記 号 処 理　42－4
（1987. 9. 25）

“日本語上手”な Common Lisp

～ Lisp マシン Explorer* での実現 ～

大田 一久，山村 陽一，森澤 好臣
日本ユニバック

Lisp マシン KS-301 (Explorer) で日本語 Common Lisp 環境 (NCL) を実現した．Common Lisp で日本語文字が使用できるのみならず，Explorer のプログラミング環境全体を通して日本語文字の使用を可能にした．
Common Lisp で日本語文字を英数字とできる限り同等に取り扱おうとする方法は既に提案されている．これは文字コードの上限を大きくすることによって日本語文字を表現し，また文字列の表現を２種類用意することにより日本語文字を含まない場合に記憶領域の消費が増大することを避けている．筆者らの実現法もほぼこれと同等であり，日本語文字を Common Lisp のひとつの文字型オブジェクトとして扱い，文字列，シンボルにも日本語文字を使用できる．日本語文字の文字集合に含まれる英数字などの取り扱いにも考慮した．ウィンドウ・システム，ファイル・システム，ネットワークでも日本語の使用を一部を除いて可能にした．

* Explorer は米国テキサスインスツルメント社の商標である．

Using Japanese Characters in Common Lisp

An Implementaion on the Lisp Machine Explorer*

(In Japanese)

Kazuhisa Ohta, Yoichi Yamamura, Yoshitomi Morisawa

Nippon Univac Kaisha Ltd.

Japanese Language capability of computer systems have great significance in Japan. It is much more important in case of knowledge system products in which the user interface play important roles. We have built a such capabiliby in the Common Lisp language and the programming environment on the Lisp machine Explorer. This facility is called Nippongo Common Lisp Environment (NCL). This paper describes the basic idea of the implementation and fetures added to the Common Lisp language. And also mentions the input method of Japanese characters and topics related to Lisp machine specific features.

* Explorer is a trade mark of Texas Instruments Incorporated.

1．はじめに

Lisp マシン KS-301 (Explorer) 上に日本語 Common Lisp 環境（NCL）を実現した．Common Lisp で日本語文字が使用できるのみならず，Explorer のプログラミング環境全体を通して日本語文字の使用を可能にした．

Common Lisp で日本語文字を英数字とできる限り同等に取り扱おうとする方法は [1] [2] で提案されている．これは Common Lisp では文字コードが特に規定されておらず，文字コードの上限が処理系に依存して決められることを利用している．また，文字列の表現を２種類用意し，日本語文字を含まない場合に記憶領域の消費が増大することを避けている．

筆者らの実現法もほぼこれと同等である．すなわち，要素当たり２バイトの文字列を用いて日本語文字を含む文字列を表現するものである．この実現法について２章で述べる．さらに日本語文字コードとしてはＪＩＳ X0208 を用いているが，これはＡＳＣＩＩに含まれる文字をすべて含んでいる．ExplorerではＡＳＣＩＩを拡張した Lisp Machine 文字（ＬＩＳＰＭ文字）コード集合を用いており，ＡＳＣＩＩに相当する文字が２種類存在することになる．この問題およびその対処について３章で述べる．また，ウィンドウ・システム，ファイル・システム，ネットワークでも日本語の使用を一部を除いて可能にした．この点について４章で述べる．５章では日本語文字の入力法について述べ，最後に各種のユーティリティでの日本語の使用について６章で簡単に述べる．

2．Common lisp と日本語文字

多くの汎用プログラミング言語は数値の処理をその主な目的とし，文字，記号の処理は副次的な機能であることが多い．Lisp は記号処理言語として設計され，他の多くのプログラミング言語とは異なった特徴を持っている．これらの特徴の内，次のような点を日本語の使用を考える際に特に考慮しなくてはならない．

その第一はシンボルである．Lisp ではシンボルが非常に重要な役割を持っている．シンボルは他の言語での識別子に相当し，関数名や変数名として用いられる．また，データとして扱うこともでき，リスト構造の中ではこれ以上分割できない最小単位として扱われる．そして，シンボルの唯一性はシステムによって保証されている．他の言語では識別子にまで日本語が使用できるものは少数派であろうが，Lisp の場合必須である．例えば，次のような関数が実際に定義できなければならない．

```
(defun 関数 (引数)    ; 日本語の関数名，変数名
  (cons 引数 nil))
```

もうひとつは文字列である．Lisp の応用においては，文字列の操作が重要な位置を占める．Lisp では文字列の操作を行う多くの関数を持っている．文字列に日本語が使用できるプログラミング言語は数多くあるが，日本語文字を２文字として扱うなど不充分な場合が多い．Lisp では日本語文字が一つの文字型のデータとして扱うことができ，文字列の１要素となり得ることが必要である．例えば，文字列に関して次のような結果が得られるようにすべきである．

```
(length "日本語 Common Lisp 環境")   => 18
(char "日本語 Common Lisp 環境" 16)  => #\環
```

これは，ファイル・システムなどでの日本語文字の表現とは独立に考えるべきである．

さらに，Lisp ではオブジェクトの印書を文字列に対して行うことができる，この場合印字表現が文字列の内容となる．この文字列はもちろん Lisp で取り扱うことができ，またこの表現を文字列から読み込むことができる．したがって，日本語文字を含むオブジェクトの印字表現は上の条件を満たしていなければならない．例えば，日本語のシンボルの印字表現は日本語の文字列になる．

```
(print-to-string '文字列)  => "文字列"
```

これは，ファイル・システムなどでの日本語文字の表現との変換を印字表現を取り扱う部分でなく，入出力の低いレベルで行う必要があることを意味している．

Common Lisp では特定の文字コード集合を規定しておらず，文字コードの上限は処理系に依存することになっている．そこで，この定数の値を大きくすることにより，文字種の多い文字セットを Common Lisp で扱うことができる．インプリメンテーションとしては文字オブジェクトの表現のうち，コードのために使用する部分を拡張することにより可能となる．

日本語文字の場合，文字コードのために２バイト用意すればよく，日本語文字を一つの文字オブジェクトとして扱うことができる．また，ビット属性およびフォント属性を持たない２バイトのコードのみからなる文字を文字列文字（string-char）型とし，これを要素とするベクタを文字列型とすればよい．シンボルの印字名は文字列であるので，読み込み時の

構文属性の問題を除けば日本語文字を含むシンボルを使うことができる．このとき，従来使用していた英数字の文字コード集合と，新しく追加しようとしている日本語文字の文字コード集合に共通部分がなければ，両方を同時に用いることできる．また，英数字と日本語文字の混在する文字列を表現することが可能である．

一方，既存のプログラム，特に処理系の内 lisp で記述してある部分を考えると，用いられている文字列はシンボルの印字名を含めて全て日本語文字を含んでいない．また今後新たに開発されるプログラムについても，システムで定義されている関数等がかなりの部分を占めると考えられる．この方法では文字列はすべて２バイトのベクタとなるため，日本語文字を含まない文字列については従来の２倍の記憶領域が必要になる．

この問題に対する一つの解決は，文字列の表現を１バイトのベクタと２バイトのベクタの２種類用意することである．すなわち，英数字のみからなる文字列は１バイトのベクタで表現し，日本語文字を含む文字列は２バイトのベクタで表現する．このとき，それぞれを現在の文字列型の副型として定義する．そして，現在の文字および文字列操作を新しい型と従来の文字型の両方に適用可能な総称的な操作として定義する．この方法では日本語文字を含まない文字列の記憶領域の消費を現在と同じに押えることができる．

Explorer では文字オブジェクトは固定長整数（fixnum）と同様の即値表現を持っている．この中でコード属性とフォント属性はとなりあって８ビットづづを用いて表現されている（図１）．これは同時に１２８種類の異なったフォントを表現できることを意味しているが，これはやや非現実的である．実際標準で提供されるフォントの種類はこれよりはるかに少ない．フォント属性が実際に使われるのはウィンドウ・システムと Zmacs エディタであり，これらのシステムで通常同時に使用できるフォントの数の上限は２６である．ウィンドウ・システムでのフォントの使用については３章で述べる．

そこで，適当な下駄を履かせることによってフォント属性を文字コードの一部として用いることができる．ＪＩＳコードを用いることにすると，２バイトの文字コードを１バイトづつに分割すると両方とも３２より必ず大きい値を持つ．すなわち，フォント属性が３２より大きい場合，日本語文字の上位バイトと見なすことができる（図２）．このような文字を long-char 型と定義する（注1.）．この表現では，long-

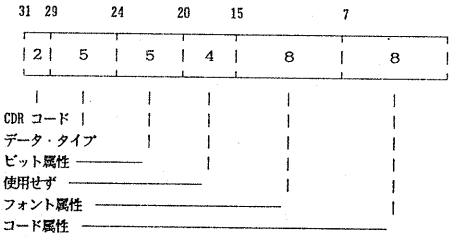
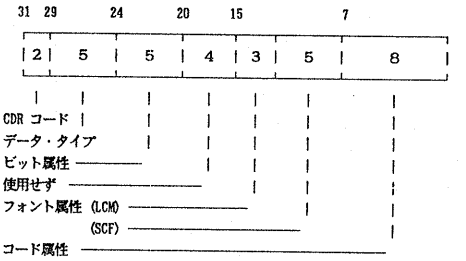


図1　従来の文字データの表現



LCM (Long Character Modifier)
　このフィールドが 0 であれば，この文字は short-char 型
　そうでなければ，long-char 型で，コード属性の一部．
SCF (Shorg Character Font)
　short-char 型の場合，このフィールドはフォント属性を表わす．
　long-char 型の場合，このフィールドはコード属性の一部．

図2　新しい文字データの表現

char 型の文字はフォント属性を持つことができない．そこで，long-char 型の文字のフォント属性は常に０であることにする．これに対して，従来の文字コードが１バイトで表現できる文字を short-char 型とする．long-char 型と short-char 型は character 型の副型であり，共通部分を持たない．

Explorer では各要素ごとにフォント属性を持つことのできる特殊な文字列がある．この文字列は fat-string と呼ばれ，ウィンドウ・システムや Zmacs エディタでマルチフォントのテキストを表現するために用いられている．この文字列は，各要素当たり２バイトの領域を持つので，long-char 型の文字のコードを一つの要素に格納することができる．short-char 型の文字は上位バイトを０とすることで表現できるので，両方の混在する文字列を表現することができる．また，上位バイトが３２より小さい値を持つ場合は，フォント属性付の short-char 型であることが判定できる．したがって，従来の使用法とも矛盾を起こさない．この文字列を

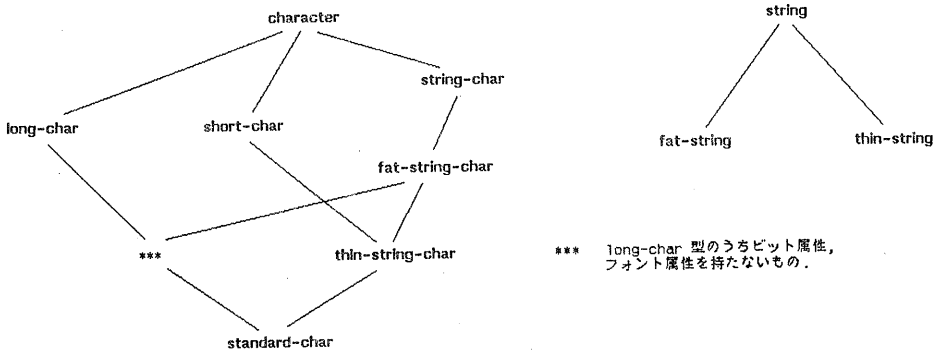


図３　型階層

fat-string 型と定義する．

fat-string 型の文字列は見かけ上通常の文字列と全く同様に扱うことができる．したがって，fat-string 型は string 型の副型と定義するのが自然である．これに対して従来の１バイトの文字列を thin-string 型と定義する．それぞれの要素となり得る文字を fat-string-char 型と thin-string-char 型と定義する．fat-string 型と thin-string 型は共通部分を持たないが，thin-string-char 型は fat-string-char 型の副型になる．さらに，thin-string-char 型は short-char 型のうちビット属性，フォント属性を持たないものと一致する．これに対して，fat-string-char 型はビット属性を持たない long-char 型と short-char 型の和集合になる（図３）．

Lisp では整数の表現を２種類持つことが一般的であり，通常はその区別を使用者は意識する必要はない．文字列についてもこの類似として考えることができる．ただし，文字列の場合その一部を破壊的に書き換える操作が可能であるが，thin-string 型の文字列の要素に fat-string-char 型の文字を代入することは不可能である．したがって，要素に対する代入が行われる可能性のある文字列は fat-string 型であることが望ましい．

しかし，文字列が作られた時点で，この可能性を判定することはシンボルの印字名などの場合を除いて一般には不可能である．したがって，安全のために fat-string 型をデフォルトとして文字列を作ることになるが，これは日本語を処理する必要のない場合まで無条件に２倍のメモリを消費することになるので望ましくない．

そこで，現在は文字列定数を読み込んだ場合，その中に long-char 型の文字が含まれていなければ，thin-string 型の文字列を作る．また，make-string のような文字列を作る関数はプログラマが明確に指定しない限り thin-string 型の文字列を作る．これに対して with-output-to-string などの場合，印字表現を予想できないので fat-string 型の文字列を作る．どちらをデフォルトとするかをグローバル変数で制御する方法も考えられるが，採用していない．

シンボルの印字名は string 型であればよいので fat-string 型の印字名を持つシンボルを作ることができる．ただし，このようなシンボルを読み込むためには long-char 型の文字の構文属性を定義しなければならない．Common Lisp では，標準の文字集合を standard-char 型として定義しており，これらの文字に基づいて Common Lisp の構文が定義されている．これらの構文属性はリードテーブルと呼ばれる表に登録されている．標準文字以外の文字についても構文属性を定義しなければならないので，各処理系はこの表に string-char 型の文字をすべて登録するのが普通である．しかし，日本語文字の数は数千個にのぼるため，この方法は現実的ではない．この問題に対する解決法は，３章で述べる．

３．ＪＩＳ文字集合を用いる場合の問題

ＪＩＳの文字集合にはＡＳＣＩＩと同じ字形をもつ文字が含まれている．例えば，アルファベット文字はＪＩＳとＡＳＣＩＩの両方に含まれている．事実，全てのＡＳＣＩＩ文字と概念的には同じ文字がＪＩＳの中に含まれている（注２）．したがって，ＪＩＳとＡＳＣＩＩを同時に使用しようとする

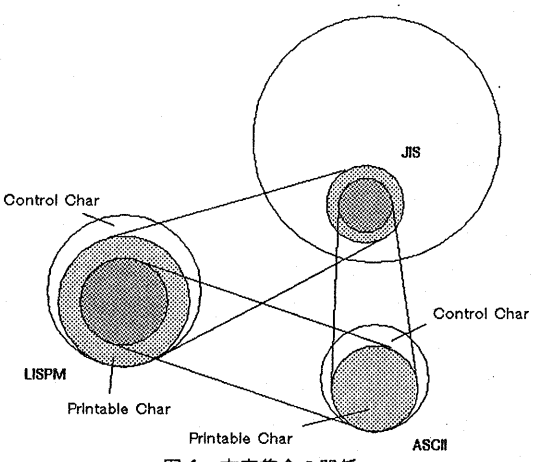


図4　文字集合の関係

と，これらの区別をどうするかという問題がある．Explorerの場合では，現在使用しているＬＩＳＰＭ（Lisp Machine）文字集合はＡＳＣＩＩを含んでいる．このＬＩＳＰＭ文字のＡＳＣＩＩを含む部分集合について，おなじ字形をもつ文字がＪＩＳに含まれている（図4）．以下，ＬＩＳＰＭ文字とＪＩＳ文字の字形の同じ文字の取扱いについて述べるが，ＡＳＣＩＩ文字とＪＩＳ文字の場合にもほぼ同様の議論が成立する．

構文に関わる場合は，ＪＩＳ文字とＬＩＳＰＭ文字の区別は混乱を招く．例えば，ＪＩＳの括弧は括弧と見なされないとすると，プログラムのソーステキスト上では括弧の対応はとれているように見えても，実際には読み込めないといったことが起こる．通常ＬＩＳＰＭとＪＩＳのアルファベットは大きさの異なる字体で印書されるが，複数のフォントを用いて印書が行なわれるような場合には，実際の字体からＬＩＳＰＭかＪＩＳかを判断するのはかなり困難であろう．また，数字の場合，ＬＩＳＰＭであれば数値になるがＪＩＳではシンボルになると言うのでは混乱を招く．したがって，ＬＩＳＰＭ文字とそれに対応するＪＩＳ文字はおなじ構文属性を持つことが望ましい．

シンボルの場合も，まったく同一の綴をＬＩＳＰＭとＪＩＳの２つの表現を考えることができる．この２つが異なったシンボルになるとすると，定義したはずの関数が未定義であるといったことが起こる．したがって，ＪＩＳで書かれたＬＡＭＢＤＡもＬＩＳＰＭで書かれた LAMBDA も同一のシンボルになって欲しい．Common Lisp ではシンボルの印字名は常に大文字に変換されるが，同様にＪＩＳ文字を対応するＬＩＳＰＭ文字に変換してやればよい．

このことから，ＪＩＳ文字のうちＬＩＳＰＭに対応するものがあるものは読み込みの際にすべてＬＩＳＰＭに変換してしまうことが考えられる．一方，このような変換を行なうと元の区別を復元することは不可能であり，無条件に変換を行なってしまうと，外部から入ってきたデータが保存されないことになる．これは Lisp 以外の既存のアプリケーションとデータを交換しようとするときの大きな障害となる．ネットワーク，コプロセッサなどによる分散処理環境を考えるときにこの性質は好ましくない．長期的には文字コード体系の標準化を検討する過程で同じ字形のコードが２種類ある問題は解決されるべきであると考えられるが，現在の段階では無条件の変換を採用するのは時機早尚であろう．

そこで，文字列として読み込まれた場合は，ＪＩＳとＬＩＳＰＭの区別を保存するが，そうでない場合はＪＩＳ文字は可能な限りＬＩＳＰＭに変換して読み込むことにした．これを実現するために，ＪＩＳとＬＩＳＰＭの対応関係に基づく変換関数を定義する．

```
(char-jis #\a) => #\a
(char-lispm #\a) => #\a
(char-lispm #\夏) => #\夏
```

ＪＩＳ文字を読み込んだ場合，この変換関数でＬＩＳＰＭに変換してからリードテーブルを参照する．ＬＩＳＰＭに変換されない文字は，構成文字とし扱う．文字列の中ではこの変換を行わない．そうすると，Common Lisp のリーダはＪＩＳ文字で表されたＳ式を読み込むことができるようになる．このとき，シンボルはＪＩＳで書かれていてもすべて印字名はＬＩＳＰＭの大文字に変換される．また，数値の表現はＪＩＳであってもＬＩＳＰＭであっても数値として読み込まれる．しかし，ＪＩＳ文字はリードテーブルに登録されていないため，読み込みマクロとして定義することはできない．

```
(eq 'ｌｉｓｐｍ 'lispm) => t
(numberp ３４) => t
```

もう一つの問題はＬＩＳＰＭ文字とＪＩＳの対応する文字を比較することである．Common Lisp では文字や文字列の比較の際に大文字小文字の区別をするかどうかによって２つの方式がある．これの類似で，アルファベットなどのＬＩＳＰＭとＪＩＳを区別するかしないかの２つの比較方式を導入した．すなわち，従来の比較関数ではＬＩＳＰＭとＡＳＣＩＩを区別し，新しく導入した比較関数では区別しない．

```
(char-equivalent #\% #\%) => t
(char-greaterp #\z #\a) => t
(char-equivalent-lessp #\z #\a) => nil
```

Common Lisp では，他にも実際の文字コードに依存することなく文字の操作ができるように各種の関数を用意している．これらの関数もＪＩＳ文字に対して一貫した結果を得るようにした．すなわち，ＪＩＳ文字に対してはＬＩＳＰＭ文字への変換関数を適用してから各種の文字関数を適用することにした．

```
(char-upcase #\a) => #\A
(char-upcase #\a) => #\A
```

ひらがなとカタカナの場合にも同様な対応関係と比較の問題がある．日本語ではひらがなは３つの例外を除いて対応する文字がカタカナに含まれている．この状況はアルファベットの大文字小文字の関係とよく似ているが，外来語はカタカナで表記するというような慣習もあり，完全に同じでよいかは疑問がある．現在のところ，シンボルの印字名を一方に変換するようなことはしておらず，次のような変換関数を用意したにとどめてある．

```
(char-hiragana #\ア) => #\あ
(char-hiragana #\a) => #\a
```

ところが，日本で用いられているＡＳＣＩＩを拡張した文字コード（ＪＩＳ x0201）には，半角カタカナと呼ばれるＪＩＳとは別のカタカナが含まれており，これが問題をややこしくする．これらのカタカナはその昔計算機での日本語処理の最後の手段として用いられたものである．このカタカナとＪＩＳのカタカナの対応を考えてやらなければならない．ここでのもっとも難しい問題は，濁音および半濁音である．というのは，半角カタカナでは濁音は濁点を続けた２文字を用いて表されている．これに対して，ＪＩＳでは１文字で濁音を表すことができる．ＪＩＳにも濁点が文字として含まれているので，２文字の表現が可能であるとはいえ，明らかに１文字の表現の方が好ましいだろう．また，ＬＩＳＰＭ文字コードでは，半角カタカナに相当する部分にＩＳＯのヨーロッパ仕様の文字を割り当てようとしている．これをどうするかという問題もあり，現在，半角カタカナは完全にはサポートしていない（注３）．

４．ウィンドウ・システム，ファイル・システム，ネットワーク

Explorer のコンソールはビットマップ・ディスプレイを用いており，ウィンドウ・システムでマルチウィンドウをサポートしている．文字の表示は各種のフォントを用いてソフトウェアで行っているため，日本語文字のフォントを用意すれば文字の表示そのものは問題なく行える．２章で述べた long-char 型の文字を日本語として表示するためには，多少の工夫が必要である．

ウィンドウ・システムではフォントを字形イメージの配列として持っている．ＬＩＳＰＭ文字の場合，文字コードを添字としてこの配列を参照することにより，その文字の字形データをビットマップとして得ることができる．ＬＩＳＰＭ文字集合はフォントの配列を有効に使用するため，通常のＡＳＣＩＩでは制御文字として用いられている部分に拡張文字を配置している．制御文字は８ビットめをＯＮにして，１２８以降の領域を用いている．これにより，フォント配列は１２８の長さを持ち，その中を無駄なく用いている．

各ウィンドウにはフォントマップと呼ばれる使用可能なフォントの表を持っており，文字のフォント属性はこの表に対するインデックスとして用いられ，実際に用いられるフォントとの対応が付けられる（注４）．このとき，フォント属性付の文字列を用いてマルチフォントのテキストをウィンドウに表示することができる．また，フォントマップ中のフォントを調べることで，ウィンドウの行の高さを決めることができる．

ＪＩＳ文字は数千個の文字から構成されているが，文字コードは連続的に使用されておらず，空き領域がかなりある．現在のフォントのデータ構造ではこのようなコードに対する字形データを保持しようとすると記憶領域が無駄になる．そこで，このような多字種文字のフォントの概念を表すために，いくつかのフォントを組にしたフォントグループを導入することにした．フォントグループはフォントの配列であり，多字種文字のの文字コードの一部を添字としてフォントグループを参照し，使用するフォントを決定する．さらに文字コードの残りを用いて得られたフォントを参照し，字形データを得る．

ＪＩＳ文字は８４の区に分かれており，それぞれは連続して文字コードを持っている．そこで各区を一つのフォントとし，フォントグループを区番で参照すればよい．区番点番は

2バイトの文字コードのそれぞれ上位バイト，下位バイトに相当し，ＬＩＳＰＭ文字の場合のフォント属性とコード属性から字形を得るメカニズムとほぼ同様である（注５）．

現在，long-char 型の文字のフォント属性は常に０であるため，各ウィンドウはただ一つのフォントグループを持つことができるようになっている．日本語文字を表示する場合も，フォントマップとフォントグループからウィンドウの行の高さを計算することができる．日本語文字のフォントグループとしては１２ｘ１２，１６ｘ１６，および２４ｘ２４の３種類の字体を用意しており，ウィンドウごとにどれを用いるかを指示することができる．

Explorer のファイル・システムでは１６ビットのデータをそのまま入出力できるファイル形式をサポートしている．これはコンパイルされたプログラムを保存するために用いられているが，文字ファイルとして用いることもできる．この属性はディレクトリに書かれており，このようなファイルをオープンすると要素が１６ビットのストリームが作られる．日本語文字を含むファイルはそのまま一文字あたり１６ビットでこの形式のファイルに格納することができる．新しくファイルを作る際は１６ビットのデータを用いることをファイルの属性として指示しなければならない．パス名には日本語文字を用いることはできない．

Explorer 同士の通信は Chaosnet を用いているが，これはコンパイルされたファイルの転送のために１６ビットデータのパケットをサポートしている．日本語ファイルを転送する場合は，同じ機構を用いて１バイト１６ビットとして行われる．メイルおよびリアルタイムの会話であるコンバースでも１６ビット文字の使用が一部を除いてサポートされている．

他のマシンとネットワークで結合する場合，日本語文字コードの表現が異なるため，文字コードの変換が必要となる．Explorer では TCP/IP をサポートしているが，TCP/IP 上での日本語文字のコードの標準がないため，日本語を含むファイルの転送はサポートしていない．TCP/IP を用いて結合する相手は主に Unix マシンであるが，その日本語文字の表現は統一されておらず，厄介な問題である．現在，EUC および SHIFT-JIS に変換してネットワーク上を転送する実験を行っている．

5．日本語文字の入力

Explorer のユーザインタフェースの最大の特長は，初心者，熟練者双方がもっている使い易さに対する，相異なる要求を同時に満たしていることである．初心者あるいは，たまにしか使わない利用者には，ポップアップ・メニューをマウスで選択する方式が適している．Explorer では，このためにサジェスチョン・メニューと呼ばれるコマンドのためのメニューを各ユーティリティごとに用意している．

一方，熟練者用としては control キー，meta キー，hyper キーといった修飾キーを用いたキーストローク・コマンドによって，キーボードから手を離すことなく，システムと対話することができる．この代表的な例が Zmacs エディタである．Zmacs エディタは，エディタ・ソフトウェアの傑作といわれる EMACS をもとに，Lisp マシン特有の機能を追加して作られたエディタである．この Zmacs の豊富な機能のなかから，文字の入力，カーソルの移動，及び簡単な編集機能を抜き出したものが入力エディタである．Explorer のキーボードからの標準的な入力にはすべて，この入力エディタが使われており，使用者プログラムからもこの入力エディタを呼び出すことができる．すなわち，使用者にはコマンドを入力する場面だろうとエディタでの文章入力であろうと，同じインタフェース（メニュー，あるいはキーストローク）で文字を入力してくことができるようになっている．

今回の日本語機能の開発ではハードウェアの変更はしない，という方針であったため，キーボード上に，カナ文字はなく，「変換キー」も勿論ない．そこで，入力エディタおよびZmacs エディタにコマンドを追加することで日本語入力のインターフェースを実現した．これをＪＩＴ（Japanese Input Toolkit）と呼んでいる．入力エディタそのものが Explorer の入力フロント・プロセッサとして位置づけられるため，日本語入力機能も各種ユーティリティはもちろんのこと，使用者のプログラムからも自由に利用することができる．

入力の方式は，ローマ字による文節単位の入力／最長一致法によるカナ漢字変換方式を採用した．現在，一部には，二文節最長一致法等による連文節変換あるいは全文変換といった機能をもつパソコンも登場しているが，現時点では，このワークステーションでの大量の文書入力といった要求は低いと判断したためである．

ＪＩＴではローマ字変換に関しては，二つのモードを設けた．英字変換モードは，入力がそのまま表示されるが，ロー

マ字逐次変換モードでは，入力された英字列がローマ字として確定した瞬間，表示がかな文字に変更されるモードである．この逐次モードは，エラーメッセージのような，文章を入力するときに便利である．Lisp のプログラムを入力する場合は依然として英数字を入力することが多く，このような状況では英字列（ローマ字列）から直接漢字に変換できると便利である．この英字モードでは，

```
(defun reidai
```

と入力した時点で変換し

```
(defun 例題
```

とすることができる．

ＪＩＴでは，原則として，Zmacs エディタ，入力エディタでリージョンとよばれる領域（文字列）をカナ漢字変換の対象としている．したがって，既に入力済みの文字列もマウスあるいはキーストローク・コマンドによってリージョンを指定し変換対象として再変換を行うことができる．また，現在入力中の文字列は，省略時の解釈規則によってリージョンが決定され変換される．したがって，Zmacs エディタ，入力エディタともに，通常の英字を入力するのと全く同じ要領でローマ字をキーインし，変換コマンドを入力することによって日本語に変換していくことができる．日本語入力のための特別なモードやウィンドウは必要ない．

カナ漢字変換でもっも大きな役割を果たす日本語辞書は，Lisp マシンの特長を生かし，効率面からも約 44,000 語の辞書をすべて内部記憶（１２８ＭＢの仮想記憶空間）上に木構造状に展開して持っている．これは約６ＭＢの領域を必要とするので，ネットワーク上に辞書サーバを設けて重複を避ける方法を検討している．

このほかに，自然言語処理システムの研究・試作を考慮して，変換の過程で得られる形態素解析の結果を，他のプログラムから容易に利用できるようにしてある．

6．プログラミング環境などでの日本語の使用

Zmacs エディタでは５章で述べたカナ漢字変換を他のコマンドと一貫した形式でサポートしている．カーソルの位置は日本語の場合も文字単位であり，英数字のマルチフォントと同時に日本語を用いることができる．Zmacs ではファイルの先頭の行を属性行として各種の情報をそこに指示することができ，そのファイルを表示するために用いられるフォントのリストもそこに記録される．日本語文字のフォントグループも属性行に記録できるようにした．

また，Zmacs では英文のテキストを取り扱うための各種の機能が用意されている．例えばカーソルの移動や文章の整形などであるが，これは英文が空白を単語の区切りとしていることに依存している．日本語のテキストを取り扱うためこれらの機能では不充分で，新たに考える必要がある．現在，このような機能はサポートしていない．

その他，インスペクタ，デバッガなどのツールでは，日本語文字の表示，入力エディタによる日本語入力をサポートしており，英字のみの場合と同様に使用することができる．また，各種のメニューなどでも日本語の表示が可能で，プログラムからも日本語の文字列を与えることにより，従来と同様の方法で使用することができる．

Explorer ではオンラインでシステムの使用法に関して各種の情報を得ることができるが，これらの場面でも日本語文字を使用することができる．例えば，関数のドキュメンテーションを日本語で与えておけば，入力エディタのコマンドでそれを表示させることができる．また，入力エディタなどのコマンドに関する情報を与えるサジェスチョンメニューでも日本語が表示可能で，カナ漢字変換に関するコマンドの説明は日本語で与えられる．（図５）

7．おわりに

Lisp マシン Explorer の日本語 Common Lisp 環境（NCL）についてその実現法を含めて紹介した．Lisp マシンという特殊な条件ではあるが，プログラム言語および環境を通して一貫した日本語機能をサポートしている．

Lisp マシンの環境はもともと非常に強力であるが，英語だけではなく日本語の環境が使用できるようになることで，我が国でもその真価を発揮するだろう．特にエキスパートシステムの応用で，日本のユーザに合わせたインターフェースが期待できる．また筆者らもソフトウェアの開発保守の場面で，電子メール，ドキュメンテーションの作成などで日本語機能の恩恵を受けている．今後さらに入力法および日本語文書のサポートなどを充実して行きたい．

日本語辞書の使用を快諾くださった九州工業大学吉田将教授，九州大学日高達助教授，九州芸術工科大学稲永紘之講師，福岡大学吉村賢治助教授，以上各位に感謝する．

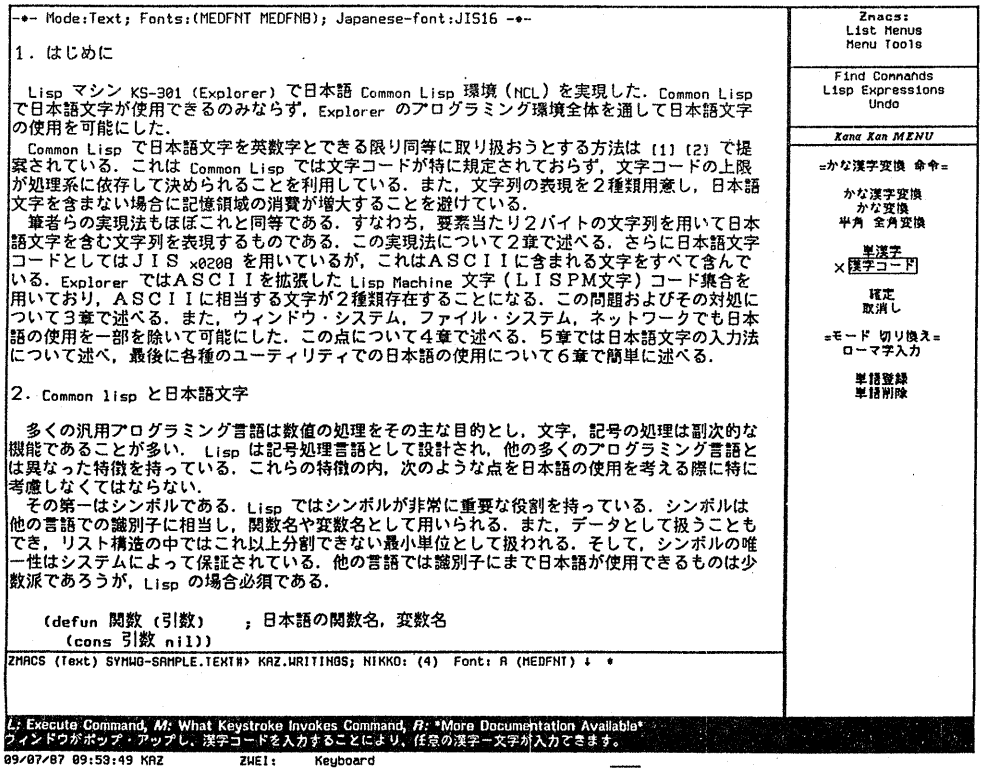


図5　日本語入力用のサジェスチョン・メニュー　(Zmacs)

注１　この表現は，ＪＩＳの文字コード体系に強く依存している．

注２　従って Common Lisp の標準文字は全てＪＩＳの中に含まれていることになる．このことは文字セットがＪＩＳのみであっても Common Lisp としてはなんら問題のないことを意味している．

注３　文字列として読み込まれたデータは保存されるが，カタカナとしては表示されない．

注４　この表の大きさは通常２６に固定されており，２６を超えるフォントを同時に使用することはほとんどない．

注５　このため，日本語テキストを表示する速度はマルチフォントのテキストの表示とほぼ同様である．

参考文献

[1] 元吉；“Common Lisp における日本語処理方式の提案”
記号処理研究会 40-6 １９８７年１月

[2] 日本電子工業振興協会；“日本語処理に関する調査”
マイクロ・コンピュータに関する調査報告書
－ Lisp 技術に関する調査 － 昭和６２年３月

[3] Steel, G.; "COMMON LISP: The language",
Digital Press, 1984
(邦訳 井田; "COMMON LISP", 共立出版)

[4] 大田他; "Explorer (KS-301)", ｂｉｔ 別冊
高機能ワークステーション，共立出版 １９８７年７月

[5] 大田；“Lisp マシンのウィンドウ・システム”
技報 UNIVAC TECHNOLOGY REVIEW No.13
１９８７年５月