記号処理 12－15
(1980. 6. 21)

# LISPマシンELISの基本設計

日比野 靖　渡邊和文　大里延康
(日本電信電話公社　武蔵野電気通信研究所)

## 1. まえがき

LiSPは、今日、実用言語として、確固たる地位を築いている。　本格的なLiSPの応用では、速い応答と大容量のメモリが要求されるので、汎用大型機のTSSでは、この要求を満すことが困難になってきている。

このため、近年、ハードウェアの試作が容易な技術的状況を背景にして、LiSPマシンの試みが盛んである[1,2,3]。

我々は、これまで、10Kセル級のLiPQ/LiPX[4]、32Kセル級のTAO-LiSP[5]を開発し、また現在100Kセル級の著名LiSP[6]を使用しているが、次のステップとして、現在、メガセル($10^6$)級の大型LiSPマシン(ELiS: Eco LISt processor)を開発中である。

この報告では、我々のねらいをどのように設計に反映させたかを中心に、ELiSの基本方式、プロセッサアーキテクチャを紹介する。

## 2. ELiS開発のねらいと基本方式

ELiS開発のねらいは、ひとことで言えば、メガセル級の大型LiSPマシンを人工知能研究の道具として使える水準に仕上げることである。　従って、単に速度や容量だけでなく、システムとしての使い易さも重要な課題となる。

ここでは、基本方式を定めるにあたっての、我々の考察を述べる。

### 2.1 処理速度と記憶容量

計算機を設計する場合、プロセッサの速度と記憶容量のバランスは、1つの重要なポイントである。

計算機を実現する立場からすると、速度、容量、いずれも、使用しうる素子の速度、容量に制限される。　また、計算機を使用する側からの要求は、アプリケーションによって異なる。

例えば、数値計算を主とする、超高速機では、内部速度を素子限界まで引き上げ(例えば、Cray-1では12.5 nsec)ているため、おのずとメモリに対する速度の要求がきびしくなり、大容量化は難しくなっている。　一方、汎用大型機では、内部速度をある程度おさえ(20～30nsec)安価なメモリ素子と高速キャッシュの組合せがとられ、内部速度とメモリシステムの速度バランスが計られる。

LiSPマシンについて言えば、内部速度の向上よりも、メモリの大容量化の要求が強い。　このため、極端な例としてはMITのCADRマシンのように、16M語(64Mバイト)の仮想アドレスをもちながら、512Kバイトの実記憶しか実装していないものもある。

ここで、メモリ素子側からの限界を考えてみる。　主記憶に、高速スタティックMOS RAMを全面的に採用するのは、コスト的に難しいので、16Kビットないし64Kビットのダイナミック MOS RAMということになる。　この種のRAMは、アクセスタイムでクラス分けされているが、最も高速のものは100 nsec台である。

このクラスの素子を用いれば、16Kビットのものを用いても、256Kバイト/ボードの構成で、8Mバイトの容量をもつメモリが、誤り訂正符号付きで、アクセスタイム400 nsec以内で実現できる。

この容量は、メガセル級のLiSPマシンの1つの目標点であり、また、このアクセスタイムは、ビットスライス・プロセッサとショットキTTLでプロセッサを構成した場合の内部速度、160～200 nsecの2倍程度であり、よくバランスしている。　160～200 nsecの内部速度で

8Mバイトもの主記憶をつけるシステムは、汎用機としては成立しないが、LiSPマシンとしては、1つの安定した点であると考えられる。

これに対して、プロセッサにECLを用いた場合は、当然、キャッシュを用いて、主記憶のアクセスタイムと内部速度の差を吸収してやる必要がある。しかしリスト処理においては、キャッシュは、大容量でなければならず、また、キャッシュ用の素子としては、内部速度とつり合いのとれた、超高速のバイポーラ・スタティックRAMを用いなければならない。さらに、後述するように、このような、高速域では、安価な制御記憶用素子がないこともあって、ECLを用いた高速のLiSPマシンは、現時点では困難が少くない。

我々は、超高速のLiSPマシンは、次のステップの課題であると考えている。

## 2.2 仮想記憶について

我々はこれまで、表1に示すような、各々そのメモリ構成に特色のあるLiSPの使用経験をもっている。

LiPQ/LiPXは、ミニコンの上に実現されたシステムであるが、リストを半仮想記憶の上におくことができ、例えば、'箱入り娘'のような問題を解くことができる。事実、日本語Q&Aシステムの辞書として、この半仮想記憶が活躍した[7]。

一方、最近使いはじめたInterlispは、TOPS20[8]の仮想記憶メカニズムの上に乗っており、セル領域をはじめすべての領域がページングの対象となる。

Tao-Lispは、32Kセルであるが、リストの半仮想化はやめ、高速ファイル(RAF: Random Access File)を使う。

LiPQ/LiPX, Tao-Lispとも、バイナリプログラム領域は、仮想化されており、事実上、いくらでも大きなプログラムが実行できる。

表1 各種LiSPのメモリ構成

| システム名 | セル空間 | 仮想化の方法 |
|---|---|---|
| LiPQ/LiPX (PDP11/55) | 10Kセル | セル空間：半仮想化<br>バイナリ領域：仮想化 |
| Tao-Lisp (PDP11/60) | 32Kセル | バイナリ領域：仮想化<br>(リストのRAFあり) |
| Interlisp (DEC2020) | >100Kセル | on demand paging<br>TOPS20 virtual memory |

新たなLiSPマシンを設計するにあたり、仮想記憶を導入すべきか否かというのが、第2の問題であった。

確かに、リスト領域の仮想化は魅力であり、メモリの物理的な制限を気にせずに大きなリストが扱える利点は大きい。

しかし、リスト処理の内容をよく調べて見ると、本格的な数式処理などを除けば、計算の中間結果として長大なリストを保存しなければならない場合はほとんどなく、大部分はS式で表わされたプログラムにすぎない。次節で述べるように、インタプリタとコンパイラの両立が図られたシステムであれば、そのような大規模なプログラムをすべてインタプリタで動作させることはないから、リスト領域を巨大な仮想空間におく必要はあまりない。このような考察から、我々はリスト領域の仮想化は行わないことにした。仮想化を行わないことにより、当然のことながら、アドレス変換のための時間のロスもなく、主記憶のメモリ素子の速度を最大限生かすことができる。ただし、将来、メモリチップを、64Kないし256Kビットに置換えることができれば、実装上の上限が32Mバイトないし128Mバイトまで拡張できるので、アドレス空間としては、最大128Mバイトまでとれるようにしている。

一方、バイナリプログラム領域は、プログラミング上の制限を除くために、仮想化が望ましい。この場合、いわゆるpagingを行うよりは、コンパイルが関数単位で行われることから、可変長のsegmentationの方がなじみがよい。

さらに，コードを自己再配置可能にしておけば，アドレス変換の必要もなく，単に関数呼出しと復帰のときのみ，手当てをしてやればよいことになる。　また，オブジェクトモジュールの管理については，もともとLiSPでは，関数atomのproperty list（あるいは，それに相当するもの）で行っているから，バイナリプログラム領域の仮想化は，LiSPの処理系に自然な形でとり入れることができる。

実際，LiPQ/LiPX，Tao-Lispでは，ソフトウェアだけでこの仮想化をサポートしている。　ELiSでは，次節で述べるように，全面的なマイクロプログラム制御をとるので，この仮想化の処理も一層能率よく行える。

## 2.3 インタプリタとコンパイラ

LiSPマシンの研究の初期においては，LiSPインタプリタ（eval）のマイクロコード化という発想のものが多いようであるが，その後は，いわゆる，中間言語のエミュレーションを行うものが主流のようである。　MITのCADR，DeutchのMicroprogrammed Interlisp [9]，BBNのInterlisp [10]，我国では，京大のNK3など，いずれもレベルは異るが，eval全体を直接マイクロコードで書いてはいない。

これに対して，神戸大のFAST-LiSPは，evalの全面的なマイクロコード化を行い，比較的遅い内部速度（300nsec）ながら，汎用大型機のソフトウェアインタプリタをはるかに越える性能を示している。　もちろん，FAST-LiSPも，コンパイルコードは中間言語式であり，この性能も大型機に匹敵する。　この結果は，高く評価されるべきものであろう。

我々は，これまで，常にインタプリタとコンパイラの両立，すなわち，S式のままの関数と，コンパイルされた関数とがどのように入り混っても，自由に実行でき，また，実行結果も同一であることをねらってシステムを開発してきた。

このねらいは，Tao-Lispでほぼ達成されていると考えているが，LiSPマシンELiSにおいても，この方針をとる。

インタプリタは，豊富なデバッグ機能をもち，かつ高速であって，会話型でLiSPを使うものにとって快適なものでなければならない。　このために，インタプリタの全面的なマイクロコード化は最も望ましい方向である。

制御記憶用の素子から考えても，従来は，バイポーラRAMしかなかったが，高速化を図った新しいプロセスによる高速スタティックMOS RAMが出現して以来，大容量の書換可能な制御記憶（WCS）を構成することは，容易になった。　特に現在，4Kビット/チップ程度であるが，16Kビット/チップが一般的になると，この傾向は一層強くなろう。

しかし，このようなインタプリタの全面的なマイクロコード化が可能なのは，マシンサイクルが，160～200 nsecの速度領域だからである。　更に高速の領域，例えば，サイクル20～30 nsecの領域では，安価なWCS用の素子はないから，どうしてもその容量が制限されてしまうので，evalの全面的なマイクロコード化は難しい。

ELiSでは，16K語（64ビット/語）のWCSをもたせ，この中に，evalと，中間言語のエミュレータの双方をおくことにしている。

# 3 システム構成

今日の技術状況では，ビットスライスマイクロプロセッサ，高速スタティックRAM，バス制御用MSIなどのロジックファミリが充実しており，サイクルタイムが160～200 nsecのマイクロプログラム制御のプロセッサを試作するのは容易である。

また，独自アーキテクチャをもつプロセッサのソフトウェア開発の問題も，市

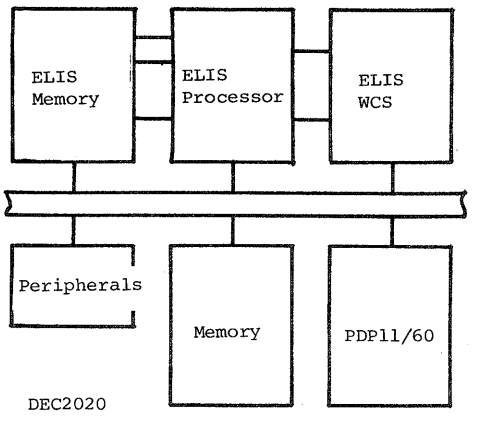


図1 ELiSのシステム構成

販コンピュータのバックエンドプロセッサとして接続する構成をとれば，そのオペレーティングシステムの機能がそのまま利用できるので，ほとんど障害にならない[11]。

さらに，ホストマシンに，バックエンドマシンの制御，診断を行わせるようにすれば，バックエンドマシンの入出力機能やコンソールパネルは省略できるのでハードウェアは必要最小限でよくなる。

ELiSのシステム構成は，このような考えに基づいており，図1に示すように，中位のミニコン(PDP-11/60)をホストとし，そのシステムバス(UNiBUS)にELiSのメモリ，プロセッサ，制御記憶を各々独立したインタフェースで接続している。

ホストは，入出力制御，ファイル管理などを行うとともに，ELiSのコンソールパネルの機能もはたす。また，将来の計画としては，DEC System 2020からのリモートアクセスも行う予定である。これらの機能は，PDP11用のリアルタイムOS，RT11のマルチタスク機能を用いて実現する。

診断モードでは，ホスト側からマイクロ命令を送り出し，ELiSの内部レジスタ，バスの状態を観察できるようにする。この機能は，ハードウェアのデバッグにも活用される。

## 4. プロセッサの基本設計

### 4.1 セルの構成とアドレッシング

プロセッサのアーキテクチャとしては，セルサイズを64ビットとし，16Mセルまでの空間をサポートできる大型のものとした。これは，将来の256Kビット/チップなどの大容量RAMの出現にも十分対応できる大きさである。

セルは，図2のように，8ビットのタグと，2つの24ビットポインタから構成されている。

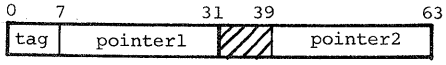


図2 セルの構成

大型のLiSPマシンの場合，アドレッシングをどう定めるかが問題となる。

メモリの読出し幅は，CAR-CDR同時読出しにするのが効果的であるとされている[ ]。従って，セルについては，アドレスはセル単位がよい。

一方，コンパイルされたコードは，LiSPマシンがスタックマシンであることから，大部分の命令は1バイト長で十分である。この点からはバイトアドレスが望ましい。

ところが，大型LiSPともなると，セルは8バイト単位となるから，バイトアドレスをとると，下位3ビットが無駄になる。

この問題を解決するために，ポインタの下位3ビットには，27ビットアドレスの上位3ビットをシフトして収容している。

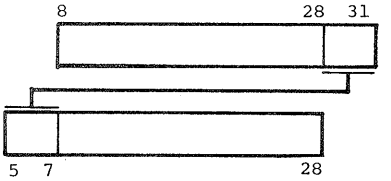


図3. セルのアドレッシング

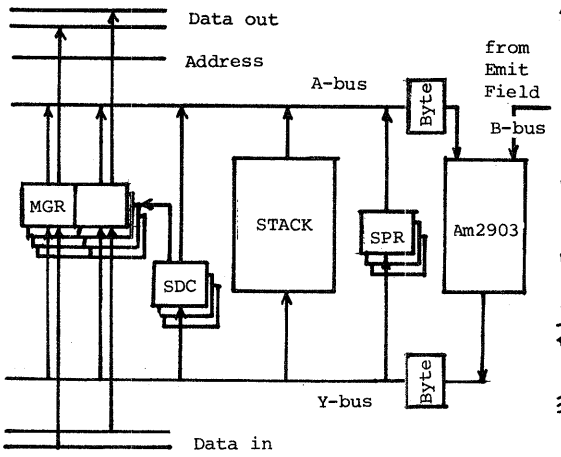


図4 ELiSのバス構成

## 4.2 メモリ多目的レジスタの導入

図4に、ELiSのバス構成を示す。設計にあたっては、LiSPマシンである特殊性と、バスアーキテクチャに適合したマイクロプログラム制御技術との調和をはかることを主眼とした。

このため、

1) 単純なバス構成とすること。
2) 制御面から見て同一の機能をもつものは統一すること。
3) ポインタたどりを効率化すること。

等を目的として、メモリ多目的レジスタ(MGR)という概念を導入している。

これは、8バイトのメモリデータレジスタ、2つの4バイトのメモリアドレスレジスタとして動作するものである。

このようなレジスタを4組備え、次に示すような目的で用いる。

1) CAR-CDRレジスタ
2) コンパイルコードの8バイト命令バッファ
3) プログラムカウンタ
4) 文字列処理時のバッファ

これらの使用目的は固定的でなく、マイクロ命令のアドレスソース指令と、データ行先指令で指定することにより、マシンの動作状態に応じて自由に定めることができるようになっている。

CAR-CDRレジスタが複数あることにより、セルの連続したポインタたどり、セルの先読みが効率化される。

また、MGRの指定を間接的に行うカウンタタイプのレジスタ(ソース行先きカウンタ:SDC)が用意されている。SDCは5ビットで、上位2ビットでMGRを選択し、下位3ビットでバイト位置を選択する。SDCの働きにより、命令バッファからの命令のとり出しや文字列処理が同一のハードウェアで能率よく行える。

## 3.3 スタック

スタックの実現法には、古くから様々な提案があるが、高速スタティックMOS RAMが使いうる今日では、相当容量の大きなものが容易に実現できる。

ELiSの場合、4Kビット/チップのRAMを用いて、16K語(32ビット/語)とした。また、スタックポインタとして、3本のレジスタを用意している。

このように比較的大きなスタックを用意した理由は、インタプリタとコンパイラの両立に有利なように、変数束縛にdeep bindingを用いているからである。

スタックは、制御スタックと変数スタックに分けて使うようになっており、境界レジスタによるオーバーフロー/アンダーフローのチェックを行っている。

# 5. マイクロプログラム

## 5.1 マイクロ命令形式

制御記憶は、スタックと同じ素子を用いて、1語64ビットで16K語実装される。図5に示すように命令形式は、いわゆる水平型である。AMDのシーケンサAm2909/2911の機能を用いると1レベルのパイプライン制御を行って、かつ、次アドレスの更新をインクリメンタによることができるから[13]、水平型であっても、順次実行されるマイクロ命令には、次アドレスフィールドを設ける必要

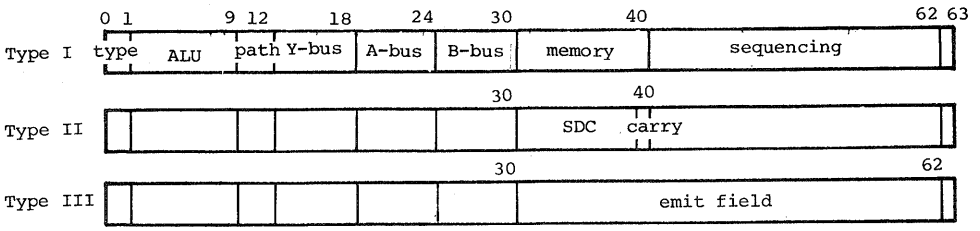


図5 ELiSのマイクロ命令形式

はない。この特徴を生かして32ビットのエミットフィールド(定数発生のためのフィールド)を設けた。定数発生機能の重要性は瀧ら[12]が指摘しているが,64ビットセルの大型LiSPマシンの場合定数も32ビットが必要になり,マイクロ命令語長が長くなってしまう。ELiSでは,マイクロ命令のタイプを分けることにより,定数発生と,メモリ操作を別タイプの命令とすることで,64ビット長とすることができた。

現在のところタイプ4は将来の拡張のために残されている。

## 5.2 マシンサイクルの制御

シーケンサでパイプラインレベルの制御を行うと,次マイクロ命令の読出しと,現在のマイクロ命令の実行が並行して行われるが,このとき問題となるのが分岐の制御である。

ハードウェアとしては,分岐を指定したマイクロ命令の,次の命令を実行後に分岐が起るようにするのが簡単であるがプログラマには使いにくい。

ELiSでは,分岐動作について,パイプラインを生かすモードと,マシンサイクルを延長して,直ちに分岐できるモードとの2つのモードを用意して,プログラマの選択にまかせることにした。

このような方法をとったのは,大規模なマイクロプログラムをコーディングする場合,ハードウェア側でできるだけ手当てをして,プログラマの負担を軽くする必要があるからである。

同様な考えで,メモリ操作についてもプログラマの負担を軽くしている。

メモリ動作は,プロセッサの内部動作と並行するようにしているが,メモリ動作が終了していないMGRを,演算のソースやデスティネーションに指定した場合,マシンサイクルを自動的に延長して,メモリの動作と同期をとるようにしている。

スタックについては,push/pop操作のために,減少後間接(push用),間接後増加(pop用)のモードを設けている。

スタックポインタを更新する時間は,演算時間のウラにかくせるので,プロセッサのサイクルタイム延長の原因にはならないが,スタックへのアクセスタイムそのものは,やはりレジスタより遅いから,基準となるマシンサイクルに納めるのは難しい。そこでスタックへのアクセルのある場合,1クロックの延長を認めることにした。

以上をまとめると,基本クロックは,60nsecで,基本サイクルは180nsecで3相クロックとし,スタックのアクセスは1クロック(60nsec)の延長,直接分岐モードでは2クロック(120nsec)の延長ということになる。メモリアクセスの場合は,待ち条件が解けるまで,60nsec単位でマシンサイクルを延長する。

# 6. LISPの仕様

ELiSで動作するLiSPの言語仕様は,TAO-Lispである。

TAO-Lispは,ELiSの試作に先だち,それ自身実用システムであるが,大型LiSPマシンの上でインプリメントされるべき新しいLiSPの言語仕様を実験

する目的で開発された。

インタプリタの構成，コンパイルの方法，EXPRとSUBR(コンパイルコード)とのリンクの方法，バイナイプログラム領域の仮想化の方法などは、そのまゝELiSでも踏襲する。

また、言語仕様のみにとどまらず，TAOで実験している操作機能は更に充実させてELiSに実現する。

## 7. むすび

以上述べた基本設計にもとづく，ELiSの諸元を表2にまとめておく。

性能としては、インタプリタで，汎用超大型機（M200クラス）の2～3倍，コンパイラで1～1.5倍程度を目標としている。　この性能は，ショットキTTLを用いた通常のマシンで実行した場合（例えば、PDP11/60上のTAO-Lisp）の、インタプリタで20倍，コンパイラで6倍程度にあたる。

目標性能が達成できるかどうかは，完成後の結果に待ちたいが，LiSPマシンというアプローチは，極めてコスト性能比のよいものであることが期待できる。

紹介したアーキテクチャの効果については、完成後の実測で明らかにして行きたい。

ELiSの設計には，瀧らのすぐれた研究の結果を参考にさせていただいた。

最後に、ELiS開発を支持して下さった山下統一第1研究室長をはじめとする研究室の諸氏、ならびに適切な御助言をいただいた竹内郁雄氏に感謝いたします。

表2　ELiSの諸元

| | | |
|---|---|---|
| プロセッサ | 内部バス幅 | 32ビット |
| | 内部レジスタ | 32語(32ビット) |
| | メモリ多目的レジスタ | 4組(64ビット) |
| | ソース行先カウンタ | 3　(5ビット) |
| | スタックポインタ | 3　(14ビット) |
| | スタック | 16K語(32ビット/語) |
| | 制御記憶 | 16K語(64ビット/語) |
| | マシンサイクル | 180n秒(可変) |
| | 制御方式 | マイクロプログラム |
| | (RALU) | (Am2903×8) |
| メモリ | アドレス形式 | バイトアドレス |
| | 読出し幅 | 64ビット |
| | アドレス空間 | $2^{27}$バイト |
| | 最大実装容量 | 8Mバイト* |
| | アクセスタイム | 380n秒 |
| | サイクルタイム | 550n秒 |
| | 誤り訂正符号 | 16ビット毎1ビット誤り訂正 |

*16Kビット/チップ使用の場合

### 参考文献

[1] Steele, G.L. and Moon A.D.: CADR, MIT AI Lab., AI Memo\WP, 1979.

[2] 瀧，金田，前川：Lispマシンの試作，情報処理論文誌，Vol.20, No.6, 1979.

[3] 長尾，中島他：LispマシンNK3のアーキテクチャとその評価，情処，記号処理研資7-4, 1979.

[4] 竹内，奥乃：LIPQリファレンス・マニュアル，通研所内資料，成果報告第13497号，1979.

[5] 竹内，大里：TAO LISPについて，情処，記号処理研資9-1，1979.

[6] Teitelman, W.: Interlisp Reference Manual, Xerox Palo Alto Research Center, 1974.

[7] Akira SHIMAZU and Hitoshi IIDA: Experimental Japanese Language Question Answering System MSSS78, Proc. of 6th IJCAI, 1979.

[8] TOPS-20 Commands Reference Manual, DEC, 1979.

[9] Deutsch, I. Peter: Experience with a Microprogrammed Interlisp System, Xerox Palo Alto Research Center 1979.

[10] Alice K. Hartley: Interlisp-11, A Personal LISP Machine, BBN, 1979.

[11] 後藤：FLATSの基本構想，bit, Vol.11, No.12, 1979.

[12] 瀧，金田他：試作LISPマシンとその評価，情処，記号処理研資7-3，1979.

[13] The Am2900 Family Data Book, AMD, 1978.