが、有効だったと考えられる。

v)　マッピングメモリとフィールド抽出回路

マッピングメモリを直接読み出して利用するのは、TPUでは、インデックスジャンプの時だけであることがわかる。したがって利用率は高くないが、除くことはできない。フィールド抽出回路（表中ＦＥＸ）の利用率は、ＴＰＵでは１％を割るが、もし実装しなければシフトとマスクにかなりのステップを食うことになる。コンパイラやタグマシンの場合には、より有効になると考えられる。

vi) ＡＬＵオペレーション

ＡＬＵオペレーションの比率はＴＰＵでは20.3%しかなく、またデータ転送不要のステップはＩＤＬＥと単なる直接ジャンプだけであるから、全実行ステップ中の71%は、単にデータの移しかえだけを行っていることになる。したがってプロセッサモジュールのバス構成を変更し、ＡＬＵの配置をかえることにより、マイクロ命令サイクルを100n sec程度短縮できるかもしれない。

その他として表には記さなかったが、ビットスライスプロセッサのレジスタファイルの利用状況も測定してみた。マイクロプログラムのコーディング経験を積むにつれて、レジスタファイルの利用率は上がっているようである。

以上から、設計段階において効率向上を期待して組み込んだ各種ハードウェアは、ほぼ期待どおりの働きをしていることが確認できた。なお、本節における評価のためのデータ源は、ＴＰＵ－６に片寄りすぎたきらいがあるが、少なくともハード利用の傾向程度は、十分に検討できたと考える。

Ⅴ．４　システムの総合評価とその考察

はじめに実行時間について考察する。ハードウェアの各部がそれぞれに効率向上に貢献していることは前節で明らかとなった。しかしながら実行時間を比べたとき、ＣＰＵのサイクルタイムが６倍も短い超大型計算機をしのぐことを説明しなければならない。これは半ば想像の域を出ないが、各ハードウェアの高度な機能によるステップ数の減少の他に、マイクロ命令コードを長くとったことにより、い

ろいろな部分での並列処理パイプライン処理が行われて、機械語命令では不可能な短いステップ数を実現したのではないかと考えられる。実際インタープリタ部分のステップ数は539ステップ（Ⅳ.6参照）しかなく、これはＬＩＳＰコンテストに参加した他のどのシステムよりも、極端に小さい値である。

次にシステムとして成功した部分と問題点についてまとめる。まずハードウェアについては、前節で示したとおり、ハードウェアスタック、ジャンプ機能、メモリアクセス機能、直接数値演算機能、マッピングメモリやフィールド抽出回路、高度の並列処理パイプライン処理機能等、いずれも成功したといえる。改善の余地のあるところとしては、ＡＬＵとバスの配置を最適化することによりさらに高速化できること、また問題点としてはマッピングメモリフラグのデコーダが固定で、ソフトの変更についてゆけないこと、スタックのアンダーフローの自動検出がないため、スタックを仮想化するためにはステップを食うことなどである。またソフトウェア面では、頻繁に現われるルーチンまたはループを小型にし、ＡＲＧＥＶＡＬなどのルーチンを新設したことが、十分に高速化に役立ったと考えられる。

さらに使いやすさや入出力の面では、まだエラー処理やデバッグ機能の面で改良の余地が大きいこと、ファイル機能を持たせること、インタラクティブなシステムに改良することなどがあり、また高速性を生かしたりアルタイム処理の応用などもおもしろいと考えられる。

次は32ビットマシンへの拡張性についてである。本シス

テムの構成のうち、32ビットマシンに適用できない部分としては、まずデータ表現がある。32ビットマシンとなれば、アドレス空間は少なくともメガワード単位となり、余ったビットにタグを付けることになる。するとタグを処理するためのフィールド抽出回路がよけいに必要となり、マッピングメモリの利用方法も異なってくると考えられる。またⅢ.5.1で述べたとおり、処理幅を倍にすると、ハードウェアの実装が著しく問題となり、研究室で実現できる規模を越える恐れがある。さらに演算時間がかかったり、メモリ容量の増大にともなってアクセス時間をよけいにとる必要があったり、メモリの仮想化の問題があったりする。一方ハードウェアスタックやジャンプ機能、直接数値演算、並列処理やパイプライン処理機能は、実装の問題を除けば利用可能と考えられる。

またバックアッププロセッサを置くことは有用であると考えられるが、実用上リード、プリントの処理は、すべてＬＩＳＰプロセッサ側でやらせた方が良いかもしれない。

最後に本システムとコンパイラについて考える。まずコンパイラを実現しようとすると、機械語命令または中間言語命令と命令コードを設定しなければならないが、本システムではメモリ参照があまり負担にならないこと、むしろ命令デコードにステップを食うことから、命令コードの小型化はあまり考えないようにし、デコードに時間を必要としないコードの使用を考えるべきである。また中間言語命令の設定には検討を要し、高度の並列処理やパイプライン処理がそこなわれないような命令体系を考察する必要がある。

一方コンパイラを実装した場合にどの程度の速度向上が期待されるかということであるが、たとえば表Ⅴ·3(b)をみると、システム関数の比率が56.3%もあり、その中からリターン処理にかかる6%を差し引いても、残りは約50%である。したがってコンパイラによって、インタープリタが行っていた処理ステップが完全におとされたとしても、速度向上は2倍にとどまる。実際には中間言語命令のフェッチとデコードにステップを食うためそれよりは悪化するはずである。これはTPUの場合であったが、TARAIのようにインタープリタルーチンのしめる割合が極端に大きい場合には、速度向上の割合も大きいと考えられる。しかしながら、TARAIと比較するとTPUプログラムの方が実際的であり、本システムにおいては、コンパイラを実装したとしても大きな速度の向上は望めないと予想される。

このことを逆から考えると、インタープリタによる処理速度がコンパイラに近づいたと言えるわけで、そこからコンパイラの必要性、中間言語の意味、高級言語との関係におけるマシンコードのあり方などの議論が新たに生まれてきそうであり、それは今後の課題として残しておきたいと考えている。

Ⅴ.5 結 言

LISPコンテストの問題プログラムの実行時間から、本システムは、他のどの大型計算機によるLISPよりも高速であることを示した。さらにプログラム実行の際の、マイクロプログラムの総実行ステップを求め、合わせてハードウェア機能の利用回数を求めることによって、ハード

ウェアスタック、各種ジャンプ機能、メモリオペレーション、直接数値演算機能、フィールド抽出やマッピングメモリ、並列処理やパイプライン処理機能が十分に利用され、処理速度向上に大きく貢献していることを示した。そして本システムが非常に速い理由として、高機能のハードウェアによるステップ数の短縮、インタープリタの改良によるステップ数の短縮、とりわけ高度の並列処理、パイプライン処理によるステップ数の短縮が大きく影響していると考えられることを述べた。またハードウェア構成上で少し残された問題点と、デバッグ機能や入出力機能の強化の必要性を示した。さらに、システムの拡張性やコンパイラについて述べ、コンパイラを実装しても大きな速度向上が望めないこと、本システムのインタープリタの処理速度がコンパイラに近づいていることを述べた。

第 6 章　　結　　論

複雑な階層構造を持つアルゴリズムを簡潔に表現できるＬＩＳＰ言語は、その反面において、実行時の処理負担が大きく、ＬＩＳＰマシンにはＬＩＳＰ処理向きのハードウェアが必要とされる。本研究では、市販のＬＳＩの使用とインタープリタのマイクロプログラム化を前提に、ソフトウェアからの要求をハードウェアに盛りこむことに注意を払い、ＬＩＳＰを高速で処理できるハードウェア、ファームウェア構造を持つ計算機システムの開発をめざした。そのもとで決定されたシステム構成は、ミニコンピュータを入出力とバックアップに用いる計算機複合体であり、またＬＩＳＰ処理の中心となって働くプロセッサモジュールには、ハードウェアスタック、強力なジャンプ機能、マッピングメモリやフィールド抽出回路、定数演算や並列処理パイプライン処理を可能にするマイクロ命令コードなどを装備し、処理の高速化を図った。またインタープリタの設計においては制御構成の改良と利用頻度の高いルーチンの高速化に努力し、ＡＲＧＥＶＡＬなどの新しいルーチンを導入した。こうして完成したＬＩＳＰマシンシステムは、ＬＩＳＰコンテストの問題を超大型計算機によるＬＩＳＰよりも高速に処理し、ハードウェア、ファームウェアの改良の効果が実証された。さらに高速性の理由をはっきりさせるため各ハードウェアの利用状況を実測し、いずれのハードウェアも、高速化に貢献していることを確認した。そして、本システムが、ＣＰＵサイクルタイムが６倍も短い超大型計算機にまさる性能を持つ原因として、それぞれ強力なハードウェアを、さらに高度な並列処理、パイプライ

ン処理で利用しているためであると結論した。また、本システムにおいて効果のあったハードウェア構造をより大規模システムに適用しようとした場合には、実装上の問題が増大すること、タグビットの取り扱いを検討しなおさねばならないことを示し、そのまま適用できない部分のあることを述べた。またソフトウェア上では、デバッグ機能や入出力機能に改良の余地のあることを示した。さらに本システムにコンパイラを実装することを検討し、本システムではコンパイラを用いても大きな速度向上が望めないこと、逆に本システムではインタープリタの速度がコンパイラに近づいていることを示し、コンパイラの必要性、高級言語とその中間言語あるいは機械語との関係というものに対し、新しい見解が持てそうであることを述べた。

謝　　辞

本研究は修士課程の研究としては十分に大きな規模であり、費用も人員も多くを必要とした。研究の機会を与えてくださった前川禎男教授には、心から感謝の意を表する。また直接の指導をいただいた金田悠紀夫助教授には、ひとかたならぬおせわになった。特に記して感謝の意を表する。またシステム工学科第四講座の小林康博氏、滝本博道氏、多田光弘氏には、ハードウェアならびにソフトウェアの製作において多大な努力を払っていただいた。心から感謝するとともに、ここにＬＩＳＰマシンシステムの完成を喜び合いたい。また和田耕一氏をはじめとする第四講座の諸氏と、神戸大学電子計算機研究会の諸氏には、ハードウェアの製作や図面作成などに協力いただいた。本システムを企画する段階においては、通産省工業技術院電子技術総合研究所の島田俊夫氏、山口喜教氏に有益な助言と御討論をいただいたことを記し、感謝の意を表する。

参考文献

[1] 瀧，金田，前川：LISPマシンの試作，情報処理学会研究会資料，計算機アーキテクチャ32-3，(Sep. 1978).

[2] 瀧，金田，前川：試作LISPマシンのインタープリタのマイクロプログラム化について，情報処理学会研究会資料，計算機アーキテクチャ33-2，(Nov. 1978).

[3] 瀧，金田，前川：試作LISPマシンとその評価，情報処理学会研究会資料，記号処理7-3，(Mar. 1979).

[4] 瀧本：試作LISPマシンのマイクロプログラム化インタープリタに関する研究，神戸大学工学部システム工学科卒業論文，(1979).

[5] 山口，島田：仮想計算機によるLISPプログラムの動的特性，信学 論文誌D，J61-D，8，(Aug. 1978).

[6] 島田，山口，坂村：LISPマシンとその評価，信学論文誌D，J59-D，6，(June. 1976).

[7] 竹内：LISP処理系コンテストの結果，情報処理学会研究会資料，記号処理5-3，(Aug. 1978).

[8] 中西：LISP入門，近代科学社，(1977).

[9] 黒川：LISPのデータ表現，情報処理 Vol. 17 No. 2，(Feb. 1976).

[10] 山口，島田，他：ACE上のLISPマシン，信学 研，EC-76-13，(May. 1976).

[11] 長尾，中村，他：LISPマシンNK3のアーキテクチャとそのマイクロインストラクション，信学 研，EC-77-17，(1977).

[12] 長尾，中村：高速補助記憶装置を使用したミニコン用LISP 1.6 システム，情報処理 Vol. 17 No. 8，(1976).

[13] W.Teitelman : INTERLISP Reference Manual, Xerox, (Feb. 1974).

[14] D.A.Moon : MACLISP Reference Manual, Project MAC, MIT, (Apr. 1974).

[15] LISP 1.9 User's Manual, EPICS-5-ON-2, ETL.

[16] M.Terashima : Algorithms Used in An Implementation of HLISP, Faculty of Science University of Tokyo, Technical Report, (Jan. 1975).

[17] R.Greenblatt : The LISP Machine, MIT AI Lab. Working Paper 79, (Nov. 1974).

[18] Tom Knight : CONS, MIT AI Lab. Working Paper 80, (Nov. 1974).

[19] D.G.Bobrow : A Model and Stack Implementation of Mutiple Environments. C.ACM Vol.16 No.10, (1973).

[20] Aian Bawden et al : LISP Machine Progress Report, MIT AI Lab. Memo No.444, (Aug. 1977).

[21] STACK MACHINES, IEEE, COMPUTER Vol.10 No.5, (May 1977).

[22] The Am2900 Family Data Book, Advanced Micro Devices, Inc.

[23] J.MacCarthy et al : LISP 1.5 Programmer's Manual, MIT Press, (1966).

付録1. 複合伝搬遅延 (combinational delays)

プロセッサモジュール内のレジスタ、バス、ALU相互の、信号の伝搬遅延時間、セットアップ時間についてまとめる。値はICメーカーの保証する最大値の和であり、配線による遅延の分を含まない。

Table A1-1. Combinational propagation delays from rising edge of ϕ_3.

from input	STACK	IMDR*	STP	MAP	reg.file	imm.data
to Y-bus (n sec)	254	188	179	224	165	159
to ALU flag (n sec)	285	219	210	255	196	190

* ICAR, ICDR

Table A1-2. Combinational set up times from Y-bus.

to set up	registers	OMDR*	NIL flag	STACK	reg.file	JUMP
set up time (n sec)	20	26	40	41	18	192

* OCAR, OCDR

以下にCYCの取扱いについてまとめる。

STACK → ALU → Y-bus : CYC = 1
MAP → ALU → Y-bus : CYC = 0
STACK → ALU → Y-bus → JUMP : CYC = 2 *
MAP → ALU → Y-bus → JUMP : CYC = 2
MAP flag jump : CYC = 1

CYCの取扱いは、伝搬遅延時間の値に余裕をみて決定した。*印は、ALUオペレーションのうちPASのみ許す。

付録2. システムの全景

メンテナンスパネル
電源ユニット

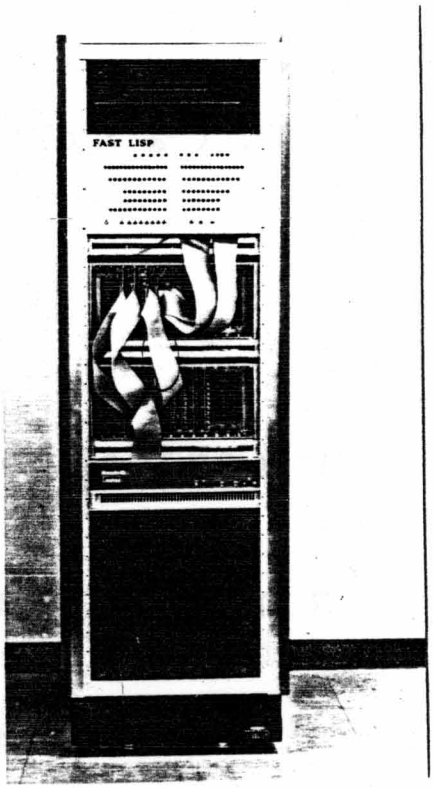
プロセッサモジュール

メモリモジュール

LSI-11

キャビネット サイズ

60cm(横) × 177cm(高さ) × 74cm(奥行)



図A2-1. LISPプロセッサと
LSI-11

図A2-2. PDC-80 マイクロコンピュータ
と入出力器機群

付録3. モニタコマンド 一覧

LSI-11上のLISPモニタのコマンド名と取扱い方について述べる。コマンド名はつぎのとおりである。

通常使用コマンド.

G	---	JOB START
(CTRL) P	---	JOB HALT
R	---	JOB RESTART
I	---	SYSTEM PARAMETER INITIALIZE

デバッグ用コマンド.

L	---	LISP MACHINE MAIN MEMORY R/W
W	---	WCS R/W
M	---	MAPPING MEMORY R/W
CM	---	CMR R/W
PG	---	PAGE REGISTER R/W
S	---	MICROPROGRAM STEP EXECUTION
B	---	マイクロプログラムブレークポイント設定と実行
N	---	ブレークポイント再設定と再実行
CL	---	MICROPROGRAM STEP-COUNTER CLEAR
PR	---	MICROPROGRAM STEP-COUNTER PRINT

イニシャルローダにより、LISPプロセッサへのローディングが終了すると、コンソールタイプライタに'/'スラッシュがプリントされる。スラッシュが出ている間がコマンド待ちの状態である。

以下、コマンドの主なものについて説明を加える。

/G : Gのキーインにより、プログラムモードでゼロ番地から実行開始となる。パラメータの初期

化などがすべて行なわれる。すなわちLISPの各ジョブのスタートコマンドである。

/R : (CTRL)Pにて停止したジョブを再開させる。入出力待ち以外で停止した時に限る。

/I : Iのキーインにより、マイクロプログラムの実行番地をゼロにセットする。

/L : Lのキーイン後、アドレスを16進4桁で入力すると、その内容がプリントされる。次の入力は3とおりに分かれる。

(CR) ··· コマンドモードにもどる。

(LF) ··· 次のアドレスの内容のプリント。

16進入力 ··· cdr部、car部の順に16進8桁を入力すると、メモリ内容が変更できる。その後(CR)か(LF)を入力する。

なおメモリ内容がプリントされたのち、CまたはDを入力すると、car部またはcdr部のみの書き替えができる。

/W : 16進14桁でWCS内容のプリントおよび書きかえのための入力を行なう他は、Lコマンドに準ずる。

/M : データのプリントと入力は、16進4桁で行なう。他はLコマンドに準ずる。

/S : Sのキーインにより、マイクロプログラムを1ステップ(メモリ参照時は2ステップ)だけ実行して、次の実行アドレスをプリントしコマンド待ちとなる。

/B : Bのキーインの後、停止させたいアドレスを16進4けたで入力すると、ただちにゼロ番地よ

り連続ステップモードで実行が開始される。停止アドレスに達すると、コマンド待ちとなる。

/N　：　Bコマンドに準ずるが、停止した番地から実行の再開となる。

その他のコマンドや操作方法の詳細については、システム工学科第4講座内部資料「LISPマシンオペレーションズマニュアル」を参照されたい。

```
;Fig. 1
;* * *  BITA  * * *

DEFINE ((

(BIT (LAMBDA
       (A)
       (COND ((NULL (CDR A)) A)
             ((NULL (CDDR A))
              (LIST (CONS (CAR A) (CONS (QUOTE $) (CDR A)))) )
             (T (BITL (CDR A) (LIST (CAR A)))) )))

(BITL (LAMBDA
        (X J)
        (COND ((NULL X) NIL)
              (T (NCONC
                   (MAPAPPEND (BIT X)
                              (FUNCTION (LAMBDA
                                          (K)
        ;++++++++++++++++++++++++++++++++++++++++++++++++++
(MAPCAR (BIT J) (FUNCTION (LAMBDA (L) (LIST L (QUOTE $) K))))  )))
        ;--------------------------------------------------
(BITL (CDR X)(APPEND J (LIST (CAR X))))                        ))))

(MAPAPPEND (LAMBDA
             (X FN)
             (COND ((NULL X) NIL)
                   (T (NCONC (FN (CAR X)) (MAPAPPEND (CDR X) FN))) )))

))
```

```
;Fig. 2
;* * *  BITB  * * *

DEFINE ((

(BIT (LAMBDA
       (A)
       (COND ((NULL A) NIL)
             ((NULL (CDR A)) A)
             (T (MAPCON
                  (BIT (CDR A))
                  (PROG2 (SETQ A (LIST (CAR A) (QUOTE $) (CADR A)))
                         (FUNCTION (LAMBDA (B) (G (CAR B)))) ))))))
```

Fig. 2 (continued)

```
(G (LAMBDA
     (B)
     (COND ((ATOM B) (LIST A))
           (T (CONS
                (LIST (CAR A) (QUOTE $) B)
                (MAPCAR (G (CAR B))
                        (FUNCTION (LAMBDA (A) (CONS A (CDR B)))) ))))))

))
```

```
;Fig. 3
;* * *  SEQUENCE or SORT  * * *

(DE EQUAL (X Y)
 (COND ((ATOM X) (EQ X Y))
       ((ATOM Y) NIL)
       ((EQUAL (CAR X) (CAR Y)) (EQUAL (CDR X) (CDR Y)))
       (T NIL) ))

(DE APPEND (X Y)
 (COND ((NULL X) Y) (T (CONS (CAR X) (APPEND (CDR X) Y)))) )

(DE SMALLER (X Y) (COND ((GREATERP Y X) X) (T Y)))

(DE MINM (L)
 (COND ((NULL L) NIL)
       ((NULL (CDR L)) (CAR L))
       (T (SMALLER (CAR L) (MINM (CDR L)))) ))

(DE DIFFLIST (A X)
 (COND ((NULL X) NIL)
       ((EQUAL A (CAR X)) (DIFFLIST A (CDR X)))
       (T (CONS (CAR X) (DIFFLIST A (CDR X)))) ))

(DE SEQUENCE (L)
 (PROG (U V W)
       (SETQ U L)
       (SETQ V (MINM L))
       (SETQ W NIL)
  A    (COND ((NULL U) (RETURN W)))
       (SETQ V (MINM U))
       (SETQ U (DIFFLIST V U))
       (SETQ W (APPEND W (LIST V)))
       (GO A) ))
```

```
;Fig. 4
;* * *  Data for sort program  * * *


(11 17 3 16 17 3 19 32 22 39 31 38 46 29 9 40 7 31 21 43)

(24 0 35 5 15 43 21 31 0 17 47 26 47 26 32 2 25 28 45 9 2
 29 4 6 48 36 28 48 31 3 37 42 20 38 48 44 33 47 31 11 )

(40 36 4 49 9 11 35 13 8 29 45 11 8 45 1 48 25 17 23 31 30
 0 28 16 41 1 36 9 26 25 16 16 4 30 35 44 45 20 14 49 17 4
 21 38 38 36 24 15 27 23 42 39 4 22 44 14 37 38 49 48 )

(44 28 26 46 44 49 42 15 7 5 18 11 0 4 21 39 43 2 21 8 5 5
 36 22 3 19 35 38 7 1 37 12 36 7 11 1 9 42 24 10 43 15 47
 47 8 29 47 16 23 43 44 25 7 11 6 30 27 41 2 37 3 39 0 1 4
 10 20 31 48 11 31 31 5 49 0 8 42 29 47 18 )

(29 9 41 18 34 43 0 13 26 38 43 10 24 1 38 16 47 38 5 36 13
 5 13 26 41 9 29 44 0 5 23 45 9 49 14 35 46 49 19 21 0 10 9
 9 20 39 1 5 15 42 15 16 3 24 20 47 3 43 33 5 34 5 24 46 14
 15 11 27 13 31 14 4 49 7 0 32 41 2 40 19 7 20 0 20 17 18 1
 42 39 8 11 41 42 30 49 23 45 6 33 45 )
```

```
;Fig. 5
;* * *  TARAI  * * *


(DE TARAI (X Y Z)
 (COND
  ((GREATERP X Y)
   (TARAI (TARAI (SUB1 X) Y Z)
          (TARAI (SUB1 Y) Z X)
          (TARAI (SUB1 Z) X Y) ))
  (T Y) ))
```

```
;Fig. 6
;* * *   TPU program by C.L. Chang  * * *


(PROG ()
 (DEFPROP XLIST (X1 X2 X3 X4 X5 X6 X7) VALUE)
 (DEFPROP YLIST (Y1 Y2 Y3 Y4 Y5 Y6 Y7) VALUE)
 (DEFPROP ZLIST (ZZ1 ZZ2 ZZ3 ZZ4 ZZ5 ZZ6 ZZ7) VALUE) )

(DE RENAME (C XY)
 (PROG (VAR Z)
       (SETQ Z ZLIST)
       (SETQ VAR (CADR C))
  B1   (COND ((NULL VAR) (GO B2)))
       (SETQ C (SUBST (CAR Z) (CAR VAR) C))
       (SETQ Z (CDR Z))
       (SETQ VAR (CDR VAR))
       (GO B1)
  B2   (SETQ Z XY)
       (SETQ VAR (CADR C))
  B3   (COND ((NULL VAR) (RETURN C)))
       (SETQ C (SUBST (CAR Z) (CAR VAR) C))
       (SETQ Z (CDR Z))
       (SETQ VAR (CDR VAR))
       (GO B3) ))

(DE INSIDE (A E)
 (COND ((ATOM E) (EQ A E))
       ((INSIDE A (CAR E)) T)
       (T (INSIDE A (CDR E))) ))

(DE DISAGREE (E1 E2)
 (COND ((NULL E1) NIL)
       ((OR (ATOM E1) (ATOM E2))
        (COND ((EQUAL E1 E2) NIL) (T (LIST E1 E2))) )
       ((EQUAL (CAR E1) (CAR E2)) (DISAGREE (CDR E1) (CDR E2)))
       ((OR (ATOM (CAR E1)) (ATOM (CAR E2))) (LIST (CAR E1) (CAR E2)))
       (T (DISAGREE (CAR E1) (CAR E2))) ))
```

Fig. 6 (continued)

```
(DE UNIFICATION (E1 E2)
 (PROG (D U D1 D2)
       (COND
        ((NOT (EQUAL (LENGTH E1) (LENGTH E2))) (RETURN (QUOTE NO))) )
  B1   (SETQ D (DISAGREE E1 E2))
       (COND ((NULL D) (RETURN (REVERSE U))))
       (SETQ D1 (CAR D))
       (SETQ D2 (CADR D))
       (COND ((OR (MEMBER D1 XLIST) (MEMBER D1 YLIST)) (GO B3)))
       (COND ((OR (MEMBER D2 XLIST) (MEMBER D2 YLIST)) (GO B4)))
  B2   (RETURN (QUOTE NO))
  B3   (COND ((INSIDE D1 D2) (GO B2)))
       (SETQ U (CONS D U))
       (SETQ E1 (SUBST D2 D1 E1))
       (SETQ E2 (SUBST D2 D1 E2))
       (GO B1)
  B4   (COND ((INSIDE D2 D1) (GO B2)))
       (SETQ U (CONS (REVERSE D) U))
       (SETQ E1 (SUBST D1 D2 E1))
       (SETQ E2 (SUBST D1 D2 E2))
       (GO B1) ))

(DE DELETEV (X Y VAR)
 (PROG (VAR1 TX TX1 X1)
       (SETQ X (APPEND X Y))
  B1   (COND ((NULL VAR) (RETURN X)))
       (SETQ VAR1 (CAR VAR))
       (SETQ TX X)
       (SETQ X1 NIL)
  B2   (COND ((NULL TX) (GO B4)))
       (SETQ TX1 (CAR TX))
       (COND ((EQ TX1 VAR1) (GO B3)))
       (SETQ X1 (CONS TX1 X1))
       (SETQ TX (CDR TX))
       (GO B2)
  B3   (SETQ X (APPEND X1 (CDR TX)))
  B4   (SETQ VAR (CDR VAR))
       (GO B1) ))
```

Fig. 6 (continued)

```
(DE URESOLVE (C1 C2 N)
 (PROG (L1 L2 VC1 VC2 X Y SIGN UNIF R RES VAR V1 V2 H HIST TC2)
       (SETQ C1 (RENAME C1 XLIST))
       (SETQ C2 (RENAME C2 YLIST))
       (SETQ L1 (CAR C1))
       (SETQ L2 (CAR C2))
       (SETQ VC1 (CADR C1))
       (SETQ VC2 (CADR C2))
       (SETQ C2 (CADDR C2))
       (SETQ X (CAR (CADDR C1)))
       (SETQ SIGN -1)
       (COND ((EQ (CAR X) (QUOTE NOT)) (GO B7)))
       (SETQ SIGN 1)
  B1   (COND
        ((NULL C2) (RETURN (LIST (REVERSE RES) (REVERSE HIST) N))) )
       (SETQ Y (CAR C2))
       (COND ((EQ (CAR Y) (QUOTE NOT)) (GO B2)))
       (GO B6)
  B2   (SETQ UNIF (UNIFICATION X (CDR Y)))
  B3   (COND ((EQUAL UNIF (QUOTE NO)) (GO B6)))
       (SETQ R (APPEND (REVERSE TC2) (CDR C2)))
       (COND ((NULL R) (RETURN (LIST (QUOTE CONTRADICTION) L1 L2))))
       (SETQ VAR NIL)
  B4   (COND ((NULL UNIF) (GO B5)))
       (SETQ V1 (CAAR UNIF))
       (SETQ V2 (CADAR UNIF))
       (SETQ VAR (CONS V1 VAR))
       (SETQ R (SUBST V2 V1 R))
       (SETQ UNIF (CDR UNIF))
       (GO B4)
  B5   (SETQ N (ADD1 N))
       (SETQ H (LIST N L1 L2 (ADD1 (LENGTH TC2))))
       (SETQ R (LIST N (DELETEV VC1 VC2 VAR) R))
       (SETQ RES (CONS R RES))
       (SETQ HIST (CONS H HIST))
  B6   (SETQ TC2 (CONS Y TC2))
       (SETQ C2 (CDR C2))
       (COND ((EQUAL SIGN 1) (GO B1)))
  B7   (COND
        ((NULL C2) (RETURN (LIST (REVERSE RES) (REVERSE HIST) N))) )
       (SETQ Y (CAR C2))
       (COND ((EQ (CAR Y) (QUOTE NOT)) (GO B6)))
       (SETQ UNIF (UNIFICATION (CDR X) Y))
       (GO B3) ))
```

Fig. 6 (continued)

```
(DE GUNIT (S1 S2 W C N)
 (PROG (L S3 SS3 W1 V U RES HIST M X)
       (COND ((NULL W) (RETURN (LIST RES HIST N))))
       (SETQ L (LENGTH (CADDR C)))
       (SETQ S3 (LIST (LIST 10000 C)))
       (SETQ SS3 S3)
  B1   (COND ((NULL W) (GO B7)))
       (SETQ W1 (CAR W))
  B2   (COND ((NULL SS3) (GO B4)))
       (SETQ V (CAR SS3))
       (COND ((GREATERP (CAR W1) (CAR V)) (GO B3)))
       (SETQ U (URESOLVE W1 (CADR V) N))
       (COND ((NULL (CAR U)) (GO B3)))
       (SETQ RES (APPEND RES (CAR U)))
       (SETQ HIST (APPEND HIST (CADR U)))
       (SETQ N (CADDR U))
  B3   (SETQ SS3 (CDR SS3))
       (GO B2)
  B4   (COND ((EQUAL (SUB1 L) 1) (GO B6)))
       (SETQ M (CAR W1))
  B5   (COND ((NULL RES) (GO B6)))
       (SETQ X (CONS (LIST M (CAR RES)) X))
       (SETQ RES (CDR RES))
       (GO B5)
  B6   (SETQ W (CDR W))
       (SETQ SS3 S3)
       (GO B1)
  B7   (SETQ L (SUB1 L))
       (COND ((EQUAL L 1) (RETURN (LIST RES HIST N))))
       (SETQ S3 X)
       (SETQ SS3 S3)
       (SETQ X NIL)
       (SETQ W (APPEND S1 S2))
       (GO B1) ))

(DE PNSORT (RES)
 (PROG (C POS NEG)
  B1   (COND ((NULL RES) (RETURN (LIST (REVERSE POS) (REVERSE NEG)))))
       (SETQ C (CAAR (CDDAR RES)))
       (COND ((EQUAL (CAR C) (QUOTE NOT)) (GO B3)))
       (SETQ POS (CONS (CAR RES) POS))
  B2   (SETQ RES (CDR RES))
       (GO B1)
  B3   (SETQ NEG (CONS (CAR RES) NEG))
       (GO B2) ))
```

Fig. 6 (continued)

```
(DE FDEPTH (C)
 (PROG (N U)
       (SETQ C (CAR (CADDR C)))
       (COND ((EQUAL (CAR C) (QUOTE NOT)) (GO B1)))
       (SETQ C (CDR C))
       (GO B2)
  B1   (SETQ C (CDDR C))
  B2   (SETQ N 0)
  B3   (COND ((NULL C) (GO B5)))
       (COND ((ATOM (CAR C)) (GO B4)))
       (SETQ U (APPEND (CDAR C) U))
  B4   (SETQ C (CDR C))
       (GO B3)
  B5   (COND ((NULL U) (RETURN N)))
       (SETQ N (ADD1 N))
       (SETQ C U)
       (SETQ U NIL)
       (GO B3) ))


(DE FTEST (RES N4)
 (PROG (C U)
  B1   (COND ((NULL RES) (RETURN (REVERSE U))))
       (SETQ C (CAR RES))
       (COND ((GREATERP (FDEPTH C) N4) (GO B2)))
       (SETQ U (CONS C U))
  B2   (SETQ RES (CDR RES))
       (GO B1) ))

(DE SUBSUME (C1 C2)
 (PROG (Z VAR U)
       (SETQ C1 (RENAME C1 XLIST))
       (SETQ C1 (CAR (CADDR C1)))
       (SETQ Z ZLIST)
       (SETQ VAR (CADR C2))
       (SETQ C2 (CAR (CADDR C2)))
  B1   (COND ((NULL VAR) (GO B2)))
       (SETQ C2 (SUBST (CAR Z) (CAR VAR) C2))
       (SETQ VAR (CDR VAR))
       (GO B1)
  B2   (SETQ U (UNIFICATION C1 C2))
       (COND ((EQUAL U (QUOTE NO)) (RETURN NIL)))
       (RETURN T) ))
```

Fig. 6 (continued)

```
(DE STEST (U RES)
 (PROG (R V W X1 Y Z)
  B1    (COND ((NULL RES) (GO B5)))
        (SETQ R (CAR RES))
        (SETQ Z (APPEND U V))
  B2    (COND ((NULL Z) (GO B3)))
        (COND ((SUBSUME (CAR Z) R) (GO B4)))
        (SETQ Z (CDR Z))
        (GO B2)
  B3    (SETQ V (CONS R V))
  B4    (SETQ RES (CDR RES))
        (GO B1)
  B5    (COND ((NULL V) (RETURN W)))
        (SETQ X1 (CAR V))
        (SETQ Z (CDR V))
  B6    (COND ((NULL Z) (GO B8)))
        (COND ((SUBSUME X1 (CAR Z)) (GO B7)))
        (SETQ Y (CONS (CAR Z) Y))
  B7    (SETQ Z (CDR Z))
        (GO B6)
  B8    (SETQ W (CONS X1 W))
        (SETQ V (REVERSE Y))
        (SETQ Y NIL)
        (GO B5) ))

(DE CONTRADICT (U V)
 (PROG (X1 Y RES)
  B1    (COND ((OR (NULL U) (NULL V)) (RETURN NIL)))
        (SETQ X1 (CAR U))
        (SETQ Y V)
  B2    (COND ((NULL Y) (GO B3)))
        (SETQ RES (URESOLVE X1 (CAR Y) -1))
        (COND ((EQUAL (CAR RES) (QUOTE CONTRADICTION)) (RETURN RES)))
        (SETQ Y (CDR Y))
        (GO B2)
  B3    (SETQ U (CDR U))
        (GO B1) ))
```

Fig. 6 (continued)

```
(DE DTREE (Z HIST N1)
 (PROG (X TX X1 H M1 M2 M N)
       (SETQ HIST (REVERSE HIST))
       (SETQ X (CDR Z))
       (SETQ Z (LIST Z))
       (COND ((GREATERP (CAR X) (CADR X)) (GO B0)))
       (SETQ X (REVERSE X))
  B0   (COND ((GREATERP (CADR X) N1) (GO B1)))
       (SETQ X (LIST (CAR X)))
  B1   (COND ((NULL X) (RETURN Z)))
       (SETQ X1 (CAR X))
  B2   (COND ((EQUAL X1 (CAAR HIST)) (GO B3)))
       (SETQ HIST (CDR HIST))
       (GO B2)
  B3   (SETQ X (CDR X))
       (SETQ H (CAR HIST))
       (SETQ Z (CONS H Z))
       (SETQ HIST (CDR HIST))
       (SETQ M1 (CADR H))
       (SETQ M2 (CADDR H))
       (COND ((GREATERP M1 N1) (GO B5)))
  B4   (COND ((GREATERP M2 N1) (GO B6)))
       (GO B1)
  B5   (SETQ N 1)
       (SETQ M M1)
       (GO B7)
  B6   (SETQ N 2)
       (SETQ M M2)
  B7   (COND ((NULL X) (GO B8)))
       (SETQ X1 (CAR X))
       (COND ((EQUAL X1 M) (GO B10)))
       (COND ((GREATERP X1 M) (GO B9)))
  B8   (SETQ X (APPEND (REVERSE TX) (CONS M X)))
       (GO B11)
  B9   (SETQ TX (CONS X1 TX))
       (SETQ X (CDR X))
       (GO B7)
  B10  (SETQ X (APPEND (REVERSE TX) X))
  B11  (SETQ TX NIL)
       (COND ((EQUAL N 2) (GO B1)))
       (GO B4) ))
```

Fig. 6 (continued)

```
(DE TPU (S1 S2 S3 W N1 N2 N3 N4)
 (PROG (S W1 TS U1 U N K CK WCK V POS NEG HIST Y X1 X)
       (SETQ S (APPEND S1 S2))
       (SETQ S (REVERSE S))
  B1   (COND ((NULL W) (GO B6)))
       (SETQ W1 (CAR W))
  B2   (SETQ TS S)
       (COND ((NULL W1) (GO B5)))
  B3   (COND ((EQ (CAR W1) (CAAR TS)) (GO B4)))
       (SETQ TS (CDR TS))
       (GO B3)
  B4   (SETQ U1 (CONS (CAR TS) U1))
       (SETQ W1 (CDR W1))
       (GO B2)
  B5   (SETQ U (CONS U1 U))
       (SETQ W (CDR W))
       (SETQ U1 NIL)
       (GO B1)
  B6   (SETQ W (REVERSE U))
       (SETQ N N1)
       (SETQ U (CONTRADICT S1 S2))
       (COND ((NOT (NULL U)) (RETURN U)))
       (SETQ K 1)
  B7   (COND ((GREATERP K N2) (RETURN (QUOTE (S IS NOT PROVED)))))
       (SETQ CK (CAR S3))
       (SETQ WCK (CAR W))
       (SETQ V (GUNIT S1 S2 WCK CK N))
       (COND ((NULL (CAR V)) (GO B12)))
       (SETQ N (CADDR V))
       (SETQ HIST (APPEND HIST (CADR V)))
       (SETQ V (CAR V))
       (COND ((LESSP K N3) (GO B8)))
       (SETQ V (FTEST V N4))
  B8   (SETQ V (PNSORT V))
       (SETQ POS (STEST S1 (CAR V)))
       (SETQ NEG (STEST S2 (CADR V)))
       (COND ((NULL (APPEND POS NEG)) (GO B12)))
       (SETQ U (CONTRADICT S1 NEG))
       (COND ((NOT (NULL U)) (RETURN (DTREE U HIST N1))))
       (SETQ U (CONTRADICT POS S2))
       (COND ((NOT (NULL U)) (RETURN (DTREE U HIST N1))))
       (SETQ S1 (APPEND S1 POS))
       (SETQ S2 (APPEND S2 NEG))
       (SETQ W (CDR W))
       (SETQ Y (APPEND POS NEG))
```

Fig. 6 (continued)

```
  B9   (COND ((NULL W) (GO B10)))
       (SETQ X1 (APPEND Y (CAR W)))
       (SETQ X (CONS X1 X))
       (SETQ W (CDR W))
       (GO B9)
  B10  (SETQ W (APPEND (REVERSE X) (LIST Y)))
       (SETQ X NIL)
  B11  (SETQ S3 (APPEND (CDR S3) (LIST CK)))
       (SETQ K (ADD1 K))
       (GO B7)
  B12  (SETQ W (APPEND (CDR W) (LIST NIL)))
       (GO B11) ))
```

```
;Fig. 7
;* * *  Nine problems to be solved by the TPU program  * * *


;TPU-1
(TPU (QUOTE ((1 (X Y) ((P (G X Y) X Y))) (2 (X Y) ((P X (H X Y) Y)))))
     (QUOTE ((3 (X) ((NOT P (K X) X (K X))))))
     (QUOTE
      ((4
        (X Y Z U V W)
        ((NOT P X Y U) (NOT P Y Z V) (NOT P X V W) (P U Z W)) )
       (5 (X Y Z U V W)
          ((NOT P X Y U) (NOT P Y Z V) (NOT P U Z W) (P X V W)) )))
     (QUOTE ((3) NIL))
 5 2 3 0 )

;TPU-2
(TPU (QUOTE
      ((1 (X) ((P E X X)))
       (2 (X) ((P X E X)))
       (3 (X) ((P X X E)))
       (4 NIL ((P A B C))) ))
     (QUOTE ((5 NIL ((NOT P B A C)))))
     (QUOTE
      ((6
        (X Y Z U V W)
        ((NOT P X Y U) (NOT P Y Z V) (NOT P X V W) (P U Z W)) )
       (7 (X Y Z U V W)
          ((NOT P X Y U) (NOT P Y Z V) (NOT P U Z W) (P X V W)) )))
     (QUOTE ((4) NIL))
 7 4 5 0 )
```

Fig. 7 (continued)

```
;TPU-3
(TPU (QUOTE ((1 (X) ((P E X X))) (2 (X) ((P (I X) X E)))))
     (QUOTE ((3 NIL ((NOT P A E A)))))
     (QUOTE
      ((4
        (X Y Z U V W)
        ((NOT P X Y U) (NOT P Y Z V) (NOT P U Z W) (P U Z W)) )
       (5 (X Y Z U V W)
          ((NOT P X Y U) (NOT P Y Z V) (NOT P U Z W) (P X V W)) )))
     (QUOTE ((3) (3)))
 5 4 5 0 )

;TPU-4
(TPU (QUOTE ((1 (X) ((P E X X))) (2 (X) ((P (I X) X E)))))
     (QUOTE ((3 (X) ((NOT P A X E)))))
     (QUOTE
      ((4
        (X Y Z U V W)
        ((NOT P X Y U) (NOT P Y Z V) (NOT P X V W) (P U Z W)) )
       (5 (X Y Z U V W)
          ((NOT P X Y U) (NOT P Y Z V) (NOT P U Z W) (P X V W)) )))
     (QUOTE ((3) (3)))
 5 4 5 0 )

;TPU-5
(TPU (QUOTE
      ((1 (X) ((P E X X)))
       (2 (X) ((P X W X)))
       (3 (X) ((P X (I X) E)))
       (4 (X) ((P (I X) X E)))
       (5 NIL ((S A))) ))
     (QUOTE ((6 NIL ((NOT S E)))))
     (QUOTE
      ((7 (X Y Z) ((NOT S X) (NOT S Y) (NOT P X (I Y) Z) (S Z)))
       (8 (X Y Z U V W)
          ((NOT P X Y U) (NOT P Y Z V) (NOT P X V W) (P U Z W)) )
       (9 (X Y Z U V W)
          ((NOT P X Y U) (NOT P Y Z V) (NOT P U Z W) (P X V W)) )))
     (QUOTE ((6) NIL NIL))
 9 4 5 0 )
```

Fig. 7 (continued)

```
;TPU-6
(TPU (QUOTE
      ((1 (X) ((P E X X)))
       (2 (X) ((P X E X)))
       (3 (X) ((P X (I X) E)))
       (4 (X) ((P (I X) X E)))
       (5 NIL ((S B))) ))
     (QUOTE ((6 NIL ((NOT S (I B))))))
     (QUOTE
      ((7 (X Y Z) ((NOT S X) (NOT S Y) (NOT P X (I Y) Z) (S Z)))
       (8 (X Y Z U V W)
          ((NOT P X Y U) (NOT P Y Z V) (NOT P X V W) (P U Z W)) )
       (9 (X Y Z U V W)
          ((NOT P X Y U) (NOT P Y Z V) (NOT P U Z W) (P X V W)) )))
     (QUOTE ((5 6) NIL NIL))
 9 4 5 0 )

;TPU-7
(TPU (QUOTE
      ((1 NIL ((P A)))
       (2 NIL ((M A (S C) (S B))))
       (3 (X) ((M X X (S X)))) ))
     (QUOTE ((4 NIL ((NOT D A B)))))
     (QUOTE
      ((5 (X Y Z) ((NOT M X Y Z) (M Y X Z)))
       (6 (X Y Z) ((NOT M X Y Z) (D X Z)))
       (7 (X Y Z U)
          ((NOT P X) (NOT M Y Z U) (NOT D X U) (D X Y) (D X Z)) )))
     (QUOTE ((1 2 3 4) (1 2 3 4) (1 2 3 4)))
 7 4 5 0 )

;TPU-8
(TPU (QUOTE ((1 NIL ((L 1 A))) (2 (X) ((D X X)))))
 NIL (QUOTE
      ((3 (X) ((P X) (D (G X) X)))
       (4 (X) ((P X) (L 1 (G X))))
       (5 (X) ((P X) (L (G X) X)))
       (6 (X) ((NOT P X) (NOT D X A)))
       (7 (X Y Z) ((NOT D X Y) (NOT D Y Z) (D X Z)))
       (8 (X) ((NOT L 1 X) (NOT L X A) (P (F X))))
       (9 (X) ((NOT L 1 X) (NOT L X A) (D (F X) X))) ))
     (QUOTE ((1 2) (1 2) (1 2) (1 2) (1 2) (1 2) (1 2)))
 9 20 21 0 )
```

Fig. 7 (continued)

```
;TPU-9
(TPU (QUOTE ((1 (X) ((L X (F X))))))
     (QUOTE ((2 (X) ((NOT L X X)))))
     (QUOTE
      ((3 (X Y) ((NOT L X Y) (NOT L Y X)))
       (4 (X Y) ((NOT D X (F Y)) (L Y X)))
       (5 (X) ((P X) (D (H X) X)))
       (6 (X) ((P X) (P (H X))))
       (7 (X) ((P X) (L (H X) X)))
       (8 (X) ((NOT P X) (NOT L A X) (L (F A) X))) ))
     (QUOTE ((1 2) (1 2) (1 2) (1 2) (1 2) (1 2)))
 8 20 21 0 )
```

```
;Fig. 8
;* * *  Expected answers  * * *

;TPU-1
((6 3 4 4)
 (11 2 6 2)
 (15 1 11 1)
 (CONTRADICTION 1 15) )

;TPU-2
((8 4 6 1)
 (21 3 8 1)
 (30 2 21 1)
 (32 30 7 2)
 (42 3 32 1)
 (55 1 42 1)
 (62 55 6 1)
 (112 5 62 3)
 (130 3 112 1)
 (CONTRADICTION 2 130) )

;TPU-3
((13 3 5 4)
 (16 2 13 2)
 (17 1 16 2)
 (18 17 4 4)
 (23 2 18 3)
 (24 1 23 2)
 (30 24 5 4)
 (46 2 30 2)
 (56 2 46 1)
 (CONTRADICTION 1 56) )

;TPU-4
((6 3 4 4)
 (11 2 6 3)
 (12 1 11 2)
 (20 12 5 4)
 (42 2 20 2)
 (62 2 42 1)
 (CONTRADICTION 1 62) )

;TPU-5
((10 6 7 4)
 (14 5 10 2)
 (18 5 14 1)
 (CONTRADICTION 3 18) )

;TPU-6
((11 5 7 1)
 (19 5 11 1)
 (23 3 19 1)
 (152 23 7 1)
 (169 6 152 3)
 (186 5 169 1)
 (CONTRADICTION 1 186) )

;TPU-7
((13 2 6 1)
 (16 13 7 3)
 (43 4 16 3)
 (66 4 43 3)
 (75 3 66 2)
 (CONTRADICTION 1 75) )

;TPU-8
((10 2 6 2)
 (15 10 3 1)
 (16 10 4 1)
 (17 10 5 1)
 (23 17 8 2)
 (25 16 23 1)
 (26 17 9 2)
 (28 16 26 1)
 (32 25 6 1)
 (33 32 7 3)
 (47 28 33 1)
 (CONTRADICTION 15 47) )

;TPU-9
((14 2 8 3)
 (16 1 14 2)
 (17 16 5 1)
 (18 16 6 1)
 (19 16 7 1)
 (20 19 3 1)
 (23 17 4 1)
 (24 23 8 2)
 (28 20 24 2)
 (CONTRADICTION 18 28) )
```

```
;Fig. 9
;* * *  Pure LISP kernel (slightly modified)  * * *


(DE #APPLY (FN ARGS A)
 (COND
  ((ATOM FN)
   (COND ((EQ FN (QUOTE CAR)) (CAAR ARGS))
         ((EQ FN (QUOTE CDR)) (CDAR ARGS))
         ((EQ FN (QUOTE CONS))
          (CONS (CAR ARGS) (CADR ARGS)) )
         ((EQ FN (QUOTE ATOM)) (ATOM (CAR ARGS)))
         ((EQ FN (QUOTE EQ)) (EQ (CAR ARGS) (CADR ARGS)))
         (T (#APPLY (#EVAL FN A) ARGS A)) ))
  ((EQ (CAR FN) (QUOTE LAMBDA))
   (#EVAL (CADDR FN) (#PAIRLIS (CADR FN) ARGS A)) )
  ((EQ (CAR FN) (QUOTE LABEL))
   (#APPLY (CADDR FN)
    ARGS   (CONS (CONS (CADR FN) (CADDR FN)) A) ))))

(DE #EVAL (E A)
 (COND ((NULL E) NIL)
       ((EQ E T) T)
       ((ATOM E) (CDR (#ASSOC E A)))
       ((EQ (CAR E) (QUOTE QUOTE)) (CADR E))
       ((EQ (CAR E) (QUOTE COND)) (#EVCON (CDR E) A))
       (T (#APPLY (CAR E) (#EVLIS (CDR E) A) A)) ))

(DE #EVCON (C A)
 (COND ((#EVAL (CAAR C) A) (#EVAL (CADAR C) A))
       (T (#EVCON (CDR C) A)) ))

(DE #EVLIS (M A)
 (COND ((NULL M) NIL)
       (T (CONS (#EVAL (CAR M) A) (#EVLIS (CDR M) A))) ))

(DE #PAIRLIS (V E A)
 (COND ((NULL V) A)
       (T (CONS
           (CONS (CAR V) (CAR E))
           (#PAIRLIS (CDR V) (CDR E) A) ))))

(DE #ASSOC (X A)
 (COND ((NULL A) NIL)
       ((EQ X (CAAR A)) (CAR A))
       (T (#ASSOC X (CDR A))) ))
```

```
;Fig. 10
;* * *  Pure LISP version of TARAI  * * *


(LABEL TARAI (LAMBDA
              (X Y Z)
              (COND
               (((LABEL GRTP
                  (LAMBDA (X Y)
                          (COND ((EQ X NIL) (QUOTE NIL))
                                ((EQ Y NIL) (QUOTE T))
                                (T (GRTP (CDR X) (CDR Y)))
                          )))
                 X Y )
                (TARAI (TARAI (CDR X) Y Z)
                       (TARAI (CDR Y) Z X)
                       (TARAI (CDR Z) X Y) ))
               (T Y) )))
```

付録5. LISPプログラム実行時の、各関数およびルーチンの実行回数に関するデータ

以下に、各関数、ルーチン（＊印）が呼ばれた回数の、測定データを示す。なお、より詳細については、システム工学科第4講座内部資料「LISPマシン測定データ, March 1979」を参照されたい。

表A5-1. TARAI-3 実行時の各関数, ルーチンの呼ばれた回数と, 全実行ステップ数に占める割合

function / routine(*)		count	step(%)
interpreter functions / routines (85.2%)	*EVAL	1,347	} 13.9
	*EVAL1	2,523	
	*ARGEVAL	5,047	28.0
	*APPLY	673	26.5
	*EXPR	673	7.4
	COND	673	9.2
system functions (14.8%)	GREATERP	673	9.3
	SUB1	504	5.5
defined fn.	TARAI	673	---

表A5-2. TPU-6 実行時の各関数ルーチンの呼ばれた回数と、全実行ステップ数に占める割合

function / routine(*)		count	step(%)
interpreter functions / routines (43.7%)	*EVAL	39,262	9.3
	*EVAL1	331,900	
	*ARGEVAL	390,002	13.0
	*APPLY	15,706	4.2
	*EXPR	15,706	1.2
	PROG	6,311	9.7
	GO	17,049	1.4
	RETURN	6,311	0.5
	COND	42,360	4.4
system functions (56.3%)	SUBST	4,361	17.3
	EQUAL	12,974	11.3
	SETQ	55,320	6.3
	MEMBER	6,392	4.8
	CAR	43,865	3.2
	NULL	25,743	2.1
	ATOM	19,319	1.6
	CDR	22,415	1.6
	OR	12,892	1.5
	CADR	9,343	1.2
	APPEND	1,210	0.8
	LENGTH	4,224	0.7
	CADDR	3,744	0.7
	LIST	2,936	0.5
	EQ	3,569	0.4
	CONS	3,379	0.4
	others	12,310	1.9
externally defined functions	RENAME	2,291	---
	INSIDE	1,836	---
	DISAGREE	7,559	---
	UNIFICATON	2,018	---
	DELETEV	184	---
	URESOLVE	499	---
	GUNIT	4	---
	PNSORT	4	---
	FDEPTH	0	---
	FTEST	0	---
	SUBSUME	1,293	---
	STEST	8	---
	CONTRADICT	8	---
	DTREE	1	---
	TPU	1	---