記号処理 30－2
（1984. 12. 14）

ポータブルバックエンド型LISPプロセッサALPS/IIIの構想

井田昌之
青山学院大学理工学部

1。この試作研究の背景

Lispの世代を機能の発展の点から、大掴みに三つに分けることができよう。

1) 第一世代：Lisp1.5 を中心とする1960年代のLisp

2) 第二世代：Maclisp,BBNlisp(InterLisp),Lisp1.6, などの1970年代のLisp．data typeとしては、配列、File、string、hash表、n-tuple 、bignumなどが、制御構造としては、DO,LET,CATCH/THROW、spaghetti stack などが出現した。Rutgers/UCI Lisp ,CMU Lisp,FranzLisp などはこの世代に発している。

3) 第三世代：ZetaLisp,InterLisp-Dなどの80年代の個人用のLISPシステム．window ,flavor,closure,package,多重process,などがとり入れられ、強力なワークステーション環境を提供している。日本で開発されたLispでこの世代に入るものは現在の所存在しないと思われる。

この中でCommon Lisp [1] は、自然発生的なLispの標準化への要求ともあいまって、期待が高まっているLisp仕様である。その開発もpublic domain にあるといえる（現在の仕様に上記の第三世代的機能は入っていないが、進行中との事である[5]）。

筆者の以前のLisp（ALPS/I Lisp 及び、Lisp05 ver2.）は、上記の整理にしたがえば、データタイプはおよそ満たしているが、制御構造の弱い第二世代であった。また、16bit 空間のシステムであり、Reduce程度の規模で限界であった。現在、コンパイラも動きだし(ver2.のものは[4] にて既発表）、Lisp05のver3化が進み、その上位互換系であるALPS/IIIのLisp処理系が照準内に入ったので、これらの構想を報告する。なお、これらの構想の応用面での特長は[6] でまとめているのでここでは述べない。また、内部仕様は未だ、変更の可能性があるので報告には含めない。

2。試作の目的と概要

研究の次の段階として次の3 点を目標とした。

1)メモリ空間の拡大：32bit 化

2)Lisp仕様のcatch up・改良

3)環境ソフトウェアの整備拡充

また、従来より筆者が行なってきた試作研究に区切りを付けたいので、その前提として、cpu には80286 を採用し、従来のデータを総括できるようにすることとした。この前提に立つと、1)については、32bit ポインタの表現が焦点となる。これについては、５。で述べる。2)については、Common Lisp を中心に研究することとし、４に述べる。3)については、Lisp処理系の中にツールを取り込むのでは無く、バックエンド型のプロセッサとし、unixに対するuucp型のインタフェースを用意し、関連機能はunix下に準備する／借用することとし、開発を統合的に行なえるように考えた。また、パソコンのOSに対しても端末としては接続できるはずである。コンパクトな入れ物にいれて、物理的な携帯性を与えることのできる規模にすれば、ポータブルなLispプロセッサとなる。

実験の狙いは次の3 点に置く。

i) CommonLisp仕様の解明、小型システム化での問題点（もしあれば）の解明
ii) バックエンド型のプロセッサに与えるべき機能の範囲の設定
更に、
iii)86seriesのプロセッサアーキテクチャのLisp適合性に対するデータの収集

このための実験装置として、次のような構成を持つALPS/IIIハードウェアとその上で動作するAPCL(Aoyama Personal Common Lisp, 仮称）を開発する。

ALPS/III: 独自のシングルボードコンピュータ（当面は既存ボードを利用）
cpu に80286/287 を用いる（当面は5MHz.）
1 MBのRAM を実装する。（最大16MB）
シリアルインタフェース（ホスト/ コンソールを接続。当面は232C)
プリンタインタフェースを持つ
64kBのROM(27256x2)を実装する（現在、27128x2=32kb)

APCL: CommonLispサブセット仕様
（CommonLisp自身ではサブセットを定義していない）
３２ビットポインタ構成（メモリは24bit space)
cdr coding表現
ホストインタフェース機能及びメンテナンス機能

３。ALPS/IIIのハードウェアへの80286 採用の目的と問題点

80286 採用の基本的な動機は、従来より進めてきた、8080～8086～というプロセッサを中心としたパーソナルLispシステムに対して一つの区切りをつけ、データをまとめたいことである。80386 は基本的には286 の延長であるし、一方で、v70 といったチップのマクロ仕様を検討する上からも現在86系のチップについてのデータを整理したい動機がある。これは、Lisp処理系の作成と矛盾せず、並行して行ないうるものと結論した。

286は16ビットプロセッサである。[2] などに記された特長から、一般に，次の表1 のような点を286 採用の理由づけとすることができよう。

表１　２８６の一般的な特長

1	保護機構の存在（各タスク内４レベルの階層保護。 os,system service,application service,application)
2	大きな空間：16M バイトの実メモリ（24 bit実空間の存在）
3	メモリ管理機能の内蔵と各々1GB までの仮想アドレス空間
4	86/88 との上位互換性（ソフトウェア投資の保護、開発コストの削減 システム／システム開発の連続性）
5	リアルタイム・マルチタスク機能の強化
6	システム構成の柔軟性（周辺の設計の容易さ... ）
7	処理速度の向上：パイプライン　バス，各命令所要クロック数の減少、内部の４unit化(bus unit,addressunit,execution unit,instruction unit)

しかし、従来の研究や，[2] などの資料を検討すると、286 採用の前提に立った時、次の問題点が予想できる。

i) 16bit プロセッサ(64k segment system)で32bit ポインタを効率良く扱えるか？

この点については、32bit 表現を基本的に、8bitと24bit に分ける。メモリ空間は、最大24bit で表現し、8bitでポインタの属性を表現することとした。

ii) 命令セットあるいはオペランド形式に不備は無いか？

iii) 286の 2つのモード（real mode/protected mode）のどちらがLispプロセッサに向いているか？

iv) また、表１の各項目がはたしてLispを意識した場合、どの程度有効であるのか？

v) 表2 に示すような命令所要クロック数の減少が認められるが、Lispの場合、これがどの程度の処理速度の向上になるか？（一般的に言って、 286real mode は高速の8086として機能するようになっている。インテルの資料では、最大6 倍になると記述されているが、同一クロックで考えると、3 倍程度を見込むことが推定できる。）

これらに対してのデータをALPS/IIIの試作を通してまとめたい。試作の手順は５で設定した。

表２　286 命令セットの86に対するadvantage の例

	命令	8086	real mode	protected mode
所要クロック減	MoveImmediate	10+EA	2 or 3	2 or 3
	push	10	3	3
	pop	8	5	5
	in	10 or 8	5	5
	out	10 or 8	3	3
	les or lds	16+EA	7	21
	mul (16bit,reg)	124+EA	21	21
	div(16bit,reg)	155+EA	22	22
付加命令	push immediate	—	3	3
	push all	—	17	17
	pop all	—	19	19
	imul immediate	—	21	21
	n bit shift	—	5+n	5+n
	ins	—	5	5
	outs	—	5	5
	enter	—	11(L=0)	11
	leave	—	5	5
	bound	—	13	13

4　APCL

APCLの仕様の設定にあたって最も基本的なことはCommon Lisp 仕様の捕えかたであるので、その点を中心に述べる。

Common Lisp を特長づけるものは、そのデータタイプと関数・Special Formの種類である。特に、データタイプは、コンパイラを前提として考えた場合に十分合理性のあるような体系になっておりインタプリタは、ほとんど眼中に無いと言ってもよい。また、関数の種類にも特長がある。特に、Special Formとした24種の機能は必須とされており、これが核となる。表3 にこれを示す。

表３ Nemes of all Common Lisp Special Forms([1]p57より)

block	catch	compiler-let	declare
eval-when	flet	function	go
if	labels	let	let*
macrolet	multiple-value-call	multiple-value-prog1	progn
progv	quote	return-from	setq
tagbody	the	throw	unwind-protect

また、マクロが強力に生かされ、使われているので、[1] で定義されているマクロ機能を具備することは必須となろう。たとえば、setfマクロが、generalized variableという考え方によりaccessとupdateを同一形式で扱うために、導入されている。

Common Lisp のデータタイプを図1 にまとめてみた。この図に適合しないtype hierarchyを持つ仕様の処理系はCommon Lisp とは言えないと考える。図1 の"..." は他の型があってもよいことを表している。たとえば、sequenceは、vectorとlistだけでなく将来の拡張が許されている。随所に"..." があり、openな構造である。ただし、list型は、　list=cons∪null,cons∧null=∅　とされている。型の拡張は、APCLでは考えない。極力、"..." のないすべての部分を実現する。Common Lisp に対するサブセッティングは、用意する関数等の量に関して存在するという見解である。(その意味では，例えば、現在のGCLispはCommon Lisp ではない。)

cons type の内部表現について、［１］は特にこだわっていないが、ある種の前提としてcdr codingを示唆しているように読める。
consの上位型であるsequence型に対しての関数は、必然的にlistとvectorに共通である等のこともある。
APCLでは32bit 表現を採用する。この32bit を86系で、Lisp05Ver2でのように直接ポインタをおくことは、スペース的にもアクセスコストの点でも損なのでcdr codingにする。（対の直接表現をとるとすると、直接アドレシングオペランドモードがないなどにより、メモリの中の32bit ポインタからそのcdr をとるなどのことはコストがかかりすぎる。）

pathname type により、外部ファイルアクセスの処理系独立化ができる。これは可搬性のあるバックエンド型を志向するALPS/IIIにとって好ましい。random-stateは乱数のため、streamは標準入出力のため、package は旧来のoblist機構のモジュラーな拡張である。hash-tableはALPS/I以来のhashed-arrayと同じ機構である。

float typeだけ4 つの副型を示し、表記の順にIdentical としてもよいとされている［１］が、全体としてのバランスからみると多少細かすぎるように思える。

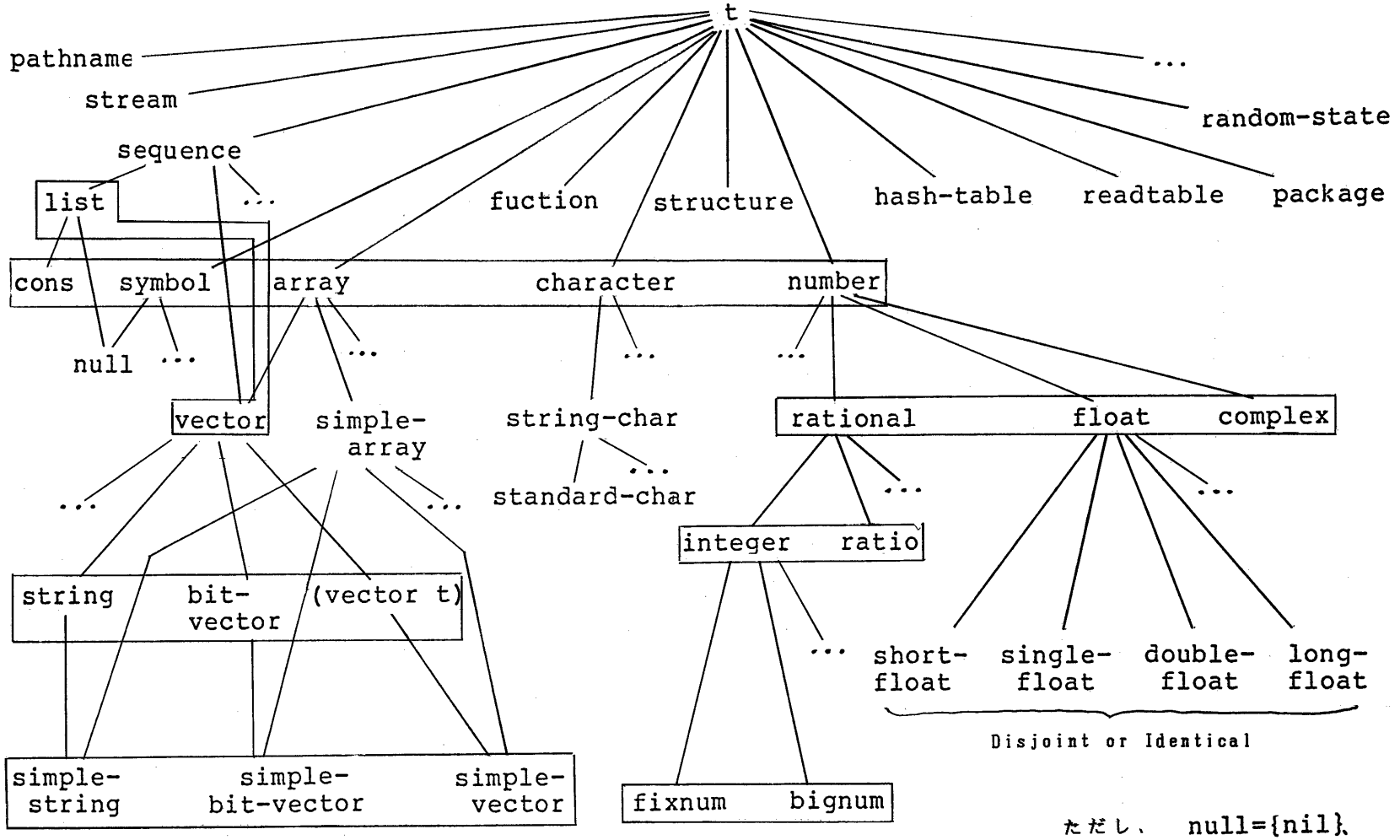

図1　Common Lisp の　Type Hierarchy

ただし、　null={nil}、

□ は　Disjoint、

すべての型の副型として、

nil 型がある。

図１の中で筆者にとって新しいものは多く、 package,readtable,structure (by defstruct), simple-array(コンパイラを意識した大きさが動的にかわらない配列), bit vector,(vector t),ratio,4 種のfloat,complex である。

また、次に示す共通仕様を表すデータ型commonを保持することを[1] は要求している。

```
common =
    { cons,symbol,(array t),(array subtype of common),string,
      fixnum,bignum,ratio,short-float,single-float,double-float,
      long-float,(complex subtype of common),standard-character,
      hash-table,readtable,package,pathname,stream,random-state,
      all types created via defstruct }
```

但し、(complex subtype of common) は、commonの定義にnumberが含まれていないからであろう。

従って、このcommonを満たすことがdata type に関するminimum requirementsであると考えられるのでこれらをAPCLは保持することとする。

なお、型に関して補足すれば、すべての型の副型としてnil 型を定義している。かなり形式的な体系に気を付けられている。図1 を見ればわかるように、simple-vector,simple-string,simple-bit-vector の組はvector及びsimple-arrayのsubsetとなっているが、この点を除けばdata type はlattice になる。

５．試作の手順

APCLの開発は、リロケータブルライブラリ／リンカを中心にしたコンパイラベースの環境を核とする。（現在はLisp05ver2との連続性からLINK86,LIB86 for RASM/RMAC and DRCを利用）。

基本的な関数はアセンブラもしくはＣで記述しROM に常駐させる。以後の関数はLispで記述しコンパイルする。

現在のコンパイラはメモリに展開するだけであるが、package typeの確立と共に各オブジェクトのリロケータブルファイル化ができるので、それにあわせてコンパイラを作りなおし、この時点で自分自身の記述が可能となる。

なお、［１］では引数の個数を５０以上許す（何故５０という数なのか不明）などと定めているのでLisp05ver2のコンパイラ／インタプリタのようにレジスタに引数を置くことはできず（もっとも32bit 化により、AX～DXを引数レジスタとするのは、既に困難。やるなら、287 の8 個のレジスタである),APCLでは、スタックにおくように設計変更する。

ハードウェアの試作については、全体を四つの期間に分ける。

i) 第一期：real mode286を用い、Lisp05ver3.0(on 86) と並行して試作する。
ii) 第二期：protected modeに拡張し24bit 空間とする。
iii)第三期：特殊ハードウェアの付加／命令セットの変更を検討し行なう。
iv) 第四期：総合的な評価を行なう。

ここでは、第一期と、第二期の置けるポインタ構成の考え方に絞って記述する。

i)第一期real mode での32bit ポインタの構成

real mode では図2 に示すように、20bit 空間（1MB)である。このため、ポインタ

の意味付を、図3 のように設計した。32bit ポインタの内、2byte目の4bit が未使用となるが、そのコストを払うだけで、アーキテクチャとの適合と第二期以降への拡張の連続性を持つことができる。

なお、オペランド形式では、現在の時点で既に、直接アドレッシングが無いので不便ではないかということが予想されているが、命令セットが拡張できないという前提にたてば、Lispデータのとりだしのためには、LES 命令などを毎回実行させるコストをかけることになるので、その効罪を調べる。

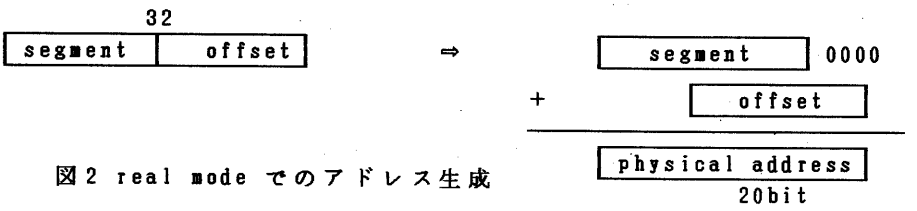

図2 real mode でのアドレス生成

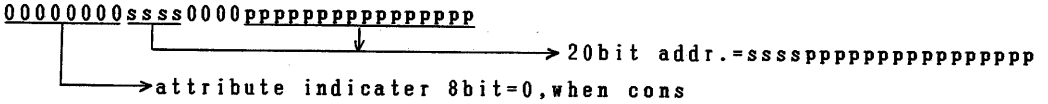

図3 real mode でのポインタ

ii) 第二期protected modeでの32bit ポインタの構成

real mode とprotected modeでは命令中のアドレッシングに対する物理アドレスの計算方法が全く異なっている。real mode の286 (及び8086) では、セグメントレジスタの内容が直接物理アドレスの形成に用いられる。これに対して、protected modeではセグメントレジスタの値はセグメントディスクリプタへのポインタ(14bit) と付加情報(2bit)として用いられる (図4)。これにより、address 空間は24bit に広げられ、16MBとなる。

ポインタ表現のセグメント値の部分は、selectorと呼ばれる。selectorが選択したディスクリプタが存在しない／誤っているのであればトラップする。この機構は実行時の型の判別に役立てられる。

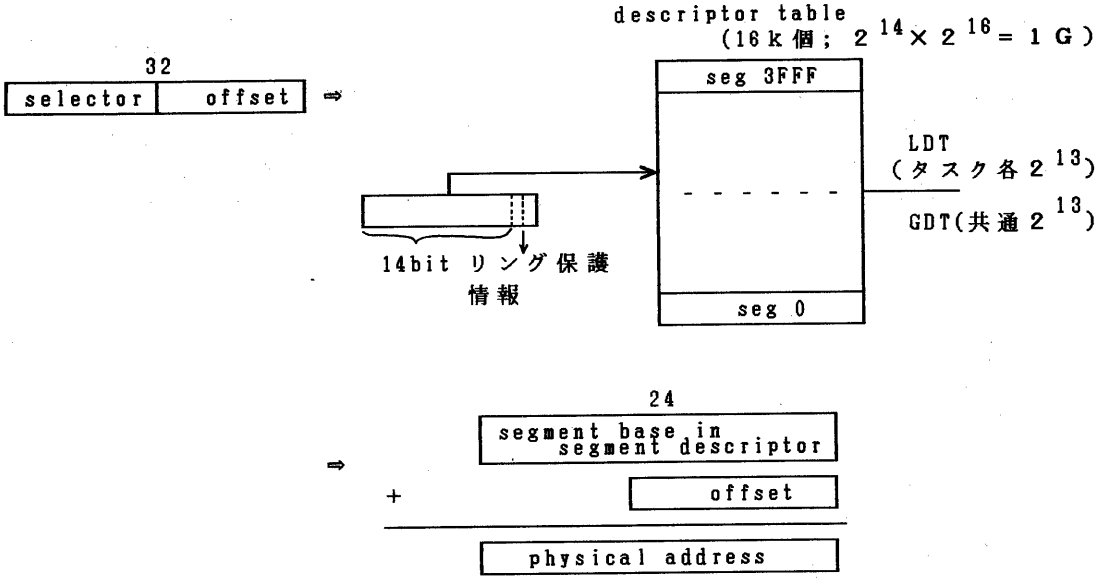

図4　protected modeでのアドレス生成

6．おわりに

ALPS/III及び、その上で動作するAPCLの構想についてまとめた。現在、5．で述べた第一期にあり、286/287 を中心とする試作は入手した既存ボードのオンボードモニタの改良を進めている段階である。Lisp仕様については、Lisp05ver3.0のLisp処理系コンテストへの対応を経て進めている段階である。

参考文献

1）Guy L.Steele Jr.et.al., "Common Lisp :The Language" , 1984, Digital Press （拙訳、共立出版より出版予定）
2）intel , "iAPX286 Programmer's Reference Manual" , 1983
3）井田 , "8086 用試作Lispの設計とその機能" wgsym no.27-2,march 1984
4）－ , " Lisp05コンパイラの設計" 、第29回全国大会,pp365-366,sept.1984
5）－ , " Lispコンファレンスに出席して" , bit vol16,no.11 pp99-101,1984
6）－ ,"A design and Implementations of personal Lisp systems considering IE ",to be appeared on 7th annual conf.on computers&IE, march 1985