MMS 21-2
(1975. 9. 30)

記号及び数式処理システム HLISP-REDUCE

後藤 英一 (東京大学 情報科学科)
金田 康正 (東京大学 物理学科)

1. はじめに

先年「記号処理シンポジウム」[文献(1)]が開かれ、さらに電々公社による計算サービスの一つとして、数式処理言語'AL'が近々公開されること、などから理解できるように、近年日本においても、数式処理システムの研究並びに開発が盛んに行われるようになった。ところで、記号及び数式処理についてのCURRENTなTOPICSは、'CACM'の分科会である'SIGSAM'の発行する'SIGSAM-BULLETINE'に多く載せられていて、それから数式処理システムの性能評価、改良による性能の向上あるいは数式処理アルゴリズムの開発等についての多くの情報を得ている。この'BULLETINE'紙をにぎわせる数式処理システムとしては、MITで開発された'MACSYMA'、英国CAMBRIDGE大学で開発された'CAMAL'、IBM社の'SCRATCHPAD'米国UTAH大学におけるA.C. HEARNの手になる'REDUCE-2'、その他'SAC-1'、'SCHOONSCHIP'等をあげることができる。それらの中の一つ'REDUCE-2'を我々[文献(2,3,4)]の開発した'HLISP'(HASH-CODED-LISP)の上にIMPLEMENTし、HLISPがFORTRAN語で記述されていることの利点を生かして、東京大学大型計算機センター(H-8800/8700)[以後簡単の為に㊧センターと記す]のほか東京大学情報科学科(H-5020)、理研、北海道大学大型計算機センター(F-230/75)等の異なる計算機の上で動作させるようにすることができた。しかも、21-KCELL程度の自由領域で、本来ならば50-KCELL程度を必要としていた問題(例えば、高エネルギー物理学におけるS-マトリックスの計算)を解くことができた。'HLISP-REDUCE'システムは、そのPORTABILITYとも相まって、かなり強力なシステムになっていることが分った。

㊧センターにおける'HLISP'をASSEMBLER語で書き直したことにより、処理速度は3倍程度向上し、入出力FILEをかなり自由に操作できるようになっている。さらに、TSSシステム下で使い易くなるように手をいれたことと、数式処理システムには必須の多倍長四則演算ルーチンをFORTRAN語、並びにASSEMBLER語で記述したことにより、数式処理システム全体としての能力は一段と向上したものと思われる。現在、'HLISP-REDUCE'システムのDEBUGを兼ねながら、実際問題への応用を考えているところであるが、以下に'REDUCE-2'そして、それのHOST言語となっている'HLISP'について述べることにする。

2. HLISP

LISP処理系を作成するにあたって、システム内部においてUNIQUENESSを保障する必要のあるOBJECT(すなわちATOM)のSEARCHは高速であることが望ましく、その為にHASH技法が用いられることが多い。ATOMのみならず、UNIQUENESSを保障する必要のある任意のS式(これをH型データ、あるいはMONOCOPYと呼んでいる。これに対して、ATOM以外のUNIQUENESSを保障する必要のないS式を、L型データと呼んでいる。)というものをLISPに導入し、それらのSEARCHに

HASH 技法を応用したものが、'HLISP'である。

UNIQUENESS が保障されるS式の導入により、2つの DATA TYPE H-MOLECULE (リスト形式の UNIQUE OBJECT)並びに ASSOCIATOR (H-MOLECULE を KEY として持っている LISP セルであり、通常の ATOM と同様に、任意の DATA TYPE を値として BIND 又は ASSIGNE することができる。ATOM を拡張したもの。) の導入が可能となり、ASSOCCOMP (ASSOCIATIVE COMPUTATION連想計算。後述。) の機能を組込むことができた。

'HLISP' は今までの LISP 1.6 の拡張であり、拡張に伴って導入された関数(例えば、H型データ同志をCONS して新しいH型データを作り出す関数、あるいはH型データであるかどうかをみる PREDICATE 等) の名前は、もともとのLISPに'H'で始まる関数はないので、'H' で始まるようにした。従って、組込関数の HCONS、HP … 等が HLISP に特有の関数であることが理解できる。以下 HLISP 全般について述べるが、組込関数、ガーベージコレクタ、その他については文献(2, 3, 4) を参照されたい。

2.1 HLISP の特徴

A) システム全体が FORTRAN 語で書かれてある為に、PORTABILITY は高い。現在の所、'HLISP'が IMPLEMENT されているのは、'HLISP-REDUCE'が動作している場所以外に、東京工業大学(F230-45S)、東京電機大学 (F230-48)、東北大学 (N2200)、電々公社(DIPS)、青山学院大学 (IBM) がある。

B) H型データの導入により、リスト型S式同志の比較は、H型データ同志であれば内部表現が等しいかどうかだけで決定してしまい、'EQUAL'を用いる必要はなく、'EQ'で充分である。これは、数式処理のように、内部表現同志の比較が多いシステムでは、速度の向上に役立つ。

C) 関数を定義する場合、その関数が'RPLACA'、'RPLACD' によって書き換えられることがないならば、その関数はH型データとして定義することができる。その際、もし関数の定義の中の部分S式の中に共通のものが多ければ多いほど、関数を定義するのに必要な領域は少なくてすむことになり、自由領域を有効に利用できることになる。その上、自由領域が足りなくなった場合に、H型データを2次記憶装置に追い出しておき、(すなわち、SEGMENT 方式に依らない、SOFTWARE に依る仮想化。) 必要になった時点で、CORE の上に再現させることが、H型データの UNIQUENESS のおかげで、高速に実行できるので、通常の LISPシステムでは定義しきれないような長いプログラムでも、HLISP では可能となる場合が生じてくる。特に、お互いに独立ないくつかの PHASE に分れているようなシステムプログラムに対しても、全体が一度に CORE にはいりきる必要がないならば、REASONABLE な速度で実行可能となる。実際には、'HLISP-REDUCE' システムがその例である。(REDUCE システムを通常のやり方で定義すると、少なくとも 20-KCELL は関数の定義の為に必要となるが、システムが大まかに3つの PHASE、すなわち、入力ルーチン、計算ルーチン、出力ルーチンに分れているので、関数の定義用に 11-KCELL もあれば動作させることが可能である。)

D) HLISP 特有の DATA TYPE、ASSOCIATOR を利用して、PROPERTY を操作する関数を定義することにより、システムプログラムの作成には必要不可欠の P-LIST を

操作する機能を高速で実行することができる。それは、通常のLISPシステムではLINEAR-SEARCHを行なっているので、PROPERTYの数が多くなればなるほど、SEARCHに要する時間は長くなるが、ASSOCIATORを用いるHLISP方式の場合では、H型データはHASHで探す為に、PROPERTYの数には独立となるからである。〔文献(3), APPENDIX F〕

E) 連想計算が行なえる。ここでいう連想計算とは次に示すことをいう。
我々がプログラムを書く場合に、FIBONACCI SERIES、ACKERMAN関数などのように、簡単で直接的なアルゴリズムであると理解し易いが、それだとRECURSIVE CALLの回数が多すぎて実用にはならない。しかし同じことを、ARRAYなどに中間結果を書えておき、適当にその中間結果を参照することによりRECURSIVE CALLの回数を減らしてスピードを向上させる、といった手法によるアルゴリズムが使える問題がある。この種の問題に対して、簡単でしかも直接的なアルゴリズムに従って計算式を定義しても、実行スピードはできる限り速める機能がHLISPに組込まれている連想計算の機能である。プログラムの方法としては、例えば'FB'と'ACKERMAN'の2つの関数を連想計算させたければ、

ASSOCCOMP (FB ACKERMAN)

というカードを一枚、'FB'並びに'ACKERMAN'の定義式のあとに挿入するだけでよい。システムは連想計算を行なう関数を評価しようとすると、まず自由領域の中に、以前計算した値がTABLEとして残っているかどうか調べ、もしあれば、TABLEに記憶されていた値を返す。もし、まだ計算していなかったのであれば、普通に評価し、その時の引数と値をTABLEにして自由領域の中にしまっておく、という処理を行なうことにより、実質的にはRECURSIVE CALLの回数、あるいは時間のかかる計算の回数を減らすこととなり、スピードが向上することになるのである。ただし注意しなければならないのは、関数の定義式の中にGLOBAL VARIABLE(S)がある場合で、この関数に対してはこの連想計算の機能は使えない、ということである。しかしこの場合は、一つDUMMYの関数を中間におくことによって、この関数に対して連想計算を行なえばよい。

F) ガーベージコレクション(GBC)は2段構えになっていて、第1段目ではH型データを2次記憶装置にはき出すことはないようになっている。第2段目に、もしH型データを2次記憶装置にはき出さざるを得なくなった場合にのみ、H型データは1度だけ2次記憶装置にはき出され、これ以降は、2次記憶装置にはき出したかどうかを調べながらガーベージコレクトを行ない、不必要な2次記憶装置への転送を少なくしている。(これができるのは、H型データがUNIQUE OBJECTであり、2次記憶装置上にコピーがすでに作られているからである。)しかし、2次記憶装置へのはき出しを行なった場合でも、ガーベージコレクション時間の全体の評価時間に対する比は3割以下であり、現在のところ支障は生じていない。

G) 多倍長整数四則演算が行なえることにより、こみいったアルゴリズムによる整数計算に対して、プログラムはFORTRAN語で行なうのと比較して簡単になるので、LIST処理のみならず、多倍長整数演算が必要な問題に対しても、このHLISPシステムは使い易くなっているものと思われる。

2.2 HLISP 組込関数/標準関数

LISP 関数は大まかに分類すると、(1) 算術関数 (2) I/O 関数 (3) PREDICATE (4) その他の4つになる。以下に HLISP に組込まれている関数の中で、特徴のあるものの説明を行なう。尚標題の標準関数というのは、EXPR、FEXPR として、システム GENERATION 時に定義される関数のことである。

1). 算術関数

多倍長整数の四則演算を行なう関数 ····· P+,P-,P*,P/,P//(REMAINDER)
同上、ただし結果は H型データ ····· H+,H-,H*,H/,H//(REMAINDER)
基本整数 ($-10^8 < n < 10^8$) の四則演算を行なう···· A+,A-,A*,A/,A//(REMAINDER)
MODULAR ARITHMETIC で使用する関数 ····· M+,M-,M*,M/(第2引数の逆数を第1引数に M* の意味でかける。)

の4種類が算術関数である。通常の四則演算を行なう関数を3種類に分けることにより、使い分けが可能となり、スピードの向上を図ることができる。また MODULAR ARITHMETIC で用いる関数を組込んであるので、MODULAR ALGORITHM による計算は高速に実行できる。(M+,..... で用いる MODULE 数はプログラムで変えることができる。)

2). I/O 関数

入出力 FILE の OPEN、CLOSE を行なう ····· OPEN,CLOSE
入出力 STREAM を変更する ····· RDS,WRS
現在のシステムの状態を FILE に DUMP したり、その FILE から DUMP 情報を読み込み、DUMP直前の状態にもどす ····· DUMP,RESTORE
SYSTEM FILE (2次記憶装置) を操作する ····· DRUMLOCK,NAMEREGISTER, DRUMSEARCH,ETC

を I/O 関数と呼んでいるが、これらを利用することにより、TSS TERMINAL からの LISP RUN,DEBUG はやり易くなる。

3). PREDICATE

内部表現(通常は POINTER)同志の比較を行なう ··· LTP,GTP,EQ,NEQ,LEP,GEP
整数(含多倍長整数)同志の比較を行なう ··· LESSP,GREATERTHANP, LESSTHANOREQUALP,GREATERTHANOREQUALP
与えられた引数が CANONICAL ORDER をなしているかどうかをみる·· ORDERP
任意の S式同志の比較を行なう ····· EQUAL,NOTEQUAL
素数かどうかをみる ····· PRIMEP
引数の CDR が NIL かどうかを調べる ····· NULLCDR
引数の CAR が ATOMかどうかを調べる ····· ATOMCAR

大小を比較する関数が2種類あるのは、1)の場合と同様、関数の使い分けによって速度の向上を図れるからである。また CANONICAL ORDER をなしているかどうかをみる ORDERP は、数式処理を行なうには必要不可欠なものである。NULLCDR, ATOMCAR は、プログラム中にしばしば出現するので、スピードの向上を考えて組込んだ。

4) その他

FORTRAN 語における 'COMPUTED GO TO' に対応する機能を持つ ··· PROG
文字を分解、あるいは合成する ····· EXPLODE, COMPRESS

HLISP システムで使用されているシステム変数（例えば、入力行のイメージを出力するかしないかを決めている変数）の値を読み出す … GETMODE
引数がリストであると、それを構成する TOP-LEVEL での ELEMENT の数を与えるが、引数が ATOM である場合、その ATOM を印刷させた場合に必要とする文字の長さを与える …… LENGTH
L型データをH型データに変換する …… HCOPY
これらは、HLISP 上でシステムを記述しようとした時に必要となるか、あるいはあると便利な関数である。

2.3 HLISP UTILITY ROUTINES

現在 HLISP システムの上で動作する UTILITY ROUTINES としては、REDUCE を別にして、以下の3つをあげることができる。(1) SYNTAX CHECKER (2) FEYNMAN DIAGRAM の数を計算するルーチン (3) 簡単な LISP-EDITOR。以下にそれらの説明を加える。

1). SYNTAX CHECKER

LISP プログラムを書く際によくひきおこす間違いとしては、(A) 括弧の付け間違い、(B) 実引数の数と仮引数の数の違い、(C) 関数がくるべき場所に、関数以外のものがきている誤り、(D) GLOBAL VARIABLE(S) と LOCAL VARIABLE(S) の混乱、等をあげることができる。これらのエラー（特に (A) ～ (C) のエラー）は、実行時にならないと見つけ出せない、という性質のものではなく、文法的な立場から、あるいは他の関数の定義式と見比べながら、関数の定義を走査するだけで判定できるものである。LISP プログラムの文法的なエラーを見つけ出し、適当なメッセージを出力しながらプログラムを診断してくれるのが、この SYNTAX CHECKER である。我々はこの SYNTAX CHECKER を学生実習に使用することにより、大幅にスループットをあげている。しかし、このような UTILITY は、学生実習用のみならずプログラム開発の道具としても便利なものである。

一方この SYNTAX CHECKER には、各 SUBR、FSUBR をそれぞれに対応する HLISP の関数番号（SUBR、FSUBR は、例えば SET は -9500 というように、整数と1対1に対応づけられていて、(SET ....) と書くよりも、(-9500 ...) とした方がスピードがあがる。）に変換した形で関数を定義し直す、SCOMPILE(S-EXPRESSION COMPILER) の機能を付加してあり、この機能を利用すると実行速度を数%程度向上させることができる。

2). FEYNMAN DIAGRAM の数を計算プログラム

これは、佐々木建昭氏（理研）によって開発されたもので、物理学にでてくる FEYNMAN DIAGRAM の数並びに、各 DIAGRAM における外線との結合状態を計算してくれるものである。

3). 簡単な LISP-EDITOR

TSS TERMINAL で LISP を実行している時に、定義式が長い関数を端末から打ち込むようなことをすると、しばしば括弧のつけ忘れ、つけすぎ、入力のし忘れ、あるいは、文字の打ち間違いを犯し、それを修正するつもりで関数の再定義を行なうと別のエラーを生じたりして、時間の損失になることが多い。また LISP プログラムの DEBUG 中に、関数の定義式を変更して実行してみたいと思うことがある。このような状況のもとで威力を発揮するのが、'LISP-EDITOR' で、この

EDITORは1日でプログラムできる程度のちいさなものだが、余分な括弧を挿入したり削除する機能、括弧のレベルを揃えて印刷する機能、S式を別のS式に変換する機能、S式の挿入、削除の機能、指示されたS式を探し出す機能を持っていて、結構便利なものである。

2.4 MAN-MACHINE SYSTEM としてみたときのHLISP

HLISP システム全体の実行能率をあげる為に、COMPILE時になって始めて値の定まるシステム変数を導入してある。それ故、FORTRAN VERSIONでは、システム定数（H型データ，L型データが作成される領域の大きさを決定する定数など）の変更を行なった場合は、一度COMPILEを行なって、前記システム変数の値を決定し、その値に従ってプログラム全体を書き直す（数箇所）ことによって、初めて完全なHLISP処理プログラムが完成する。この操作は、システム定数の変更を行なった場合にだけ行なえばよいので、それほど問題はないと思われる。しかし、㊫センターにおけるASSEMBLER VERSIONは、RE-ASSEMBLEの必要がないように作成した。また豊富なJOB PARAMETERSを準備することにより、同一のOBJECTプログラムのもとでのシステムGENERATIONに対する自由度を上げることができた。

一般的にいって、システムプログラムのBATCH的な使い方とTSS下における使い方との大きな違いは、一度入力式の評価の途中でエラーが生じてしまった場合に、JOBが終了して始めてそのことが分るか、あるいは即座にそのエラーに対してUSERがRESPONCEを行なうことができるか、ということにあることは明らかである。従ってHLISPシステムもそのことを考慮して、TSS的に使用する場合と、BATCH的に使う場合とで、システムのJOB ENVIRONMENTが変わるようになっている。すなわち、TSSではできる限りエラーの回復処置を行ない、エラーが重大でない限りは引き続く入力を受けつけるようにするが、BATCHでは、SYSTEM CLEARを指示するカードが現われるまで、入力カードは読み飛ばされるようになっている。尚TSS下でもBATCH的に、またBATCH下でもTSS的処理を行なうことは可能である。

HLISPシステムはINTERPRETERであり、1文字先読みを行ないながら入力式のPARSINGを行なっている。その為入力レーチェに何ら特別な処置を行なわなかったならば、端末からS式を入力し、ETXのKEYを押し入力の終わりをシステムに知らされた際に、最後に入力した文字が、TOP-LEVELにおけるS式を閉じるリ'であったならば、そのS式のPARSINGが終了しないことになり、従ってそのS式を評価できないので計算結果が出力されずにシステムは次の入力を要求してくる。USERの心理として、LISPにしろ、FORTRANにしろ、1行分のデータを打ち込んだならば、その入力行に対する答を即座に知りたいと思うので、前記のように、次の入力を行なって初めて前の結果が出るシステムは、TSSでは使いずらく感ずることになる。このことを考慮して、ASSEMBLER VERSIONではTSS的に使う場合と、BATCH的に使う場合とでは入力レーチェの処理方法を変更するようになっている。

以上の処置をHLISP INTERPRETERに施し、LISP-EDITOR、SYNTAX CHECKER、入出力のSTREAMを自由に選べる関数の導入、並びにDUMP、RESTOREの機能等を付加することにより、㊫センターにおけるHLISPシステムはかなり使い易くなったものと思われる。

今後のHLISPシステムの改良点としては、SYSTEM INTERFACEとして大切なI/Oの

充実、関数の評価の途中でループ等の異常状態を引き起こした場合に、端末から割り込みをかけて、TOP-LEVEL にもどすか、あるいはそのまま JOB を終了させるか、など その時の状況によって プログラムの流れを左右できる割込みルーチンの組み込みが考えられる。

3. 数式及び記号処理言語 'REDUCE-2'

'REDUCE' は ALGOL TYPE の HIGH LEVEL 言語であり、'REDUCE' 語で書かれたソースプログラムを LISP プログラムに変換し、その変換されたプログラムを LISP 処理系が実行することにより結果を出力する、という手順を ふんで処理される。変換されてできる LISP プログラムは、大部分が実行時ルーチンを CALL するような形をしている。

'REDUCE' システムは自分自身の処理系を自分自身の言語で記述してあり、ソースプログラムにしておおよそ 6000 枚から成っている。これを IMPLEMENT するには、ブートストラップルーチンを LISP で書くだけでよく、UTAH 大学からはソースプログラムの形で譲り受け、こちらの LISP に合うようブートストラップルーチンを書き、S式への変換を行なった。その結果 2000 枚程度の S式となり、現在これを標準関数として登録するのに CPU TIME にして 20 秒以内で完了する。HLISP との INTERFACE をとる為の変更部分はブートストラップルーチンの中に組み入れることにより行え、FLOATING 関係の場所を除き、変換されてできた S式に手を入れることはなかった。(この作業は HLISP 以前の LISP システムを使用し、1974 2月から3月にかけて行なった。) 変換されて完成した 'REDUCE' システムを作動させてみて、システム BUG を2コ発見し、対策を講ずることにより、それ以後は安定に動作している。

'HLISP-REDUCE' システムは INTERACTIVE にも実行できるようになっていて、簡単な COMMAND (ON INT;) で BATCH 的な使い方からの切り換えを行なうことができる。しかし、現在まで㊧セレターで使用した経験からみると、BATCH 的な使い方でも充分実用になっている。

一方この言語の持つ特徴としては、まず第一に使い易い、という点である。その他パターンマッチングの機能がすぐれている、という点をあげることができるが、一般的な特徴並びに使用方法については、"REDUCE-2 USER'S MANUAL" [文献(5)] をご覧いただきたい。このシステムは 'MACSYMA' のように、大きくて、現在知られている数学公式の2割以上を記憶している、ということはなく、記憶されている公式も限られている為に、USER はいちいちと工夫してプログラムする必要があるが、これはメモリーの関係上しかたないと思われる。しかし、この言語は特殊な分野に強い、というのではなく、一般目的用のシステムであるので、広範囲にわたる USER が利用できるものと思われる。

今後このシステムを使い易くする為にも、積分ルーチン, 因数分解ルーチン等有用なルーチンの開発、さらに処理速度をあげる為に、HLISP 特有の DATA TYPE、並びに ASSOCOMP の積極的利用、COMPILER の検討、等が今後の研究課題となるであろう。

以下に㊧セレターで 'HLISP-REDUCE' システムを使用している例を示す。

② ベルヌーイ数の計算。プログラムは端末から入力した。入力部分には下線が引いてある。

```
@BEGIN NIL
          REDUCE 2 (SEP-1-75)....
@COMMENT CALCULATION OF BERNOULLI NUMBERS¥
          COMMENT CALCULATION OF BERNOULLI NUMBERS¥
@ALGEBRAIC PROCEDURE BM(A,B);BEGIN;
          ALGEBRAIC PROCEDURE BM(A,B);
          BEGIN;
@RETURN((FOR I:=1:A PRODUCT I)/((FOR I:=1:B PRODUCT I)*
@(FOR I:=1:(A-B) PRODUCT I)));END;
          RETURN((FOR I:=1:A PRODUCT I)/((FOR I:=1:B PRODUCT I)*(FOR
          I:=1:(A-B) PRODUCT I)));
                    ERROR       -9485
       M  DRUM SEARCH ERROR,R
       A  -9533
       W  BM
       R  BM
       B  PTS NOT NOT PTS CONS PTS CONS PTS PTS
       P  ENTERING REDUCE-2 FUNCTION PUTD ...
          END;
@BM(10,4);
          BM(10,4);
          210
@ARRAY B(60); B(0):=1¥B(1):=-1/2¥ FOR I:=2 STEP 2 UNTIL 60 DO WRITE
          ARRAY B(60);
          B(0):=1¥
          B(1):=-1/2¥
@B(I):=1/2-1/(I+1)-(FOR K:=2 STEP 2 UNTIL I-1 SUM BM(I+1,K)*B(K)/(I+1));
          FOR I:=2 STEP 2 UNTIL 60 DO WRITE B(I):=1/2-1/(I+1)-(FOR
          K:=2 STEP 2 UNTIL I-1 SUM BM(I+1,K)*B(K)/(I+1));
          B(2):=1/6
          B(4):=( - 1)/30
          B(6):=1/42
          B(8):=( - 1)/30
          B(10):=5/66
          B(12):=( - 691)/2730
          B(14):=7/6
                    GBC             NLCELL =      22391  NHO =      13391  NTM1 =
          B(16):=( - 3617)/510
          B(18):=43867/798
          B(20):=( - 174611)/330
                    GBC             NLCELL =      22395  NHO =      13391  NTM1 =
          B(22):=854513/138
          B(24):=( - 236364091)/2730
                    GBC             NLCELL =      22370  NHO =      13391  NTM1 =
          B(26):=8553103/6
                    GBC             NLCELL =      22367  NHO =      13391  NTM1 =
          B(28):=( - 23749461029)/870
                    GBC             NLCELL =      22375  NHO =      13391  NTM1 =
          B(30):=8615841276005/14322
                    GBC             NLCELL =      22370  NHO =      13391  NTM1 =
          B(32):=( - 7709321041217)/510
                    GBC             NLCELL =      22364  NHO =      13391  NTM1 =
          B(34):=2577687858367/6
                    GBC             NLCELL =      22354  NHO =      13391  NTM1 =
          B(36):=( - 263152715308534773 73)/1919190
                    GBC             NLCELL =      22348  NHO =      13391  NTM1 =
          B(38):=615298718506727 39/126
                    GBC             NLCELL =      22337  NHO =      13391  NTM1 =
                    GBC             NLCELL =      22348  NHO =      13391  NTM1 =
          B(40):=( - 54827369491607315630 71)/284130
                    GBC             NLCELL =      22319  NHO =      13391  NTM1 =
                    GBC             NLCELL =      22342  NHO =      13391  NTM1 =
          B(42):=5016321694904925648880 3/59598
                    GBC             NLCELL =      22339  NHO =      13391  NTM1 =
                    GBC             NLCELL =      22332  NHO =      13391  NTM1 =
```

${}_{10}C_B$ を計算する BM(A,B) の定義

デバッグモードで実行している為に出力されたメッセージ通常は出ない。

${}_{10}C_4 = 210$

誤って定義をいれたので、端末上で消した。

計算式 $B_0=1,\ B_1=-1/2,\ B_{2n+1}=0,$

$B_i = \frac{1}{2} - \frac{1}{i+1}\sum_{k=2}^{i-1} {}_{i+1}C_k * B_k \qquad i=2,4,\cdots$

GBCの情報。通常は出力されない。

@ 100!, 50!, 100!/50!, 51*52*…100, 100!/(50!*50!) の計算。

```
@BEGIN NIL
          REDUCE 2 (SEP-1-75)....
@COMMENT(BIGNUM DEMONSTRATION)¥
          COMMENT(BIGNUM DEMONSTRATION)¥
@X:=FOR I:=1:100 PRODUCT I;                                  ——— X=100!
          X:=FOR I:=1:100 PRODUCT I;
          X:=933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182826253697920827223758251185210916864000000000000000000000000
@W:=FOR I:=1:50 PRODUCT I;                                   ——— W=50!
          W:=FOR I:=1:50 PRODUCT I;
          W:=30414093201713378043612608166064768844377641568960512000000000000
@Z:=X/W;                                                     ——— Z=100!/50!
          Z:=X/W;
          Z:=30685187562549660372027304595294697392284597216846889594477869869821589587723550720000000000000
@COMMENT(Z:=100!/50! = 51*52* ...*100)¥
          COMMENT(Z:=100!/50! = 51*52* ...*100)¥
@FOR I:=51:100 PRODUCT I;                                    ——— 51*52*…100
          FOR I:=51:100 PRODUCT I;
          3068518756254966037202730459529469739228459721684688959447786986982158958772355072000000000000
@COMMENT (NOW CALCULATE 100!/(50!*50!))¥
          COMMENT (NOW CALCULATE 100!/(50!*50!))¥
@Z/W;                                                        ——— 100!/(50!*50!)
          Z/W;
          1008913445455641933348124972 56
@COMMENT(BIGNUM DEMONSTRATION END)¥
```

4. 結語

現在のところ、日本国内において、学術研究者が利用可能な数式処理システムは IBM データセンターにおける 'FORMAC' 並びに、電々公社 'AL' そして、本文に示す 'HLISP-REDUCE' のみである。('MACSYMA' は東芝の 'MULTICS' システムの上で利用可能であるが、一般には公開されていない。東洋工業の 'FAMILIA' については著者らは聞き及んでいない。) しかし、今後数式処理は、研究を行なう上での重要な手段になるものと思われる。しかるに、'MACSYMA'、'CAMAL' は共に、日本においてもソースプログラムを入手することが可能であり、かつ有能なシステムでもあるので、日本の学術文化発展の為には、身近かでそれらのシステムが利用できるようにしておくことも必要ではないかと思われる。またそうすることにより、数式処理アルゴリズムの研究も、実際にそのアルゴリズムを IMPLEMENT することでその有用性の CHECK、他のアルゴリズムとの比較が行なえることになり、今後の計算機科学の進歩発展を押し進めることになろう。

我々は HLISP システムを FORTRAN 語によって記述することにより、PORTABILITY を向上させることができたが、他の計算機に移す際に問題となったのは、マニュアルの不備による間違いを除けば、二次記憶装置に対する I/O、並びに倍長整数宣言、EQUIVALENCE 文による ADDRESS 付けの問題 (IBM) ぐらいで、ほとんど問題とはならなかった。しかし㊎センター (IBM 360/370 TYPE の計算機について考えてみると、多くの SUBROUTINE を CALL する為に、LINK に費す時間が問題となってきている。今後 STRUCTURED PROGRAMMING で分り易いプログラムを書こうとすると、たくさんの PROCEDURE/SUBROUTINE に分けざるを得なくなる場合が多くなると思われる。こ

の場合は、LINK時間はばかにならず、今後LINKに用する時間ができるだけ最少になるようなARCHITECTUREをHARD-SOFTの両面から考える必要があるのではないかと思われる。(㊎センターにおける実測によると、多倍長四則演算ルーチンに関して、SYSTEM SUBROUTINEをCALLするFORTRAN VERSIONとIN LINEに展開したASSEMBLER VERSIONとでは、10～20数倍違っていた。又I/Oに関しては10倍程度の違いがあるようである。またH-5020上のFORTRAN VERSIONでPUSHDOWNルーチンを展開することにより、2割スピードが向上した経験を持っている。)㊌センターにおけるASSEMBLER VERSIONはFORTRAN VERSIONに対して速度は3倍になる。だが、360-TYPEのREGISTERがたくさんある計算機に対しては、よく使用するシステム変数をREGISTER(S)上におくことにより、2倍程度まで速度をあげることは可能であるように思えるが、FORTRANのOPTIMIZEが相当進んでいる事実を考慮にいれると、REGISTERの数がこうしかるべき程度の計算機に対しては、ASSEMBLERで記述してもそれほどのスピードの向上は望めないと思われる。従ってREGISTERの数が少ない計算機に対して、LISP処理系のように配列の参照が多いシステムをFORTRANで書いたとしても、LINKの問題を除けば、速度に関して問題は生じないのではないかと思われる。

システム全体をASSEMBLER語に書き直すにあたっては、SUBROUTINEを一つ一つおとしていき、FORTRANとLINKをとりながらDEBUGを行ない、全ルーチンが変換されたあとでLINKに対するOPTIMIZEを行い、入出力をシステムマクロで書き直す、という手順をとったが、それほどの困難さは感じられなかった。またREGISTERSのOPTIMIZE並びにGLOBALに使用するREGISTERSをいくつか定めた為に、コンパイルリストを見ずに直接FORTRANからHAND COMPILEを行なった。作業開始からほぼ4カ月目に変換することによって発生した虫はなくなった。作業方法としては全てTSS端末を使用した。ソースの内容がLOGINする毎に大幅に変わった今回のような場合は、タイプが苦痛でなければ、TSSでやるのが便利のようである。また資源の節約にもなっている。尚現在のところ、ソースは約12000枚、OBJECTは約60K-BYTESである。

文　献

(1). 「記号処理シンポジウム報告集」 1974.7.7-9. 情報処理学会 プログラムシンポジウム委員会

(2). E. GOTO :"MONOCOPY AND ASSOCIATIVE ALGORITHMS IN AN EXTENDED LISP" TECHNICAL REPORT 74-03. INFORMATION SCIENCE LABORATORIES, THE UNIVERSITY OF TOKYO.

(3). Y. KANADA:"IMPLEMENTATION OF HLISP AND ALGEBRAIC MANIPULATION LANGUAGE REDUCE-2" TECHNICAL REPORT 75-01, IBID.

(4). M. TERASHIMA:"ALGORITHMS USED IN AN IMPLEMENTATION OF HLISP" TECHNICAL REPORT 75-02, IBID.

(5). A.C. HEARN:"REDUCE-2 USER'S MANUAL" 2ND. ED. 1973.