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

Microprogrammed Interpreter of the Experimental LISP Machine

瀧 和男　金田 悠紀夫　前川 禎男
Kazuo TAKI　Yukio KANEDA　Sadao MAEKAWA
神戸大学工学部
Kobe University

1.まえがき

LISP言語で記述されたプログラムの高速処理を目的とした専用コンピュータであるLISPマシンの研究は、単にLISPプログラムの高速処理の達成だけでなく、斬新なアーキテクチャを持つ新世代のコンピュータに対する先導的な役割を担うものと強く期待されている。

われわれのLISPマシンは、インタープリタのマイクロプログラム化と市販のLSIを使用することを前提として、まずインタープリタの基本設計を先行させ、その中で必要とされる機能を洗い出し、高速化に役立つ部分をハードウェア化する設計方針をとった。特にビットスライスALU、マイクロプログラムシーケンサLSIの使用は、ハードウェア量を低減させ、またハードウェアスタック、フィールド／ビット処理機能とともに高速化に役立っている。[1]

本報告はLISPマシンの概要について述べるとともに、インタープリタの制御構造を述べ、その高速化に各ハードウェア機能がはたしている役割をソフトウェアの立場から概説する。

2.ハードウェアの概要

2.1 LISPマシンシステムの構成

LISPマシンシステムのハードウェア構成図を図1に示す。LISPプログラム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LSI－ミニコン（dec社 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。システム構成は、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。

2.2 LISPプロセッサモジュールの構成

インタープリタからの要求により、図2のハードウェア構成をとった。ビットスライスALUはAdvanced Micro Devices社の4ビットスライスであるAm2903を4個使用している。制御部を構成するCCUは、Am2910マイクロプログラムシーケンサ、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。そのほか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、豊富な条件テスト回路、メモリの番地を3ビットのコードに変換するマッピングメモリ回路、等がある。

WCS　：　マイクロプログラム用のWCSは、1語 56ビットを4096語実装（アクセスタイム 150nsecのMOSメモリ）し、インタープリタ全体のマイクロプログラム化をはかる。またマイクロ命令周期は 300nsecである。

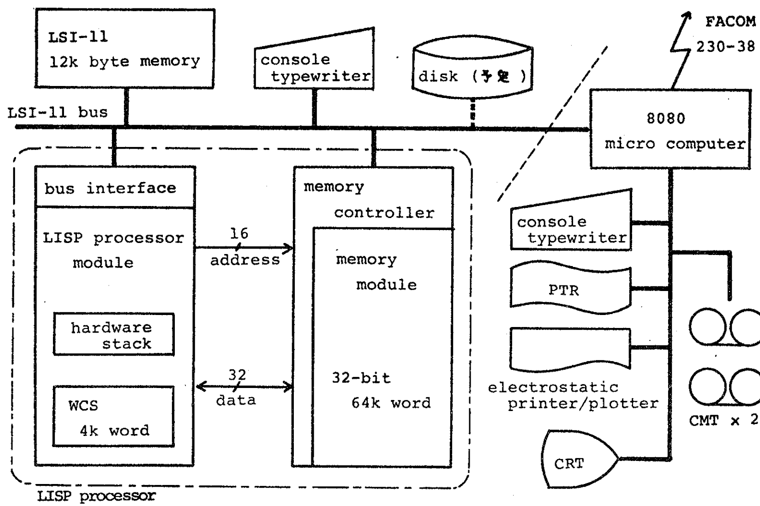


図1．LISPマシンシステムのハードウェア構成

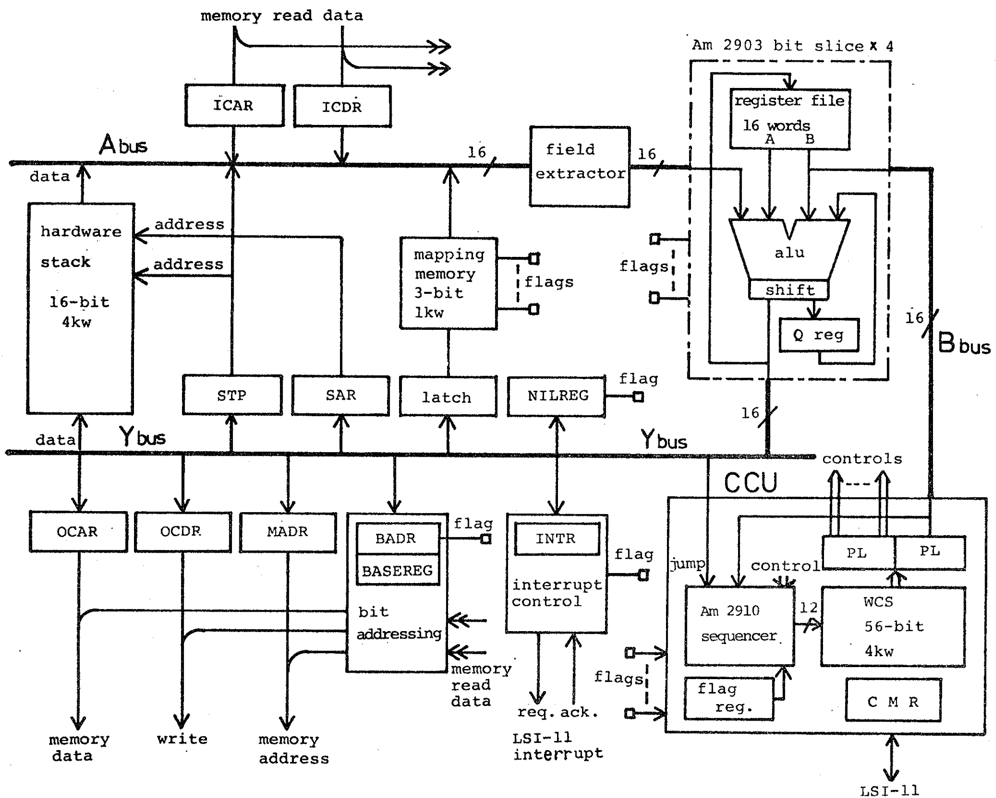


図2．プロセッサモジュールのハードウェア構成

CMR ： LSI-11から読み書きできる4ビットのレジスタである。run/halt、initializeの各ビットにより、LISPプロセッサの実行、停止、初期化、等の制御を行なう。

バス構成 ： Aバス（ソース）、Yバス（デスティネーション）、Bバス（CCUからの定数）から構成され、ALUはレジスタファイル、Aバス、Bバス上のデータに対する演算機能を有している。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出しを可能にしている。

フラグ ： ALUでの演算結果や、ビットアドレッシング回路等からの信号は、フラグレジスタを介してマ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。

ハードウェアスタック ： 70nsec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つのポインタレジスタSTP、SARを用いて行なう。いずれもカウンタタイプのレジスタで、間接後自動減少、自動増加後間接のモードを有する。これによりマイクロステップが減少できる。またSTP、SARを利用して、スタックからスタックへのデータ転送が1マイクロサイクルで完了する。

フィールド抽出回路 ： AバスとALUの間にはいり、データのマスキングとシフトを同時に行なう。

マッピングメモリ ： 3ビット×1k語構成で、主記憶アドレスを入力として、3ビットのコードを出力する。すなわち、主記憶を64語毎に1024区画に分割し、各区画に利用種別を表わす3ビットのタグを付けたと考えることができる。図3がLISPマシンのメモリ割当であるが、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。また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、リスト、文字アトム、数値、アトムの判定が簡単に行なえる。

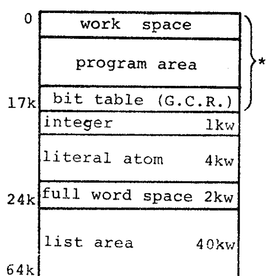


図3. メモリ割当

その他 ： メモリとのデータ受け渡しは、ICAR、ICDR、OCAR、OCDRを通して行なう。MADR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。

また、LSI-11に割込みを発生させるためのレジスタとして、INTRがある。INTRにデータを書き込むと、その値がベクタアドレスとなってLSI-11に割込みを発生し、受け付けられると、フラグレジスタにiackフラグが立つ。

また、ビットアドレッシング回路は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なうもので、ガーベージコレクションにおけるマーキング操作に用いる。

3. システムの初期化と入出力処理

LISPマシンシステムの初期設定とプログラムの実行は、図4、図5に示す手順で行なわれている。

3.1 LISPプロセッサの初期設定

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アトム類の作成。
iii) 各レジスタの初期設定。
iv) 自由リスト作成と自由リストポインタの値のセット。

を上げることができる。

i)、ii) はLSI-11が、CMTにデータの形で格納されているものを、WCSと主記憶に書き込むことによって行なわれる。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なわれるもので、LSI-11からのsignal（CMR={init}）信号により、LISPプロセッサが動作を開始して実行される。

3.2 式の読込みと結果のプリント

プログラムは、式formの読込み、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ないながら実行している。

式の読込みは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ない、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。

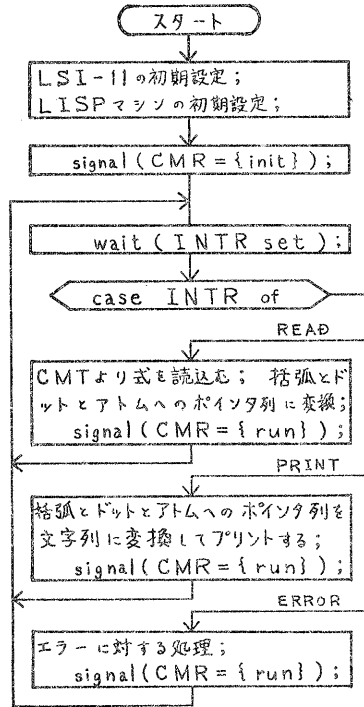


図4．LSI-11側の手順

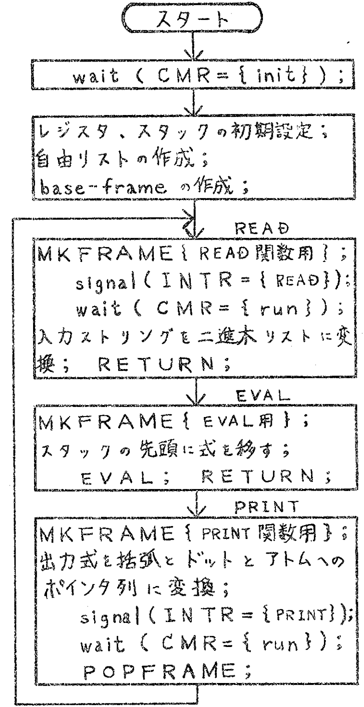


図5．LISPプロセッサ側の手順

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なう。

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、次の式の読込要求を出す。

となる。

4. LISP言語の仕様

4.1 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6に示す。

文字アトムをLISP 1.5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、属性値の取り出しも容易にした。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3の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なっている。

データ型	語数	表現
小整数	0	ポインタの値そのもの, -8192 ≦ n ≦ 8191
整数	1	31 30 … 0 sign (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 未使用 | 未使用
リスト要素	1	31 … 16 15 … 0 cdr | car

図6．データ表現

4．2 関数型の種類

関数型は、マシンコードで記述されたSUBR、FSUBR、ユーザ定義のEXPR、FEXPRの他に、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、MFSUBR、MRFSUBRを加えた。前二者が再帰呼出しを許しているのに対し、後二者は再帰を許さない。主な関数を次に示す。

MRSUBR : EVAL, APPLY, MAP, etc.
MRFSUBR : COND, PROG, LIST, etc.
MSUBR : CAR, CDR, CONS, EQ, etc.
MFSUBR : DE, DF, QUOTE, SETQ, etc.

5. LISPマシンのインタープリタ

5．1 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a-list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（後述）。

ii) APPLYルーチンの改良（後述）

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ない、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、強大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。

5.2 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7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(PC と 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。

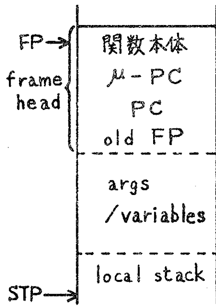


図7. frameの構造

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。

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。

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8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、呼び出し元の frame head には、マイクロプログラムの戻り番地と PC レジスタの内容が、保存されている(μ-PC と PC 位置)。

関数本体での処理の流れの例を図9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 を用いて計算が行なわれ、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、関数値として呼出し元へ返す値を呼出し元の stacktop (現FP位置)に置き、現frame を消滅させ、呼出し元の 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

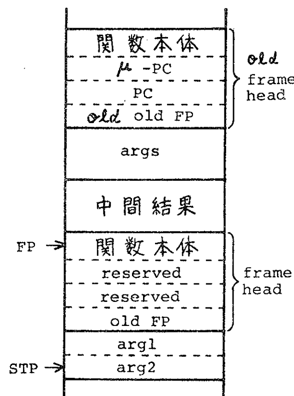


図8. MRSUBR関数へジャンプするときのスタックの構造

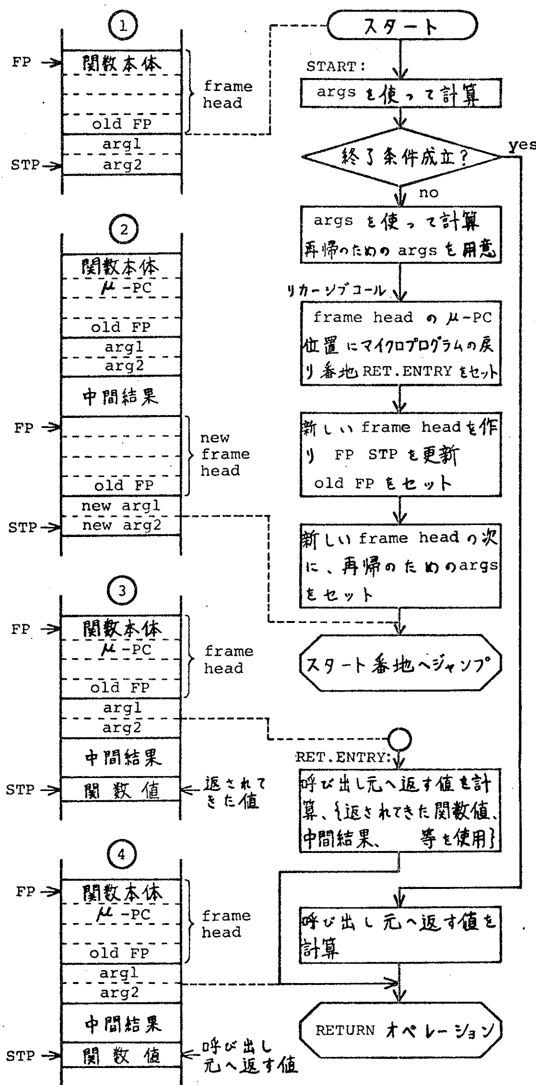


図9．MRSUBR関数本体における処理の流れの例

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の復元も行なう。

MRSUBR関数の本体とは、1つ前のframe headの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。

5．3 関数EVALと式の評価

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10に示す。また、図11はLISP 1.5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10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なう。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10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12）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なわれる。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11のspread[evlis[---]]に相当する関数、ARGEVAL（図13）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。

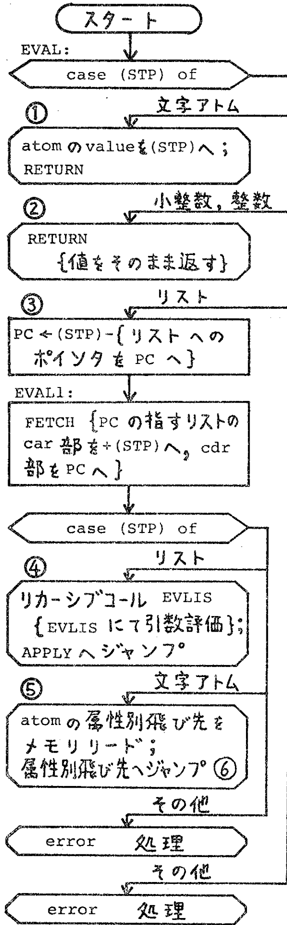


図10．EVAL ルーチン

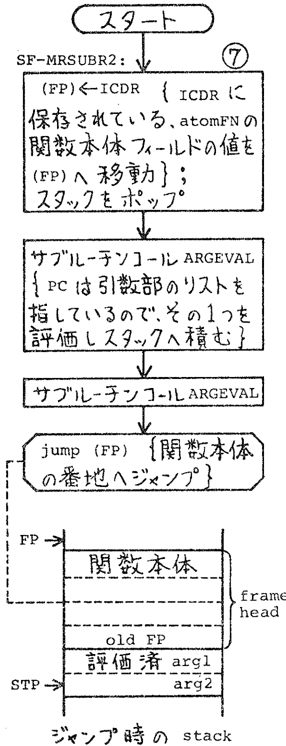


図12．SF-MRSUBR2 ルーチン

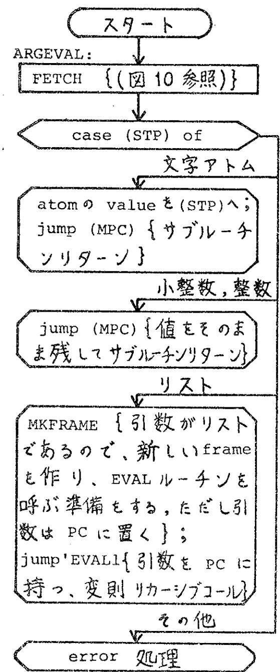


図13．ARGEVAL ルーチン

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、引数として再び（関数名　引数　---）の形が現れた場合には、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なう。この場合高速化のために、EVAL のスタート番地ではなくて、EVAL 1 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。

```
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]
         ⑦{spread[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11．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14．LISP 1.5 における APPLY の定義

5．4 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れるルーチンは、図14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［1］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。

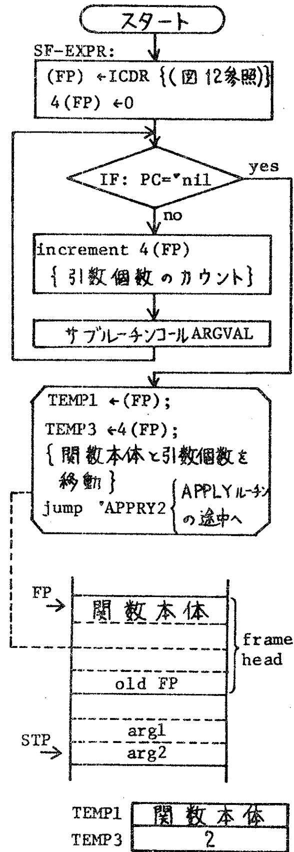
（DE USRFN (X Y)(COND---))

その結果、アトム USRFN の属性別飛び先フィールドには、((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15）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。

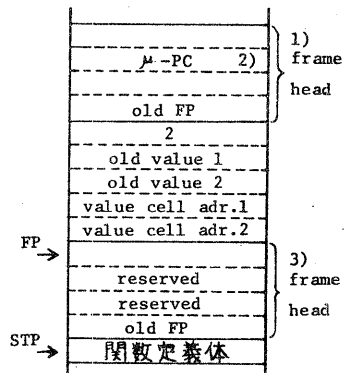
APPLY2では、shallow-bindingを行なったのち、アトムのold-valueとその番地の対をスタックに残し、関数定義体（USRFNの本体のうち（COND---）の部分）を引数として、EVALをリカーシブコールする。戻り番地としては、old-valueを書き戻すための、REBINDルーチンの入口がセットされる。このときのスタックの状態を図16に示す。

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LISP 1.5と同じである。



ジャンプ時のスタックとレジスタ

図15． SF-EXPRルーチン



1) REBINDのframe（もとAPPLYのframe）

2) REBINDルーチンの入口のアドレス

3) EVALのframe

図16． old valueを保持するスタック

6. あとがき

以上、インタープリタの高速化の要点と、その中で各種ハードウェア機能がどのように利用されるかを中心に述べてきた。本LISPマシンシステムの設計思想の中心は、なんといっても、ビットスライスLSIの使用という制限のもとで、インタープリタの制御構造を極力ハードウェアに反映させようとするところにあるが、ハードウェア化した機能のそれぞれの成否については、システムの稼動やシミュレータの完成を待たねばならない。

ともあれ、新しいアーキテクチャを持つ計算機が、研究室レベルで試作可能な時代を迎えたのは、喜ばしい限りである。

インタープリタの設計の中で生じてきた、リストの要素を順にたどってゆくPCの考え方、すなわちリストのcar部を機械語のように解釈、実行してゆく制御方式は、新しいアーキテクチャの先導としても興味あるもので、今後の研究課題の1つでもある。

謝辞

研究に協力してくださった、システム工学科第4講座の、小林康博氏、滝本博道氏、多田光弘氏に感謝します。

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] 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)。

[3] 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).

[4] 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).

[5] 中西正和：LISP入門，近代科学社，(1977).

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

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

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

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

[10] D.G.Bobrow: A Model and Stack Implementation of Multiple Environments. C.ACM Vol.16 No10, (1973).

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