マイクロプロセッサを用いた Lisp マシン ALPS/I†

井田昌之††　間野浩太郎††

本論文では，マイクロプロセッサ（インテル 8080）とバルクメモリ（64 k 語，1 語＝36 ビット)を中心に作成された Lisp 専用システム ALPS/I (Aoyama List Processing System/I) について述べている．Lisp インタプリタは 11 k バイトの PROM に書き込まれており，RAM 8 k バイトをその作業域に持っている．

プロセッサ側の RAM と Lisp データを保持するバルクメモリとの間には，簡潔かつ高速なデータ転送を可能とする特殊インタフェースが用意されている．このインタフェースには AAM と呼ばれる RAM アドレス変換機構があり，それらは付加された 1 バイト型の擬似 I/O 命令により起動される．これにより 16 ビット空間が効果的に拡張されており，その原理が述べられる．

Lisp 処理系の構成にあたっては，ハッシングの利用による素情報の処理の統一的な簡潔化・高速化，連想子の組込み，いくつかの処理アルゴリズムの改良などを行っている．

また，組込関数が約 100 用意されており，それらと 56 k 語ある自由領域を用いてかなり大規模なプログラムでも実行させることができる．

1. はじめに

記号処理言語 Lisp は二進木を中心とする独特な言語[1]で，広く人工知能関連分野に用いられている．その処理系は，文献 2) などに Lisp 自身により定義されており，それらにより比較的容易に基本処理系（インタプリタ）を作成できる事が知られている．

しかし，実際的な処理系にするには一般的な機能整備に加え，二進木リスト等の保持に充分なメモリ（例えば 64 k セル）を備える必要がある．このため，一部の例[20]を除いて実用的な Lisp システムの多くは大型機上で作成されている[3)~6)]．更に，国内における Lisp 処理系の絶対数が少なく，手軽に Lisp プログラムを作成実行できない現状がある．そこで我々は，手近な処理系を目指して，価格／性能比志向の Lisp 専用機を開発する計画を立てた．

Lisp の特性及び使用上の便宜から専用機の満たすべき条件を次のように考える．1)電源投入のみで作動し，安価である．2)数十 k セル程度以上の Lisp データの保持を許す大容量メモリの具備．3)頻繁な Lisp データ参照の高速化機構．4)可用性向上のためのシステム整備と各種関数の組込．

この条件を満たすよう我々は Lisp マシン ALPS/I (Aoyama List Processing System/I) を作成した．ALPS/I では上記条件に対応して，1)最近急激に低価格になっており，また拡張性・市場性がある 8 ビットマイクロプロセッサを CPU に採用し，処理系は半固定記憶 (PROM) に記憶させる．2)その安価傾向から大容量の IC メモリを採用する．参照の効率から語構成（1 語 32 bit 以上，特に 8 ないし 9 の倍数 bit 長のものは入手が容易）を取り，物理的なアドレス空間は CPU 能力とのかね合いから 16 bit で構成する．3)バルクメモリに対する専用インタフェースを作成し低速なマイクロプロセッサの通常の入出力命令によらず，転送時のソフトウェアステップ数を最小におさえ高速化を計る．4)約 100 の関数を組込み会話型処理形態を提供する．2 章ではこの ALPS/I のハードウェア構成を，3 章ではその上に構築された Lisp 処理系について述べる．

† Lisp Machine Based on a Micro-processor: ALPS/I by MASAYUKI IDA and KOUTARO MANO (Faculty of Science and Engineering, Aoyama Gakuin University).

†† 青山学院大学理工学部経営工学科

2. ハードウェア構成

2.1 8 bit マイクロプロセッサとバルクメモリ

ALPS/I システムは 8 bit マイクロプロセッサ・インテル 8080[8]，PROM 11 kbyte，RAM 8 kbyte，Inkjet Printer (PTR/PTP 付)，カセットテープ装置，バルクメモリ 64 kW (1 W＝36 bit) より構成される（**図 1，図 2**）．8080 及びバルク IC メモリの採用に特徴があり，この点を中心に記述を行う．

まず，拡張性・市場性を考え CPU に 8 bit マイクロプロセッサ (8080) の採用が設計当初に決定された．その理由は次の通りである．

1）設計当時（50 年 3 月）入手が容易であり，また段階的なシステム拡充が可能と思われた．（現在では，

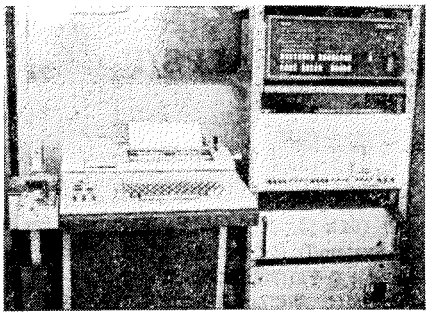

図 1 ALPS/I システム
Fig. 1 ALPS/I system.

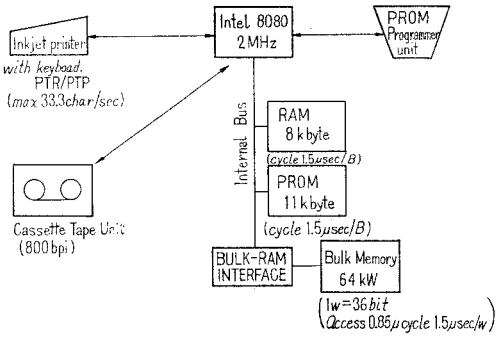

図 2 ハードウェア構成
Fig. 2 Hardware configuration.

CPU, RAM, PROM, 入出力などのチップ, モジュールの高速・高能力化が進み, これらに置きかえれば少なくとも倍以上の速度が見込まれる.)

2) ハードウェアスタックポインタがあり, 再帰呼出し等に便利でかつ比較的速い. (再帰呼出しの可能な call 及び ret 命令は現在 12.5μsec 及び 7.0μsec で, 0.5μsec のメモリを用いれば 8.5μsec 及び 5μsec で実行できる.)

3) 1バイト語長の命令が多くステップ数に比してコンパクトに作成できる. (7800 ステップで 11 kbyte であり, 1命令あたり約 1.4 バイトを占める. またインタプリタのみでは 2 kbyte 弱の大きさしか要しない.)

4) 算術演算に比して単純なデータ転送の多い Lisp 処理系の場合, 2) 及び 3) により他の言語の場合と比べてマイクロプロセッサの低速性による速度低下はそれ程ふえないと思われる.

Lisp 処理系は, 明示的にリストを扱うため, メモリアドレス空間の設定方法は他の言語以上に処理能力及び速度に直接影響する. 例えば 16 ビットミニコンでの Lisp 処理系では, 64 k 語以上の記憶能力を与えるため二次記憶へのリスト退避手続を用意し, その管理を使用者に任せる事はよく行われてきた. また最近では, ソフトウェアによるページング機構を介して仮想ストレージを外部記憶上に持ち, 仮想アドレスをポインタ情報とする例も報告されている[20]. 前者の二次記憶の利用法は簡便である半面, Lisp 使用者に負担がかかる. 後者の仮想化は対象となるリンクリストの非局所性の問題, ソフトウェアによる場合はそのアドレス変換の低速性の是否が吟味されねばならない.

記憶能力の拡大には上記の他に, メモリバンクの切換えがミニコン等でよく見られる. バンクの切換えは仮想化に比べて一般に高速性があるので, メモリ参照の多い Lisp 処理系においては好ましい. ALPS/I ではそのため, システム用の内部メモリ (PROM と RAM) と Lisp データ格納用メモリ (バルクメモリ) を分離し, 各々独立したアドレス空間を持たせて能力を上げている.

内部メモリはバイトアクセスされるが, バルクメモリは①バイトアクセスしバンクを増す ②Lisp の基本単位(セル)を1ロケーションとした1バンクを構成し, 語アクセスする: の2方法が考えられる. ①は一般的ではあるが, Lisp ではセルアクセスできればよいので②より低速となる. ②の方法での問題点は, 内部メモリとアクセス幅が異なる (バイト対語) 事である. この点を解決する事ができれば Lisp セルと処理系を切離す事ができ, かつ1ロケーション1セルとなるのでアドレス情報の処理が容易となり, 8080 を用いても比較的高速にデータを参照できる. このために考え出された機構を次節に示す.

2.2 バルクメモリ・RAM インタフェース

バルク上の1セルは 32 bit+3 bit のフラグより成る. 8080 の命令ではこうした語データを処理する事はできないため, 内部メモリの RAM へ転送し処理を行う事とする. 通常の入出力命令を介していたのでは (DMA チップの起動であっても) 遅いので, 転送用の命令を擬似的に追加する. 転送の起動には5クロックしか要しない. 命令実行時のアドレスバス 16 bit, データバス 8 bit のみにより転送指示情報はすべて合成され, 他のメモリ参照等は行われない.

転送において, バルクアドレスはすべて任意に指定できねばならないが, RAM アドレスは専用機である特質を活かせば任意性は不要で複数指定できれば充分

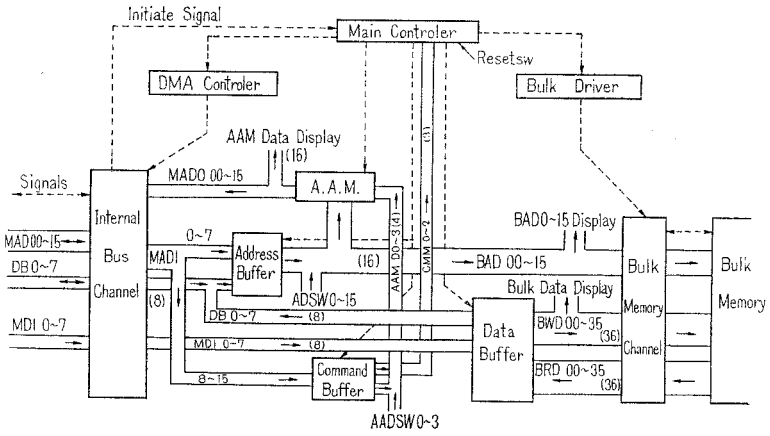

図 3 バルク-RAM インタフェース
Fig. 3 Bulk-RAM interface.

である．Lisp 処理系の場合次の指摘ができる．①サブルーチン (subr) 呼出しの引数はたかだか 4 個である．②リンクのたどり，スタックの読出しなどにはたかだか 2 個のバッファがあればよい．③再帰手続は比較的少ない．以上の事から作業域も含め RAM 側には 16 個のバッファがあれば充分である．そして引数の受渡しはバッファ上でそのまま行え，RAM 上での二次的転送は不要である*．16 通りの RAM 側バッファの先頭アドレスは，このため動的に変える必要はなく，実際の転送時には識別のために 4 bit (0～15) あればよい．作成された回路は次の 4 点にまとめられる (**図 3**)．

1) RAM アドレス変換機構として AAM (Address Association Memory) を持つ．転送時には RAM 論理アドレスとして AAM レジスタ番号を指示し，実番地を生成させる．AAM は 16 bit 16 語の RAM (アクセス 30 nsec) である．

2) 転送命令として transfer (TR) 命令を擬似的に 1 バイト命令に加え，最小の CPU サイクルの消費で転送が行われる．必要な本来の TR 命令は 3 オペランド命令で次の形式を持つ．

TR *c*, *addr*1, *addr*2

c は転送制御を表わし，現在 3 種類存在する．*c*=1 : (Bulk [*addr*1])→(Ram [AAM [*addr*2]], …,

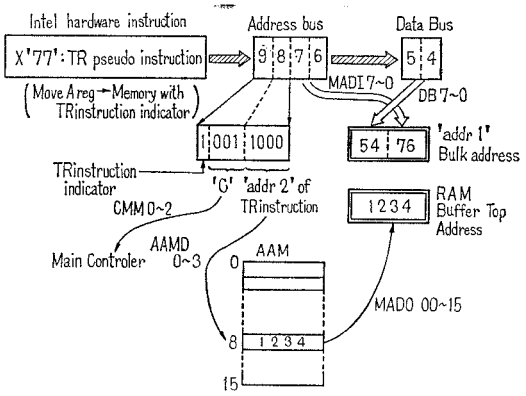

図 4 TR 擬似命令からのバルクアドレス及び RAM アドレスの形成例
Fig. 4 An example of the formation of bulk and RAM addresses from TR pseudo instruction.

Ram [AAM [*addr*2]+4]) ; *c*=2 : *c*=1 の逆方向転送 (Ram→バルク) ; *c*=4 : AAM [*addr*2]←*addr* 1 ; *addr*1 及び *addr*2 は各々，0≦*addr*1≦65535，0≦*addr*2≦15 である．

この TR 命令のために，1 バイト命令のうち，未実装領域 (32768～) へのストア命令 (「MOV M, n」及び STAX 命令) を代用し，*c* に 3 bit, *addr*1 に 16 bit, *addr*2 に 4 bit をあてる (**図 4**)．この結果入出力接続では 100μsec 以上を要するバルク 1 語 RAM 5 byte 間の転送を約 20.5μ で行う**．回路は常時プロセッサから出されるメモリストア命令を監視する．memory write cycle でかつアドレスバスの最

* 他関数の呼び出しを含む subr の場合にのみ退避を push, pop 命令により行う．約 70 の subr のうち，13 個の関数において行われるだけである．

** 実インタフェース時間は約 11 μsec. RAM 所要時間=1.5μsec/byte×5，バルク所要時間=1.5μsec+reflesh time (約 60μsec おきに 1.7μsec).

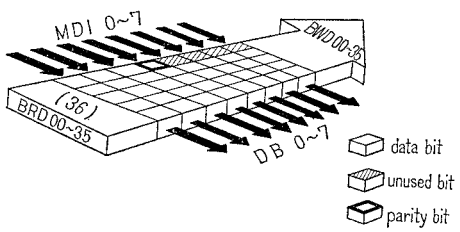

図 5 変則データバッファの構造
Fig. 5 Structure of augmented data buffer.

上位 bit が1である時，データバス 8 bit・アドレスバス 16 bit をラッチし，CPU をホールドさせ転送を行う．

3）変則データバッファ(**図5**)を持つ．byte 単位と 36 bit 単位の2種類のアクセスを許しており，RAM・バルク間の転送時にはパリティ生成・チェック及び第5 byte の下 4 bit の無視・0の挿入が行われる．

4）独立した保守パネルがあり，リセット機能の他，バルクメモリ・AAM の内容の表示，データ転送エラー，コマンドエラーの表示と動作の停止を行う．

3. Lisp プロセッサ構成

3.1 概　要

Lisp プロセッサは IBM 370/135 上に作成されたクロスソフトウェア[11](アセンブラ及びシミュレータ)上でハードウェア作成・改良と並行して開発された*.

この Lisp プロセッサは Lisp 1.5[2] を基準として次の特徴を持っている．

1) hashing によるアトムの格納．P-list はない．

2) hashing 機構を生かしたハッシュ化配列[12]（連想三つ組の保持可能），連想計算機能[5]を持つ．システム識別子 Harray, Hexpr を各々導入し，これらのインタプリタへの簡潔な組込み．

3) stack を用いた deep binding による変数束縛[13]．

4) evlis の結果の即時回収（3.3 節参照）．

5) Lisp 1.6 型 PROG＋ラベルの高速サーチ．

6) 機械語ルーチンの一時的なロード機能(PROM プログラマ，各種保守ルーチン，ユーザ組込 subr 等)．

3.2 メモリ構成

メモリ構成を**図6**に示す．バルクメモリ及び RAM の初期化は電源投入時に動作するルーチンにより行われる（この間約 2.33 秒）．

主記憶空間は次の順に割当てられる．

1) システム常駐域 I (0～8191 番地)

2) スタック領域 (8192～14335) スタックポインタにより直接使用される領域．3072 段可能である．

3) バイナリプログラムスペース (144336～15875) 擬関数 load により機械語プログラムをロードできる．*fn 2 の名によりそのプログラムを Lisp 内から呼びだす事ができる．

4) システム作業域 (15876～16383)

5) システム常駐域 II (16384～20479)

また，バルクメモリは次の順に割り当てられる．

1) 変数束縛用スタック：1024 段のスタックで car 部に引数，cdr 部に値を持つ．

2) フルワードスペース：基本整数以外の数が格納される．数の一意性は保証されている．

3) prog 中間言語ロード域 (3.5 節参照)

4) GC 保護用スタック：1024 段のスタックで中間結果の退避に用いる．

5) アトムストリング域：アトムの pname 及びストリングアトムの文字は各々4字を越える場合，その残りはこの領域にチェインされる．

6) H-領域：2語1組で使用される (3.4 節参照)．

7) フリーストレージ：56 k セルを格納できる．

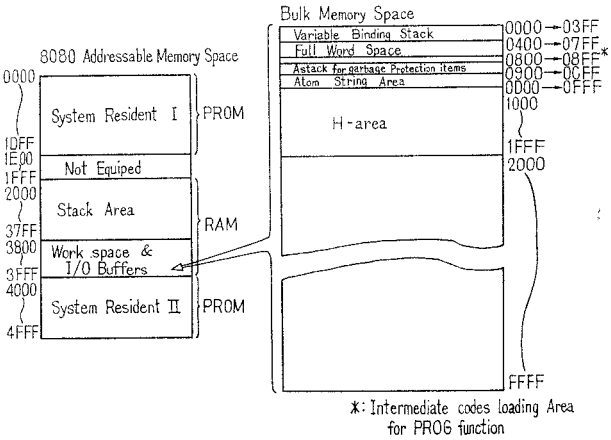

図 6 メモリ割付けと階層構造
Fig. 6 Memory allocation and hierarchy.

* シミュレータ ALPS/I は全構成の模擬，実行時間の計測，各種デバッグエイドが含められている．またアセンブラはマクロ機能を持っている．

このバルクアドレス空間は，基本整数 (0～1023) はアドレス自身を値とするなど[12]，データの処理効率と識別性を高めるように配列されている*．例えば numberp 及び atom 述語関数は，

numberp[x]=[$addr[x]$**<2048→T; T→NIL]

atom[x]=[$addr[x]$<8192→T; T→NIL]

と定義され，1回の比較により値が決定される．

3.3 ガベジコレクション

ALPS/I におけるガベジコレクタは3段階に動作する．

1）即時回収コレクタ[18]：インタプリタ内で生成され，即座に不要になるものは，通常のガベジコレクタの動作を待たずに，各々の不要になった時点で回収される．例えば実引数リストの作成を行うインタプリタ内の evlis の値は目的関数の適用後は不要で，即時回収される．各種の例題の実行においては，消費セルのうち約 20% が即時回収されている．

2）ガベジコレクタ (GC)：GC はフリー領域が一杯になった時に動作する．回収に先立つマーキングは後述されるH分子のうち属性値1から6の値部，属性値 8, 9 の値部及び鍵部，束縛スタック，GC 保護用スタック，current cons 対象の順に行われ，フリー領域の線形走査をその後行い，マークされなかったセルの回収及びマークの除去がなされる．

3）グランドガベジコレクタ (GGC)：GGC はH領域のオーバーフロー時に動作する．GC に加えて不要になったH分子及びフルワードセルを回収する．

3.4 H 領域の構成と管理

H領域は 2048 エントリーを持つ 4k 語の領域で，H分子を格納する．H分子は属性部・鍵部・バリアント部・値部及びステータス部を持ち，システム内での一意性が保証されるものをいう（図 7）．

このH分子 h に対しては システム用に次の7種の基本関数が定義されている．

i）属性値の取り出し：*get-attrib* [h]=a.

ii）値の取り出し：*value* [h]=v.

iii）バリアント部の取り出し：*variant* [h]=va.

va は atomic symbol に対しては key となる pname の文字数，associator 等には鍵部の情報のチェックサム．

iv）鍵部の取り出し：*key* [h]=k. k は atomic symbol に対しては pname string，他の場合は鍵となる（key 1*. key 2*）のドット対．

v）H分子の作成：*hstore* [h; v; va; k; *attribute*]=h'. h' は (v, va, k, *attribute*) により決定されるH分子の格納される番地．h 番地が既に他のH分子により占められている場合には，空きがあるまで rehash され h' が決められる．この時，*and* 関数の定義[2]の通りにバリアント部が等しい場合にのみ第2ワードがアクセスされ key が比較されるので，バルク参照ワード数が削減されている（図 8）．

vi）H分子の削除：*delete* [h]=

set [*status*[h]; *not-active-but-used*][17]

更に (a o v) 型の連想子に対しては，

vii）key として a 及び o をもつH分子の値 v の取り出し．*hassoc* [a; o]=v.

H分子の格納は hashing[16] により行われる．col-

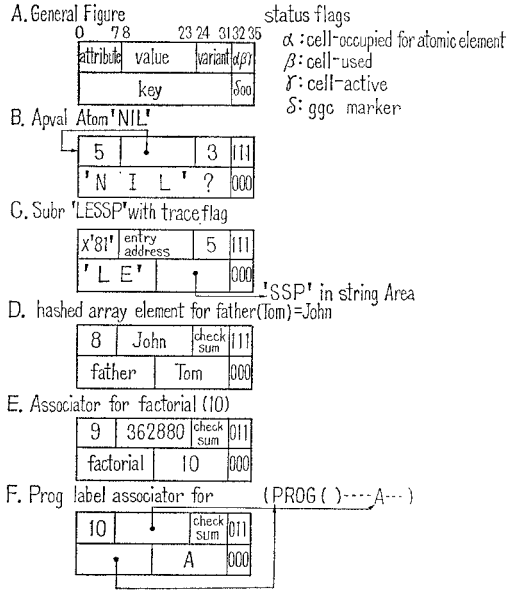

図 7 H分子のデータ構造とその例

Fig. 7 Data structue of H-molecule and its examples.

```
hstore [h; v; va; k; attribute]=prog [[slot];
        slot=h;
  LOOP [cell-not-active [slot]→return [make-hmolecule []];
        and [eq [attribute; get-attrib [slot]];
             eq [variant [slot]; va]; eq [key[slot]; k]]→
                return [store-value [slot]]];
        slot =rehash [slot];
       [eq [slot; h]→ggc[]]; go [LOOP]]
```

図 8 hstore 関数の M 式表現

Fig. 8 M-expression of hstore function.

* 文献 13) にも示唆されているように，メモリ配置の一層の高効率化は将来の課題である．

** *addr*[x] は x の格納された実番地を値とする関数とする．

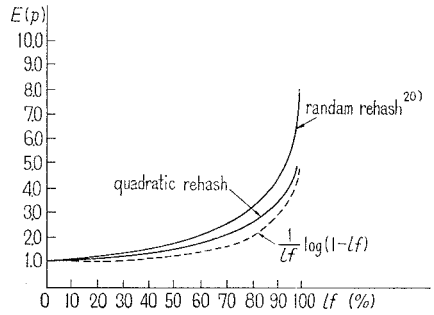

図 9 非ランダムデータ[17]に対する平均探査回数
Fig. 9 Average number of probes.

lision は rehash により解決する*. rehashing 手法は $R=1$ とした quadratic search[15),16)] を用い，次の手順により行う．

ⅰ) set k=hash [key], $j=R$

ⅱ) もし k 番目の要素が空であるか，等しい key を持っていれば終了．

ⅲ) さもなければ $k=k+j$, $j=j+1$ としてステップ2へ戻る．

この時の平均探索回数 $E(P)$ は，

$E(P)=-\log(1-\rho)/\rho$, ρ: load factor で与えられる．また，サーチの最大区間は $N-R+1$ である[16)]．この rehash アルゴリズムは以前に用いられていた random rehash[17)] と比較して高速であり，また load factor がふえても実質探索回数がそれほどふえないように改良されている（**図 9**）．

このように扱われるH分子の生成及び消去手順は**表1**にまとめる事ができる．

3.5 Lisp インタプリタの構成

ALPS/I に組込まれた万能関数のうち，*apply*, *eval*, *evlis* の M 式による定義を **図 10** に示す．

Lisp インタプリタは1命令あたり1.4バイトと，1バイト命令を比較的多用して構成されている．各々の subr, fsubr 関数の記述は次の規則を基本とする．

ⅰ) 引数はバルクバッファを直接用いる．

ⅱ) 引数の退避が必要となるモジュールにおいては引数の性質に応じて，ハードウェアスタックないしは GC 保護用スタックへのスタック命令を随時記述する．

ⅲ) 関数の値は D, E レジスタ[8)] へ返す．

作成されたインタプリタについては，関数引数の処理と prog の処理について次に説明する．

関数引数は Optional Freeze により処理する[18)]．すなわち，関数引数において凍結の必要のある自由変数は第三引数以降につなげ，関数引数の実行は，凍結された値の再束縛を行ったのちに行われる（図 10）．この機構は **function* により引用する時に働き，通常の *function* 関数による引用は *quote* と同じ処理がされる．

prog インタプリタは中間言語域を使用して展開を行い処理する[12)]．展開に際してはラベルの連想子の登録，prog 変数の *local* 宣言の追加が行われる．*go* 文は Lisp 1.6 の仕様に従い，飛び先を動的に指定できる．ラベル連想子により行先が保持されているので登録されるラベル数に依存せずに高速に分岐が行われる．アトム引数の *go* 文の場合は中間言語域上の対応する部分が1度 *go* 関数に与えられると無条件分岐を行う ***go* に書きかえられ，short-cut が生じる．これを**図 11** に示す．この中の *change-statement* [x ; y] は現在 sequence-counter が指した中間言語の function 部を x に，operand 部を y におきかえる擬関数とする．この修飾は展開されたコードに対して行われ，元のS式はそのままで他の処理にはまったく影響を及ぼさない．

表 1 H分子の分類
Table 1 H-molecule properties.

	atomic symbols								associators		
attribute name	—	subr	fsubr	expr	fexpr	apval	hexpr	harray	harray element	assoccomp associator	prog label associator
attribute value	0	1	2	3	4	5	6	7	8	9	10
created by	read etc.	—	—	define deflist	deflist	cset csetq	deflist assoccomp	array	seta	hexpr function refer	prog label
deleted by	ggc	—	—	—	—	—	—	dearray	delete dearray	ggc	ggc

* 他でも指摘されているように，チェイン法の方が同一エントリ数の比較では高速ではあるが，テーブルサイズを一定とした時には，チェイン法はエントリ数が半減し，実テーブルサイズに対するロードファクタも半減する．

```
                                    (Evalquote, Evcon, Suppressed)
apply [fn; args]=prog [[temp1; temp2; v];
  [atom [fn]→[eq [get-attrib [fn]; EXPR]→return [apply [value [fn]; args]];
                 eq [get-attrib [fn]; SUBR]→return [progn [spread [args]; call [value [fn]]]];
                 eq [get-attrib [fn]; HEXPR→[hdotp [fn; args]→return [
                                    hassoc [fn; args]]; T→progn [v=apply [value [fn]; args];
                                    hstore [fn; args; v]; return [v]]];
                 eq [get-attrib [fn]; APVAL]→return [value [fn]];
                 T→return [apply [sassoc1 [fn; ERRORA2]; args]]];
  temp1=car [fn]; temp2=cdr [fn];
  [eq [temp1; LAMBDA]→progn [stacksave []; stackpush-loop [car [temp2]; args];
                                    v=eval [cadr [temp2]]; stackrestore []; return [v]];
   eq [temp1; LABEL]→ progn [stacksave []; stackpush [car [temp2]; cadr [temp2]];
                                    v=apply [cadr [temp2]; args];
                                    stackrestore []; return [v]];
   eq [temp1; FUNARG]→progn [stacksave[]; stackpush-loop [cadr [temp2]; caddr [temp2]];
                                    v=apply [car [temp2]; args];
                                    stackrestore []; return [v]]];
    return [apply [eval [fn]; args]]]
  eval [form]=prog [[temp1; temp2];
   [numberp [form]→return [form];
    atom [form]→return [[eq [get-attrib [form]; APVAL]→value [form];
                                    T→sassoc1 [form; ERRORA8]]];
   temp1=car [form]; temp2=cdr [form];
  [not [atom [temp1]]→progn [v=apply [temp1; evlis [temp2]]; igc [*last]; return [v]];
   v=get-attrib [temp1];
  [ev [v; EXPR]→progn [v=apply [value [temp1]; evlis [temp2]]; igc [*last]; return [v]];
   eq [v; FEXPR]→return [apply [value [temp1]; cons [temp2; NIL]];
   eq [v; SUBR]→progn [spread [evlis [temp2]; v=call [value [temp1]]; igc [*last]; return[v]];
   eq [v; FSUBR]→return [call [value [temp1]]];
   eq [v; HEXPR]→return[[prog2 [w=evlis [temp2]; hdotp [temp1; w]]→
                              hassoc [temp1; w]; T→progn [v=apply [value [temp1]; w];
                              igc [*last]; hstore [temp1; w; v]]];
   eq [v; HARRAY]→return [hassoc [temp1; [eval [car [temp2]]]]];
   return [eval [cons [sassoc1 [temp1; ERRORA9]; temp 2]]]]
evlis [m]=[null [m]→setq [*last; NIL];
          null [cdr [m]]→setq [*last; cons [eval [car [m]]; NIL]];
          T→cons [eval [car [m]]; evlis [car [m]]]]
hdotp [x; y] is a presence predicate.
      if the associator having (x*. y*) key is found, then true else false.
hassoc [x; y] is a function to obtain the value of the associator having (x*. y*) key.
```

図 10 Lisp インタプリタの M 式表現

Fig. 10 M-expression of the Lisp interpreter.

```
go[x]=[atom[x]→[addr=hassoc [x; pform]→
            progn [change-statement [**go; addr];
                   seq-counter=addr];
          T→label-error[]];
       addr=hassoc [eval [x]; pform]→addr;
       T→labe-error []]
**go [x]=setq [seq-counter; x]
pform is an atom, whose value is a current prog-body
    top address, seq-counter is an atom to indicate a
    next prog statement to be eval-ed.
```

図 11 go 及び **go 関数

Fig. 11 go and **go function.

4. ま と め

4.1 総 括

廉価ではあるが低速な 8 ビットマイクロプロセッサと最近廉価になったダイナミック型の IC バルクメモリとを組み合わせて実用的な規模と速度の Lisp 専用機を作る方法とその基礎となる考え方を示した．

作成に際して設定した基準は 1)経済性 2)機能 3)速度 であったが，これらは満足したと考えられる．**表 2** は文献 19) において設定された標準問題の実行結果で，ミニコンのもの[20]を一例に比較したものである．ほぼ同等の実行時間であるが，prog インタプリタの高速化のために Sort の問題では速い値となっている．

機能的には 56 k の自由領域を用いて，今まで小型機の Lisp 処理系では扱えなかった既発表の Lisp プログラムの多くは実行できるようになっている．Lisp に基づく数式処理言語 REDUCE はそのプログラム自身で約 44 k 語を占めるが，現在小型の応用問題な

表 2 テストプログラム[19]の実行時間
Table 2 Execution time obtained from test programs.[19]

	ALPS/I		Reference data[20]	
	Execution Time* (msec)	Used cells	Execution Time (msec)	Average GC Times
WANGA	176	826	100	0
WANGB	970	1,285	600	0
BITA 6	8,750	4,738	5,532	0
BITA 7	29,800	15,836	22,432	0.3
BITA 8	99,500	54,514	75,925	0.8
BITB 6	1,980	1,283	1,495	0
BITB 7	5,950	3,476	4,345	0
BITB 8	19,400	11,145	15,825	0.1
Sort 60	77,500 (54,500)**	31,402	92,825	1.0
Sort 80	110,000 (83,000)**	47,227	164,200	1.7
Sort 100	137,000 (104,500)**	over 56 k	246,850	2.5

*: ALPS/I does not require garbage collection to process these test programs.
Time required for iterative execution of test programs was measured with a stopwatch.
**: When using system built-in subr EQUAL, APPEND not as expr.

ら処理できるようになっている．$(x+y)^5$ の展開に約 40 秒を要している*．

4.2 今後の展望

これまでの研究で痛感したのは次の 2 点である．

1）アドレス演算に関して　　Lisp で不可欠なポインタのつけかえは 16 ビット転送命令があるので問題はなかった．しかしアドレス情報の比較・加減算は約 100 カ所も存在するので，この高速化を可能とする命令の存在により処理系の速度を大幅にあげる事ができる．

2）スタックの強化　　2 組のハードウェアスタックがあれば約 60 カ所の記述を高速化できる．またスタックの高機能化も望まれる．

将来の更に大規模な応用に対しては，アドレス空間の拡張・新しい素子（例えば 16 ビットチップ）の採用・コンパイラ化なども検討する必要があろうが，これまでの考え方の延長で対処できよう．

謝辞　作成にあたって協力を仰いだ小林茂男・重光宏之・山方宏修・森芳喜君を始めとする青山学院大学理工学部経営工学科間野研究室の諸兄，貴重な御助言を頂いた東京大学理学部後藤英一教授に謹しんで感謝いたします．

* RAM 及び PROM は現在 1.0μ 秒の速度である．またバルクメモリ参照時間はほぼ全処理時間の 18% 程度を占めている．現在発表されている素子を使えば処理系をかえずに約 2 倍の速度における事は容易である．

参 考 文 献

1) McCarthy, J.: Recursive Functions of Symbolic Expressions and Their Computation by Machine, Commun. ACM, Vol. 3, No. 4, pp. 184-195 (1960).
2) McCarthy, J., et al.: LISP 1.5 Programmer's Manual, MIT Press (1966).
3) Quam, L. H. and Diffie, W.: Stanford LISP 1.6 Manual, SAILON 28.4 (1970).
4) Teitelman, W. and Bobrow, D. G., et al.: BBN LISP Reference Manual, BBN (Feb. 1972).
5) 後藤英一: Monocopy and Associative Algorithms in an Extended Lisp, 東大テクニカルレポート (May 1974).
6) LISP 1.9 User's Manual, EPICS 5-ON-2, 電子技術総合研究所 (Mar. 1976).
7) Greenblatt, R.: The LISP Machine, MIT AI Lab. Working Paper 79 (Nov. 1974).
8) Intellec 8 Mod 80 reference manual, INTEL (1972).
9) 間野，井田: マイクロコンピュータを用いた Lisp マシン ALPS/I，情報処理学会大会予稿集，pp. 711-712 (1976).
10) 井田 他: ALPS/I のバルクメモリアクセス機構と Lisp インタプリタ，ibid., pp. 713-714.
11) 重光，小林: ALPS/I 固定プログラム開発のためのサポートシステム，ibid., pp. 715-716.
12) 井田: Lisp マシン ALPS/I，情報処理学会記号処理研究委員会 51 年度報告集，pp. 142-159, (March 1977).
13) 黒川利明: Lisp のデータ表現，情報処理，Vol. 17, No. 2, pp. 127-132 (Feb. 1976).
14) Morris, R.: Scatter Storage techniques, Commun. ACM, Vol. 11, No. 1, pp. 38-44 (Jan. 1968).
15) Maurer, W.: An Improved hash code for scatter storage, Commun. ACM, Vol. 11, No. 1, pp. 35-37 (Jan. 1968).
16) Hopgood, F. and Davenport, J.: The quadratic hash method when the table size is a power of 2. Computer Journal, Vol. 15, No. 4, pp. 314-315 (1972).
17) 小林，山方: Simulation による ALPS/I の hashing Algorithm の分析，情報処理学会記号処理研究委員会報告集，pp. 166-176 (March 1977).
18) 後藤英一: 連載 Lisp 入門，bit, Vol. 6, No. 1 -No. 13, 共立出版 (1974-75).
19) 記号処理シンポジウム報告集，情報処理学会 (1974).
20) 長尾，中村: 高速補助記憶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, pp. 720-728 (1976).

付　録　ALPS/I 組込関数各論

ALPS/I には subr 73, fsubr 20, apval 12 個の組込アトムがあるが，このうちのいくつかを説明する．

i）**fn1* 及び **fn2*：*load* 関数によりロードされたオブジェクトに与えられる名前．

ii）*assoccomp*：1引数関数に対して連想計算を指定する擬関数．

iii）*try* 関数：条件が満たされる要素を探しだす関数．*try*[x；fn]=*prog*[[xx]；xx=x；LOOP [*atom*[xx]→*return*[xx]：*fn*[*car*[xx]]→*return* [*car*[xx]]]；xx=*cdr*[xx]；*go*[LOOP]]

iv）*walk* 及び *unwalk*：インタプリタのトレースとして *eval*, *evlis* の出入口の引数及び値の印刷を制御する．

v）**progn*：*go* 文，*return* 文を許す *progn*

vi）*seta*：ハッシュ配列要素への値のセット．参照は Harray element refer としてインタプリタ内で処理される（図 **11**）．*seta*[a；b；c]=*hstore*[*hash* [a；b]；c；*make-variant*[a；b]；(a*. b*)；8]

vii）*array*, *dearray*, *delete*：ハッシュ化配列の宣言は *array* により行う，配列の消去には *dearray* を，一部の要素の消去には *delete* を用いる．

viii）*load*：Intel 標準の 16 進数フォーマットにより，キーボードないしは紙テープからオブジェクトプログラムをロードする．

（昭和 52 年 6 月 15 日受付）
（昭和 53 年 7 月 12 日採録）