山形大学紀要（工学）第17巻　第1号　昭和57年1月
Bull. of Yamagata Univ., Eng., Vol. 17, No. 1, Jan. 1982

Implementation of SLISP and SREDUCE on a 32 bit Machine

Yosio TAKAHASI

Technical College, Yamagata University

(Received September 1, 1981)

Abstract

We investigate and resolve the problems which arose during the course of implementation of symbolic manipulation language SLISP and algebraic manipulation language SREDUCE on a byte machine. Some of the problems are complicated since SLISP was originaly developed on a word machine while the present host computer is a byte machine.

It is recognized that the clues to resolve such problems are both reprogramming the source program with taking the differences of operating system of two machine into account and proper transformation of the internal representations of SLISP data so as to adapt for the new host machine. We re-examine fully the internal representations of SLISP data and the system initialization routine and improve them. The resultant system program gains considerable portability. Then we explain a new built-in function of the present version of SLISP in some details as a typical example that the recursion processing is required. The new built-in functions are consist of the basic functions of Lisp and the functions for pattern matching. Both of them contribute to increase the processing ability of SLISP.

§1. Introduction

SLISP is one of the dialects of the programming language Lisp [1]～[5], [8] and was designed and programed by Suzuki [6] for NEAC ACOS 700 in 1978. All of the SLISP system program is written in ACOS−6 FORTRAN. The processing system SLISP consists of interpreter and compiler. The compiler is designed to adopt certain intermediate code as the object code and the system program provides the execution routine(called "executor") of this intermediate language. Judging from this feature of SLISP, it looks quite easy to implement SLISP on other computer system e.g. byte machine, eventhough ACOS 700 is a word machine(36 bit/word). But as described in the following section, actual work

became laborious task against such anticipation.

SREDUCE is a modified version of REDUCE 2 so as to adapt for SLISP. REDUCE 2 is a large Lisp based general purpose algebraic manipulation language which was developed by Hearn and its feature has an Algol style form. It provides several algebraic and analytic operations which are neccessary in manipulating polynomials, basic elementary functions, matrices, tensors and etc. Manipulations of such quantities will appear in a wide variety of the fields of natural science. The details on REDUCE 2 are given in Ref. 7.

The motive that we attempt to implement SLISP and SREDUCE on our computer system(FACOM M-140 F) has its origin in the following elements. Firstly the Lisp occupies an important position in the field of imformation science from any stanbpoint, namely basic knowledge, application and education. Therefore it is highly desirable that such language become to be available in our computer system. Secondly if SLISP works, then SREDUCE will also work. SREDUCE will be helpful to natural scientists and engineers. Thirdly the source program of SLISP is written in Fortran.

Nowadays it is often occurs the case that one obtains some source program which was developed on some computer system and next, one tries to implement it on any available computer system which is different from the previous one. In such a case, perhaps many problems come out during the implemementation process, even though the source is coded in standardized language like Fortran. Generally speaking, these problems will be cumbersome except for simple ones e.g. compiler or linkage editor errors. The degree of complexity of problems increases more in the case that the word length of the new host machine decreases from that of the old one compared with the reverse case. The difference of the word length between the two host machines has a significant meaning to the present investigation. It is somewhat interesting for us to deal with the implementation which corresponds to the former case(e.g. from 36 bit version to 32 bit version) from the general viewpoint of portability of the program.

This paper is intended to describe the investigation and the resolution of troublesome problems which arose during the implementation. Another aim of the present paper is to explain the new built-in functions of SLISP which become to be available in the new version. But explanation to all of them needs considerable space and therefore, we restrict ourself to choose one typical fnnction. Instead we give its detailed explanation.

Since the the knowledge on both the storage allocation and the internal representation of the SLISP data in the Fortran source plays the role of vital importance, we shall summarize it briefly in §2. The contents of Ref. 6 alone seems to be insufficient for us to achieve the actual work. In §3, we shall present the problematic points in the implementation process and explain the resolution method

of these points. We shall then exhibit in §4 a part of the source program which corresponds to the new bnilt-in function "*SUBLA*". §5 will be devoted summary and conclusion of this paper. In Appendix I, we shall state the definition of the atom of SLISP in terms of BNF syntax.

§2. Storage Allocation and Data Structure

Since both the storage allocation and the data structures of SLISP in the Fortran source are very important to implement SLISP on an actual computer, we shall give compact description about them.

2.1 Storage Allocation and System Constants

SLISP has three important system common areas; free storage area, control stack (or simply called "c-stack") area and pname area. Among them, the first

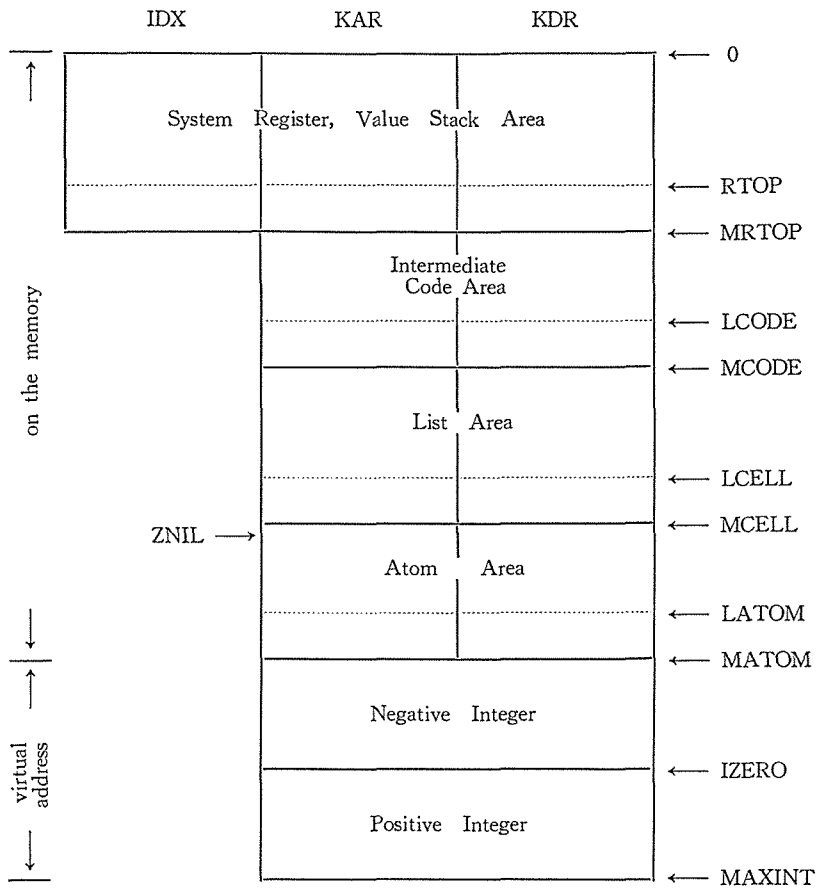


Fig. 1: Schematic storage allocation diagram of SLISP free storage area. IDX, KAR and KDR are system common areas.

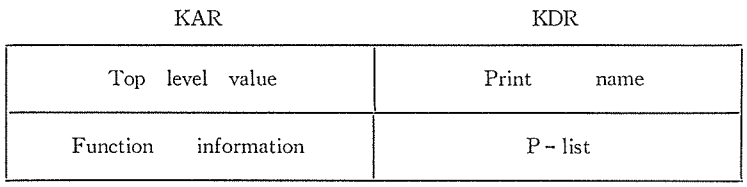
one is of particular interest for processing system. It is divided further into five parts according to the roles, namely, value stack(or simply called "v-stack) area, intermediate code area, list area, character atom area and integer atom area. While the first four areas are on the memory, the last one is in virtual address.

Some of the system constants in the Fortran source such as MCELL, MATOM, MAXINT and etc. specify the way of partitioning the free storage area(Fig. 1). Their values are given externally as the input data or calculated internally when the processing system is initialized. Other system constants those specify the c-stack capacity, the pname capacity and the internal data representations of the SLISP data are given by BLOCK DATA subprogram.

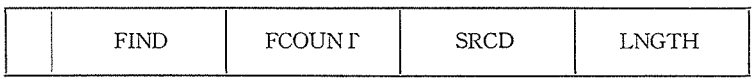
2.2 Data Types and Internal Representations

i) Character Atom: Roughly speaking, a character atom is expressed by a string of characters leading by an alphabetic letter and following any character string except for containing delimiter(s) of Lisp. But to say more precisely, a character string which contains delimiter(s) of Lisp is also a legitimate member of character atoms of SLISP if it satisfies certain definite syntax and is called "special atom" by its different appearance from the ordinary atoms. These two kinds of atoms are identically the same in data structurc, and thus we do not make any distinction between the two. In Appendix I, we give more rigorous definition of the character atom of SLISP in terms of BNF syntax.

Internal representation of a character atom is shown in Fig. 2.la-b. One character atom occupies 4 Fortran words as a whole in the free storage area and each word has definite role. The first word which is named "top level value" contains a pointer for top level value of an atom, say "X", viz. the pointer that points to the value part or the function body or the GOTO index of the Fortran



(1a). Character atom.



(lb). Fine structure of "function information" part of a character atom.

Fig 2: Internal representation of the SLISP data.

source, according to if "X" is not function or if it is one of the user-defined functions(EXPR, CEXPR, FEXPR, LEXPR) or if it is one of the built-in functions(SUBR, FSUBR, LSUBR), respectively.

When any atom, say "X" too, is first registered by SLISP input routine, the system program stores the value part of "X" with the pointer which points to "X" itself. This value is never rewritten unless some pseudo-function of SLISP operates on it. The top level value of a compiled user-defined function(CEXPR) "X" is zero initially. More precisely, when "X" is compiled, its top level value is set to zero by the system program. But when "X" is called for the first time after it has been compiled, its function body(represented by "object code") is transfered from a secondary storage into the intermediate code area and at the same time the value of "X" is replaced by the pointer for the location where transfered codes is stored.

The second word called "print name" has the form of $l \times$ IKETA + (pointer for pname area), where l is the number of printing symbols of "X" and IKETA is a system constant in the Fortran source. The fourth word "p-list" contains a pointer for property list. When "X" has no property, it points to NIL.

The third word "function information" possesses the informations about function "X". It is partitioned further in four parts i.e. FIND, FCOUNT, SRCD and LNGTH. The first part "FIND" holds function indicator that indicates function type of "X" (see Table I). The second part "FCOUNT", the third part "SRCD" and the last part "LNGTH" keeps representative information about intermediate code of the function "X" i.e. the size of intermediate code in record unit, the record number of secondary storage (direct access file) and the length of intermediate code, respectively.

The system constants K1, K2 and K3 in the Fortran source determine the way of partitioning "function information" and therefore have great importance to both the design and the implementation SLISP. Their values are fixed by taking into account of many factors such as for example, the word length of host

Function Type	Undefined	SUBR0	SUBR1	SUBR2	EXPR	CEXPR	FSUBR	FEXPR	LSUBR	LEXPR
FIND	0	1	2	3	4	5	6	7	8	9

Table I : Function type and Function indicator (FIND).

The symbol SUBRn denotes "SUBR with n arguments". When these functions are turned on the trace flag, then the new values of FIND are obtained by adding 10 to the old values.

machine, number of function types, the capacity of secondary storage and etc.

ii) Integer Atom: The data structure of an integer atom is quite simple since it is in the virtual address (see Fig. 1). Its location is obtained by adding its numerical value to the system constant IZERO in the Fortran source.

iii) Other kinds of Data: SLISP provides four other data types, namely list, intermediate code, bound variable information and control information. Their structures are shown schematically in Fig. 2.2−2.4. More detailed descriptions of them are found in Ref. 6.

In this reference, Suzuki discussed several methods to represent variable environment of Lisp and explained the reason why he chose to adopt the method to use c-stack and v-stack in designing his SLISP.

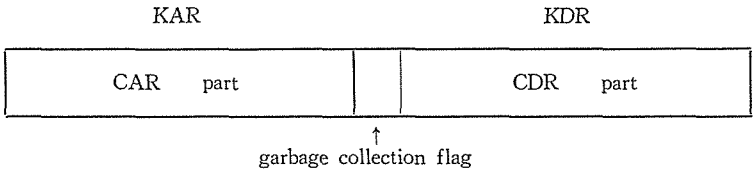


Fig. 2.2 List(cell). "CAR" and "CDR" represents car- and cdr- part of list form, respectively. The garbage collection flag occupies one bit of KDR. It is utilized by the system as a mark to represent whether given list is garbage or not.

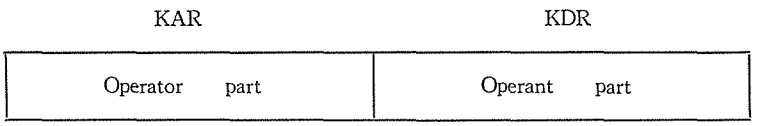


Fig. 2.3 Intermediate code.

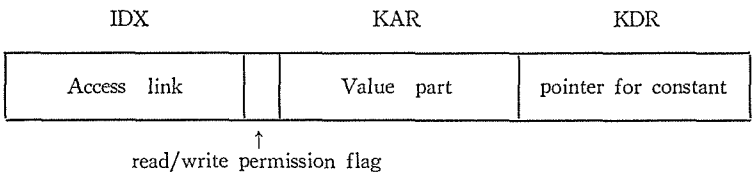


Fig. 2.4 Value stack information.

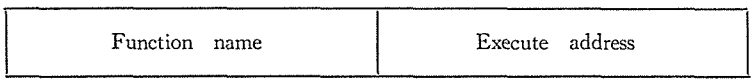


Fig. 2.5 Control stack information.

§3. Problems in the Implementation of SLISP and SREDUCE

There arose cumbersome problems as the work of implementation of SLISP on our computer proceeds. Acoording to our investigation, the cause of such problems can be classified into almost two groups. One of them concerns with the functional differences of the operating system between ACOS-6 (NEAC ACOS 700) and OSⅣ/X8 (FACOM M-140 F), where ACOS-6 is the operating system under which the original version of SLISP has been implemented and OSⅣ/X8 is the operating system of our computer. The other cause is the difference of the word length between the two machines, namely 36 bit/word(ACOS-6) and 32 bit/word (OSⅣ/X8). Even though the source program of SLISP is written in Fortran, it lacks of portability in certain parts where the system program utilizes explicitly or implicitly various characteristic features of ACOS-6 FORTRAN.

First let us describe the major difference of the two operating systems. At the earlier stage of implementation process, even the system initialization routine of SLISP did not work at all. Among many questions, the question about pointer setting mechanism for NIL(one of the most important Lisp constants) seemed to be quite puzzling and considerable effort was required to find the cause. Under OSⅣ/X8 the user program area is not zero cleared initially and maintains fragmentary records of previous jobs, while it is essential for SLISP program to work correctly that certain variables and array elements of the Fortran source should be zero cleared at the begining of execution. If the program without necessary modifications to cope with such points is executed under OSⅣ/X8, these variables and array elements are initially set wrong values which are determined by job environment at that time. Such job should be aborted or terminated with curious result because memory protection exception will certainly occur.

Our computer system does not provide the zero clear function while all of the present (big) machines do provide such function as one of the loader's options from the standpoint of job security protection. Unfortunately it is considerably hard to know which variable or array element should be zero cleared initially by the inspection of source program alone. After many trial and error, the system initialization subprogram has been reprogrammed in a satisfactory form. In the present version of SLISP system program, such variables and array elements decrease and only the arrays KAR and KDR which is declared by COMMON statement is set to zero in the BLOCK DATA subprogram.

Next we discuss the word length difference. One of the important conditions for system design of SLISP is that the value of important system constant such as *IZERO*, *IKETA*, *K1*, *K2* and *K3* are so chosen that the internal representation of SLISP data is adapted for the word length of host computer. These system constants determine the way how a given SLISP datum is represented in the Fortran system program. *K1*, *K2* and *K3* play the roles of key to utilize

intermediate codes which are saved in system reserved secondary storage. If these values do not fit the conditions of system design, all the compiled function cannot be available for us at all. Because the word length of ACOS-6 is larger than that of our machine, we must carefully re-choose these system constants, and we must reprogram certain parts of the original version without harming other elements of the system. Generally speaking, such modifications are rather difficult in the present case - from ACOS-6 version to OSⅣ/X8 version - than in the reverse case, since the word length of host machine decreases in the former case. The optimal value set of these system constants for 32 bit machine as well as the corresponding set for 36 bit machine are determined and are tabulated in Table II.

System Constant	MAXINT	IZERO	K1	K2	K3
32 bit version	$2^{31}-1$	2^{30}	2^{26}	2^{21}	2^{10}
36 bit version	$2^{35}-1$	2^{34}	2^{30}	2^{24}	2^{12}

Table Ⅱ: Several System Constants.

The optimal values of several important word length dependent system constants are shown for both the 32 bit version and the 36 bit version. Since the value of *K1* was incorrect in the original 36 bit version, it has been corrected.

By following the steps of program flow in the Fortran source, we find that several nontrivial relations exist between the system constants. These relations play fundamental roles to decide the optimal values *K1*, *K2*, *K3* and *IKETA*. They are as follows:

$IKETA > MNAME,\ MNAME\text{-}128 > ZNIL\text{-}IKETA \cdot [ZNIL/IKETA]$,
where *MNAME* denotes the capacity of pname area.

$dim(IDX) = MRTOP$.
Here $dim(IDX)$ and MRTOP means the dimension of array IDX and the depth of v-stack, respectively.

K1/K2-1:: =capacity of the system reserved secondary storage (direct access file) in record unit.

K2/K3-1:: =capacity of one executable block of compiled object in record unit.

K3-1:: =maximum allowed length of one executable block of compiled object.

The numerous knowledge about the internal structure of SLISP is of course utilized to improve the system program. The effort is concentrated mainly on the

problem to raise up the portablity of the SLISP and the problem to decrease processing times both of the interpreter and the compiler. Furthermore, by altering twelve system constants and array sizes of the Fortran source, we try to increase the user area of SLISP.

As for the implementation of SREDUCE, the actual work was easier compared with the case of SLISP. But supplementation of few functions to the SREDUCE source was required since they were used and yet have not been defined in the source.

4. New built-in Functions

We implement a total of about 30 new SUBRs and FSUBRs in order to enhance the processing ability of new version of SLISP. They are sorted in two classes, namely, the basic functions of Lisp and the functions for pattern matching. These new built-in functions are also useful to increase the efficiency of the SREDUCE. They are classified into various kinds by their operations and a good deal of space is necessary in giving explanation to all of them, even to the brief one. Thus we omit to do so here and are planning to give explanations of the new built-in functions as well as the built-in functions of the original version in a next publication.

Instead, we choose one part of the system program which describes new function "*SUBLA*" as one of the typical examples which need the recursion processing. "*SUBLA*" is a SUBR with 2 arguments and given in M-expression as follows;

$$\begin{aligned} subla[x;y] = {} & [null[x] \vee null[y] \rightarrow y; \\ & \text{atom}[y] \rightarrow [\lambda[u]; [u \rightarrow cdr[u]; T \rightarrow y][assoc*[y; x]]; \\ & T \rightarrow cons[subla[x; car[y]]; subla[x; cdr[y]]]]]. \end{aligned}$$

..... When "*SUBLA*" is evaluated, the second argument y (list form) is translated according to the substitution table x (list form of doted pairs).

As is well known, since Fortran does not allow recursive call, we must utilize certain control stack in order to realize the recursion processing. In the following, we show the system program of "*SUBLA*" in an Algol style language without using recursive call in the place of the original Fortran source for brevity and clarity.

```
integer procedure subla[x, y];
value x, y; integer a, b, u, v, x, y, z;
begin
    cpshdwn[66, 0]; u:=x; v:=y;
    if u=znil then z:=v
        else label:
```

```
begin for w: = v while w < znil do
  begin
    cpshdwn[65,cdr[v]]; v: = car[v];
  end
  if v = znil then z: = znil
    else
    begin w: = assoc*[v,u];
      if w < znil then z: = cdr[w]
        else z: = u
  end
cpopup[a,b];
for w: = a while w = 66 do
      if a = 65 then
        begin
          cpshdwn[64,z]; v: = b; goto label
        end
      else
        begin
          z: = cons[b,z]; cpopup[a,b]; w: = a
        end
          subla: = z
        end;
```

Here "*car*", "*cdr*" and "*assoc**" are other procedure identifiers which correspond to built-in functions of the original SLISP version. The procedure "*cpshdwn*" and "*cpopup*" has the operation of control stack manipulation, pushdown and popup, respectively. Furthermore the constant "*znil*" points to the SLISP constant NIL. The procedure "*subla*" returns the pointer that points to the evaluated value of "*SUBLA*".

§5. Summary and Conclusion

We have discussed in this paper problems which arose during the implementation of SLISP on 32 bit machine and resolution methods of such problems. Other material, namely the internal representation of the SLISP data, the implementation of SREDUCE and the new built-in functions have also been described too. It is shown that both variable initialization in the Fortran source(zero clear) and proper reformation of the internal representation of character atom (re-choice of the system constants and reprogramming) are indispensable to accomplish the purpose. Apparrently detailed knowledge of the storage allocation and the internal

representation of SLISP data in the Fortran source are required for better understanding of SLISP itself and still more required for completion of implementation. At present, the new 32 bit version of SLISP as well as SREDUCE works correctly under OSⅣ/X8. Since this operating system provides large virtual memory for the user, we have been extended the cell size of SLISP in the new version (49k-cell) about two times or more larger than that of ACOS-6 version (22k-cell).

By virtue of the newly implemented built-in functions and many improvements of the Fortran source (optimizaions in the source level), the processing speed of the present version of SLISP becomes fairly fast. For example, all of the processing times required to execute several SREDUCE programs which have been applied to the research problem in solid state physics decrease alout 20% or more compared with the results of the previous version of SLISP.

We made efforts to modify the SLISP source program as portable as possible. Consequently, the new version has acquired considerable portability compared with the original 36 bit version. The only operating system dependent parts are the part where time measuring system subroutines are called since these subroutines depend on the hardware. Though we have not described here, a number of debuggings, revisions and supplements have been done not only to SLISP but also to SREDUCE. Yet many other convenient functions such as for example, backtracking, multipleprecision arithmetic, pretty printer, syntax-checker and etc. are not provided in the present version. More flexible operation of I/O file access is also lacking. Perhaps the last one will be attained easily if we adopt Fortran 77 as the host language of SLISP.

Acknowledgements

I would like to express my sincere thanks to Dr. M. Suzuki who not only recognizes the SLISP implementation on our computer system but also gives me thorough guidance. Without his zeal support present implementation would not be completed. I also wish to thanks Professor S. Katsura, Miss K. Ichinoseki and Dr. T. Aoki whose supports and valuable suggestions contribute remarkably to accomplishment of implementation. In the course of this work, FACOM M-140 F (Information Engineering Dep., Technical College, Yamagata University) and NEAC ACOS 900 (Computer Center, Tohoku University) was used. This work is supported in part by the Japan Society of the Promotion of Science under Grant No. 574100.

Appendix I. Definition of Atom in SLISP

SLISP has two types of atom namely character atom and integer atom. The restrictions on the atom are relaxed further compared with those of LISP 1.5 [1].

The length of character string of an atom is allowed up to 128. The atom so-called "special atom" for convenience, is essentially the same to the ordinary atom except for containing delimiter(s). In the present version, the value range of integer atom is restricted in the interval which is determined by the word length of host machine and certain system constants. We show the definition of the SLISP atom in terms of BNF syntax as follows:

S1. <letter class> :: = one of the characters(alphabets or symbolic characters) those can be dealt with the Fortran of host machine exclusive of <blank class>, <digit>, <sign> and $ (or ¥ instead of $ for certain machines).

In the following, the replacement of $ with ¥ is assumed if it needs.

S2. <digit> :: = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9.

S3. <sign> :: = + | −.

S4. <blank class> :: = , | blank symbol.

S5. < > :: = <blank class> | < > <blank class>.

S6. <delimiter> :: = < > | . | (|).

S7. <character atom> :: = <name atom> | <special atom>.

S8. <special atom> :: = $ $ d <character string which contains delimiter(s)> d,

where d is an arbitraly character which can be dealt with the Fortran of host machine and at the same time is not contained in <character string which contains delimiter(s)>.

S9. <name atom> :: = <alphanumeric atom> | <sign> | <sign> <name atom> | <name atom> <sign>.

S10. <alphanumeric atom> :: = <letter class> | <character string which may contain $ but must not contain delimiter(s)> | <letter class> <alphanumeric atom> | <alphanumeric atom> <digit>.

S11. <integer atom> :: = <digit> | <sign> <digit> | <integer atom> <digit>.

Here let us show several nontrivial examples of <character atom>.

$1980, A5+2, X=10−2, A=B/C. All of these examples are regarded to be legitimate <name atom> of SLISP.

$ $ $SPECIAL ATOM$, $ $/ERROR!/ are both <special atom>s and the corresponding print name is "SPECIAL ATOM" and "ERROR!", respectively.

References

(1) J. McCarthy, P. W. Abrahams, D. J. Edwards, T. P. Hart and M. I. Levin, "LISP 1.5 Programmer's Manual", The M. I. T. Press(1962).

(2) L. H. Quam and W. Diffie, "Stanford LISP 1.6 Manual," Stanford Artificial Intelligence Laboratory, Operating Note 28.7(1968).

(3) D. Moon, "MACLISP Reference Manual," Laboratory of Computer Science, Massachusetts Institute of Technology (1974).

(4) W. Teitelman, "INTERLISP Reference Manual(4-th rev.)," Palo Alto Reserch Center, Xerox Corporation(1978).

(5) Y. Kanada, "HLISP and Supplementary HLISP-REDUCE Manual," Dep. of Information Science, University of Tokyo (1978).

(6) M. Suzuki, "Implementation of a Lisp Processing System and its Applications," Master's Thesis, Dep. of Applied Physics, Tohoku University(1978).

(7) A. C. Hearn, "REDUCE 2 User's Manual(2-nd ed.)," Utah Computation Physics, Report No. UCP-19, University of Utah(1973).

(8) J. B. Marti, A. C. Hearn, M. L. Griss and C. Griss, "Standard LISP Report," ACM SIGPLAN Notice Vol. 14 No. 10, 48(1979).

山形大学紀要（工学）第17巻　第１号　昭和57年１月

32ビットマシンへのSLISP及びSREDUCEのインプリメンテーション

高　橋　良　雄

工業短期大学部

要旨

記号処理言語SLISPと数式処理言語SREDUCEをバイトマシンへインプリメントする過程で生じた問題を調べ解決する。SLISPが元来ワードマシンで開発されたのに対し，今回のホストマシンはバイトマシンであるから問題点のいくつかは面到な問題である。

その様な問題点を解決する手懸りは，二つのマシンのオペレーティング・システムの違いを考慮して原始プログラムを再プログラミングする事と，新しいホストマシンに適合するようにSLISPデータ内部表現を正しく変換する事の両方にある事が認識される。SLISPデータ内部表現と処理系初期化ルーチンを良く再点検し改良する。その結果，システムプログラムは著しいポータビリティを獲得した。

次いで，再帰処理が必要となる典型的一例として，SLISP現版中のある一つの新たな組込関数についてやゝ詳細に説明する。新たな組込関数はLispの基本関数とパターン・マッチング用の関数とからなっている。これらはともにSLISPの処理能力を増強するのに寄与している。