記 号 処 理 16 − 5
(1 9 8 1. 3. 2 7)

Some Improvements on Utah Standard Lisp

Kenji Namba
SONY Corporation
Audio Technology Center
Minato-ku, Tokyo, 108 Japan

-- Abstract -- Standard Lisp on IBM 360 & 370 proposed and implemented by A.C.Hearn and J.Fitch et al. of the University of Utah had still several bugs and some execution-time inefficiency. This paper describes these bug-fixings, modifications and additions of comfortable features, and some improvements in the stack operation and compiler-interface which contribute to a speed-up of the factor of about 1.2～1.6. It also urges the necessity of variety of datatypes for a more practical Lisp system, such as vectors, hash tables, bit/byte strings.

-- Keywords & Phrases -- Lisp system, Bug-fixing, Maintenance.

§ 0 -- Introduction --

Standard Lisp (SLISP in the sequel) was proposed and implemented by A.C.Hearn and J.Fitch et al.[5,2] to minimize the difficulties associated in transporting the REDUCE algebraic system to various Lisp dialects. And currently the REDUCE on SLISP/IBM 360 & 370 runs on hundreads of sites over the world[6]. We became a REDUCE user through the famous HLISP/Fortran-REDUCE[3,8,9] since 1977; and in 1979 we had run SLISP/IBM-REDUCE(version:Oct-15-78); and since January '80 and up to now, we have been using SLISP/IBM-REDUCE(ver.:Apr-15-79). Between these last two releases, the basic interpreters written in IBM 360 & 370 assembly language are identical word by word, letter by letter.

We encountered some difficulties in the system -- some are dependent on our site, and some were essentially by bugs survived so long the history of the maintenance efforts of various individuals since J.McCarthy[7]. Studying the interpreter code and locating and fixing bugs, we found better ways to do the same jobs and made minor modifications in the interpreter and the system FRONT-END file -- but intact the REDUCE system file. Two functions are added, four deleted and some modified.

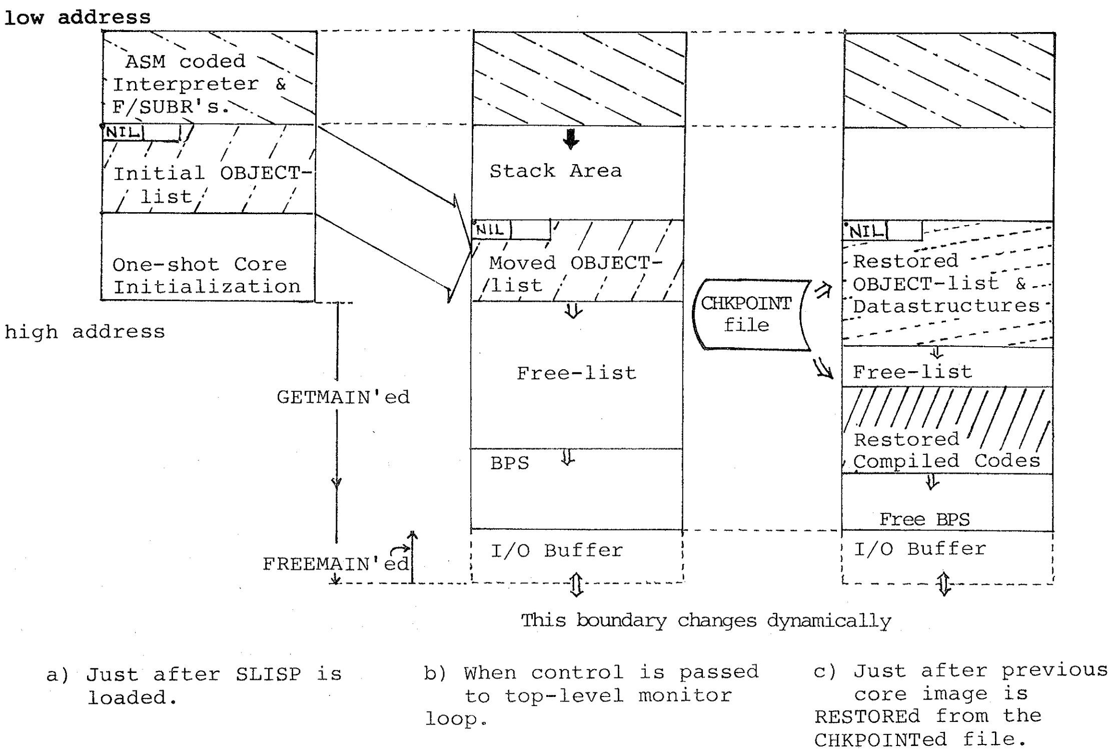
The organization of this paper is as follows: after a short overview of the original SLISP in section 1, we deal with bugs and how they are fixed in section 2; in section 3, with features made more comfortable; in section 4 speeding-ups are discussed, and in section 5, with the concluding remarks, we discuss the desirable features of the Lisp system when we are to use it as an indispensable programming vehicle in versatile applications as well as experimental or theoretical tool. This proposal is heavily influenced by our long practices in both Lisp, Snobol4[4], and Spitbol[1].

§ 1 -- Original SLISP/IBM, an Overview --

The SLISP is a rather conservative Lisp system. Its top level is EVAL, but EVAL and APPLY utilize hidden association-list, A-List ("ALIST" is the actual ASM label for the memory location for it). It has no Vector datatype but is simulated by a package in the FRONT-END file: MKVEC for creation, PUTV and GETV for value store and retrieval.

Its memory-maps at three initial stages are illustrated in Fig.1. Note that it is implicitly assumed that memory area acquired by the initial GETMAIN macro call is contagious and is in higher address space.

The Free-Cell Storage, FCS, is a collection of conventional dotted-pair cells, and even the Atoms are constructed by this basic cell. Fig.2 shows the general field assignment and examples for some Atom headers. The two tag fields are used as "what kind of cell the container is", instead of the "rich pointer" used in HLISP, for example, in which type-information of a pointed cell is stored in the pointing field, reducing the memory-access time in type-checking. Garbage-Collector, GC, uses as the mark-bit the MSB in the CDR-tag. Most of the cells have x'00' in these two tag fields.



low address
ASM coded Interpreter & F/SUBR's.
NIL
Initial OBJECT-list
One-shot Core Initialization
high address
GETMAIN'ed
FREEMAIN'ed
Stack Area
NIL
Moved OBJECT-/list
Free-list
BPS
I/O Buffer
CHKPOINT file
NIL
Restored OBJECT-list & Datastructures
Free-list
Restored Compiled Codes
Free BPS
I/O Buffer
This boundary changes dynamically
a) Just after SLISP is loaded.
b) When control is passed to top-level monitor loop.
c) Just after previous core image is RESTOREd from the CHKPOINTed file.

Fig. 1) The memory-Map of the Original SLISP/IBM at first few stages.

The Property-List, P-List, is a list composed of indicator-value pairs and the Flag-indicators. The Value of an Atom is put on this P-List under the name of APVAL or FLUID. The function definitions are put under the name of EXPR, FEXPR, SUBR, FSUBR or MACRO. In the evaluation of an expression, the value or function definition of an Atom is searched by traversing the P-List for the indicators in the above order; if the earch was unfruitful, then the current A-List is tried.
The datatype String is represented as a literal Atom, flagged by the indicator STRING as the first element in the P-List, and has a value cell pointing to itself under the indicator of APVAL in order to evaluate the STRING itself.

§ 2 -- The Malfunctions and How They are Fixed --

There were the following three bugs:

a) The relocation code in the function RESTORE changed the value of OBLIST when the initial GETMAIN macro call returned noncontagious memory (this happened when SLISP was loaded by CALL command in a command procedure of TSO).
b) ERRORSET did not handle the A-List properly. If it was evoked in EXPR or FEXPR with Fluid variables, the updated A-List was not stored into the word "ALIST".
c) EQUAL bombed if non-EQ function-pointers or file DCB's were passed.

The bug a) occurred because of the following two reasons(Fig.3):

1°) Initial OBLIST made by "ECHO" macro had erroneous CDR-tag in the value cell of APVAL in its P-List(Fig.2-g). AWD=x'40' is reserved for SUBR's or FSUBR's, ASM-coded or later compiled.
2°) After copying old memory image, the relocation phase starts scanning the entire active FCS to adjust the pointers in CAR/CDR-address part to either δMAIN or δNIL. If the CDR-tag AWD was encountered, it checked the CAR-pointer only against OLDBPBAS to determine whether it was compiled or ASM-coded.

This was corrected by enhancing the "ECHO" macro which, linking the Atoms, deposits initial Atom headers and P-Names and P-List indicator-value pairs such as APVAL, FLUID, SUBR or FSUBR. It handles properly the blank-embedded P-Names by

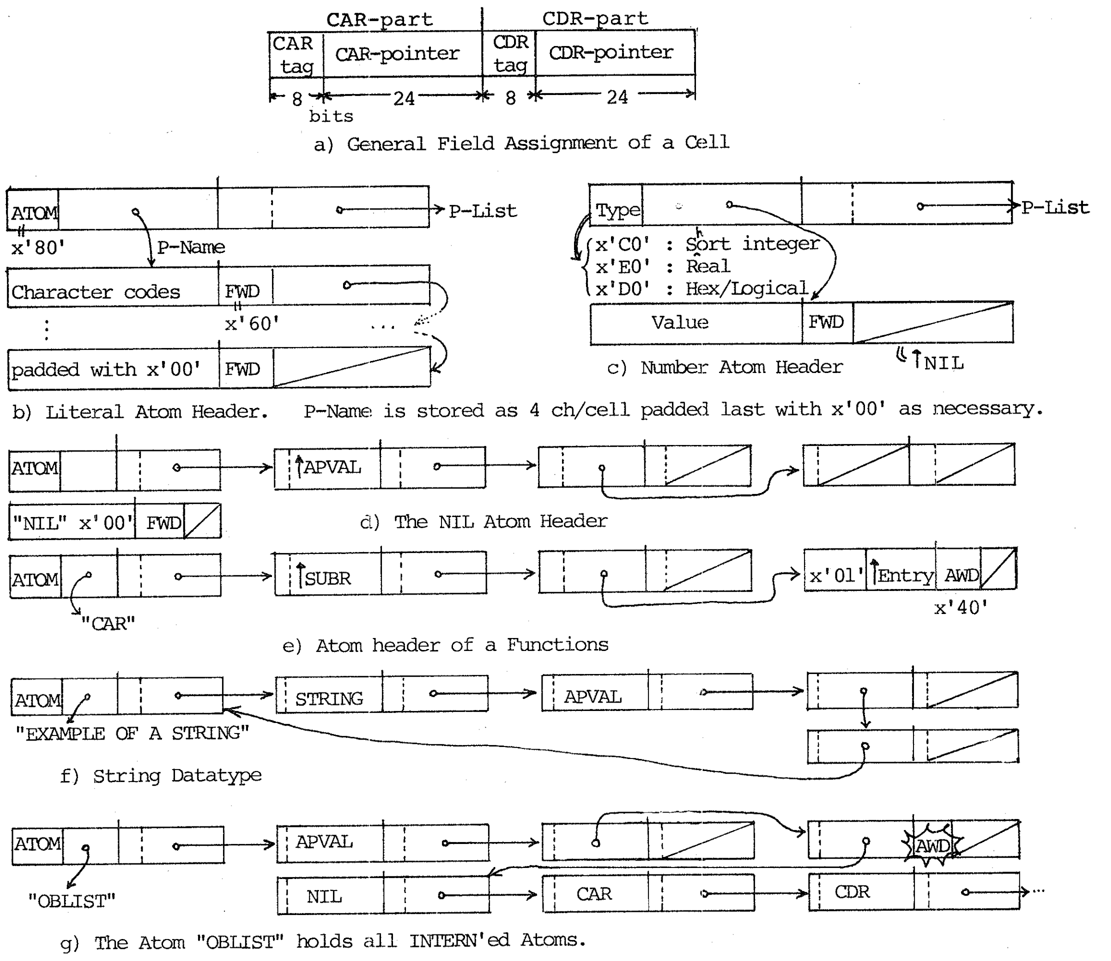


Fig. 2) Examples of Atom Header Structure.

extending the calling syntax. The comparison in the relocation of RESTORE is changed to that against OLDNIL instead of OLDBPBAS, because the relocation amount is the same for both FCS and BPS.

Bug b) was a rather unexpected one. REDUCE system frequently relies upon this function but with deliberate protection of FLUID declarations.

Bug c) was found by running the SLISP verifier, a program in the REDUCE system MT of an early release, which we obtained through Mr. N.Inada of RIKEN. The function EQUAL now checks that both arguments are valid FCS pointers before it renders recursive call to itself.

§ 3 -- Features Made More Comfortable --

The following 10 features are added or modified to make the SLISP more comforable to TSO or Batch users:

3-1) In our TSO/MVS environment, a session is abruptly cancelled by the monitor if it has consumed only 3 minutes of CPU-time. The original SLISP stopped the Task Timer during the GC; SLISP user was unable to know exactly how much of time was left for him/her to save the current state of computation by CHKPOINT. Now the Timer ticks continuously once it is started just before top-level monitor loop gets control, and user can get the consumed time from that point by '(TIME)'. And he/she can take an appropriate action for the later restart. As for the subsidial informations, GC's CPU-time and GC-count are maintained.

3-2) When the error A2, A6, A8 or A9 occurs, the identifier's P-Name for the unbound variable, undefined function or missing GO-label in the PROG is printed to

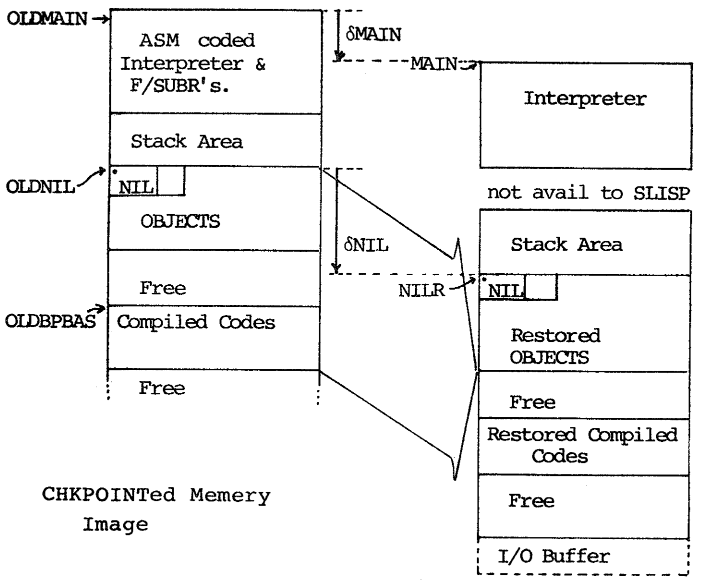


Fig. 3) Memory Map in RESTORE'ing, when noncontagious area is allocated by the initial GETMAIN.

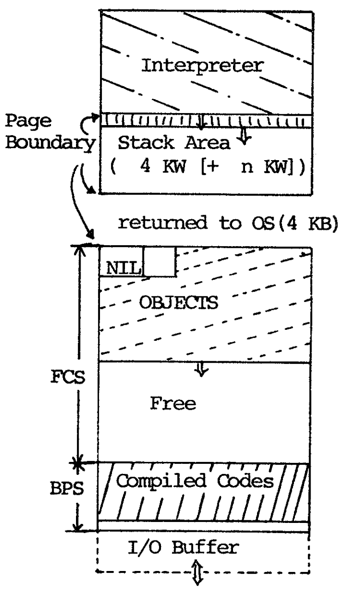


Fig. 4) New Memory Map at Execution Time.

facilitate programmers to quickly locate the errors.

3-3) When the absolute system bomb occurs, SLISP dumps for the last resort the old Program Status Words(Program Counter, Interrupt code, etc.), all 16 general registers' contents and some address constants proper to SLISP. In the original, PSW's PC was always the same, pointing within the TRAP handler. Now this PC points the next instruction to be executed after the one which actually caused the trouble.

3-4) Two new functions are added to facilitate system maintenance efforts:

!%CELL(LOC:{integer(short or hex), or alike}): dotted-pair

Type: EVAL, SPREAD

If LOC ⩾ 0, it returns a dotted-pair whose CAR/CDR is the hex-num representation of the corresponding contents of the cell pointed by LOC. If 0 > LOC ⩾ -16, it returns the contents of the general register |LOC| -1 in a hex-num as the single element in a list. No argument check is made, so one can obtain, for example, Atom header of any x by

(!%CELL (LIST (LIST (QUOTE x)))).

!%ZAP(LOC:integer, VAL:integer, MSK:{integer in the range of 0-15}):NIL

Type: EVAL, SPREAD

This rewrites any location pointed by LOC by the value and length specified by VAL and MSK for the duration of the current SLISP run. This is a Lisp version of 370-machine instruction, Store Characters under the Mask:

STCM R_x,MSK,LOC.

The 4 bytes in VAL correspond to the 4 bits in MSK from left to right respectively; modification length is the number of 1's in the MSK; bytes selected from VAL are packed and stored to the memory starting at LOC contagiously(some cares for the 360's).

3-5) And two BPS save/restore functions, BPSCHKPT/BOSRSTR, are deleted because of their poor ability to cooperate well with FCS data-structures as documented in the Manual[2,p.31]. Also garbage code bodies for RBLK/WBLK are deleted. Their corresponding Atom headers are made comment lines in the original and are inaccessible to users.

3-6) Messages are made shorter and richer of informations. Also redundant space-lines/page-eject are supressed to save the forest resources. If functions are TRACE'd, a global trace-level counter is maintained with modulo 32, and encoded into one character out of "012...9ABC...STU@" in the trace print out(as this is a very simple counter it betrays if GO is traced or evaluation fails within an ERRORSET environment).

3-7) FLAG code is changed to examine that the Atom has not been FLAG'ed yet by that indicator before calling CONS to affix the indicator on the top of its P-List. Flag duplications occurred during REDUCE assembly and its algebraic mode. They are LOSE(to inhibit redefinition of functions), GLOBAL, and USED!*(to maintain proper database in algebraic mode). REMFLAG is shortened to remove only the first indicator found in the P-List.

3-8) Although it is a slight deviation from the Standard, some predicates are changed to return non-NIL values instead of the trite T.

FLAGP: a sublist of the P-List of the first argument whose first element is the second argument.
MEMBER: a sublist of the second argument whose top element is EQUAL to the first.

CODEP, IDP, STRINGP, LOGP, BIGP, FLOATP, DIGIT, LITER, DELCP and SEPRP :	their argument is returned. (The last two are moved down to ASM code, which are used in REDUCE token reader and defined there).

3-9) After the normal REDUCE assembly, the compiler and its associated database are removed by '(REMCMP)', but some garbages were left in the FCS. Writing a removal package, CLEANER, was easy by the aids of the new function, !%CELL. In short, it scans down the OBLIST and gathers all Atoms without P-List which are neither pointed from FCS except OBLIST link nor from BPS. It also checks special P-Lists of QUOTES and FLUIDS which hold QUOTE'd expressions or value-cells of variables declared FLUID at compilation time.
By this CLEANER package, some 2,500 cells are released.

3-10) We find a rather kludge way to recover the control to the Lisp top-level monitor loop just after pressing the BREAK-/Attension-interrupt-key when the computation falls in some undesirable situations. Under the normal condition, these interruptions cause the direct exit to TSO monitor, and the user program is purged from the memory.
To do this pseudo-BREAK-recovery, one must assemble and link-edit with the option "TEST". Immediately after the BREAK-key is pressed and TSO indicates "READY", just enter "TEST" without operand. One can get the control indirectly through the TEST processor passed to our Lisp after examining and restoring few registers. In our experiences, R_{12},R_3,R_4, R_5 and R_7 are candidates to be restored. The normal values for them are easily obtained by an intensional floating-zero-division, which is TRAP'ed and subsequently registers' contents are dumped.

§ 4 -- Speed-ups --

As mentioned earlier, CPU-time is very tight in our environment and it is absolutely necessary to have a very fast Lisp system. We made the following 5 major changes:

4-1) Stack Operation: In the original, only in PUSH was the stack-overflow checked by a time consuming BXH* instruction, which takes 3~4 CPU cyle time for 3033†. Now This check is rendered to the hardware by aligning the start of the FCS at the page boundary and releasing just one page to OS between the stack area and the Atom NIL. PUSH macro occupies 6 bytes instead of 8 -- thus both time and space are saved:

```
ST     &R,0(,PDS)    ───→   ST     &R,0(,PDS)
BXH    PDS,K4,ERG2   ───→   AR     PDS,K4
```

(PDS=R_7 points next usable stack word, K4=R_4 holds the constant 4 and NILR=R_5 holds the address of the Atom NIL).
If the stack-overflow occurs, control is passed to TRAP handler with the indication of "Protection Exception", i.e. page-key is different and read/write protected. The consecutive stack operations are also time or space optimized by using STM/LM§ instructions.

----- ----- -----
* Branch on Index High, an instruction to form loop. The fall-through case is decoded later, which is the normal in PUSH.
† not an official data from IBM; measured by a timer program. 1 Cyc.time = 57 nsec.
§ Store Multiple/Load Multiple. Multiple consecutive registers are accessed.

In the EVAL code, PUSH of the return point(R_2, the linkage register) is postponed until just before some recursion may begin; it saves a little CPU-time for the Atomic arguments.

As a consequence of this, the stack area cannot be changed by SETSIZE or CONDENSE; they simply ignore the first argument. It can be only set by EXEC S-parameter once; its default size is 4 K words and expandable by 1 K words step.

4-2) Compiler-Interface: The compiler is written in SLISP, and there are some tricks in the ASM-code. When a function call occurs from a compiled code, double branches were made in the original; first BAL* into a fixed location at the beginning of base register R_{12}, which is another Branch to the desired function(Branch Table)are. Now this Table is a collection of model instructions to be deposited directly into the BPS. This change in the FRONT-END file is made easily using !%CELL. Also at the entry prologue, PUSH'ing some registers became shorter codes (about 200 words are saved in the total including REDUCE system).

4-3) EVLIS-SPREAD: These internal functions are never called in the normal EVAL. New @EVLIS and @SPREAD are coded. The former pushes evaluated arguments onto the stack and the latter spreads them to the appropriate register(s) or pseudo-register (s). The register R_{15} is used as the frame pointer to communicate between these two. The CONS'ings are avoided and again LM/MVC† are used instread of the list-traversal, resulting less GC evocations.

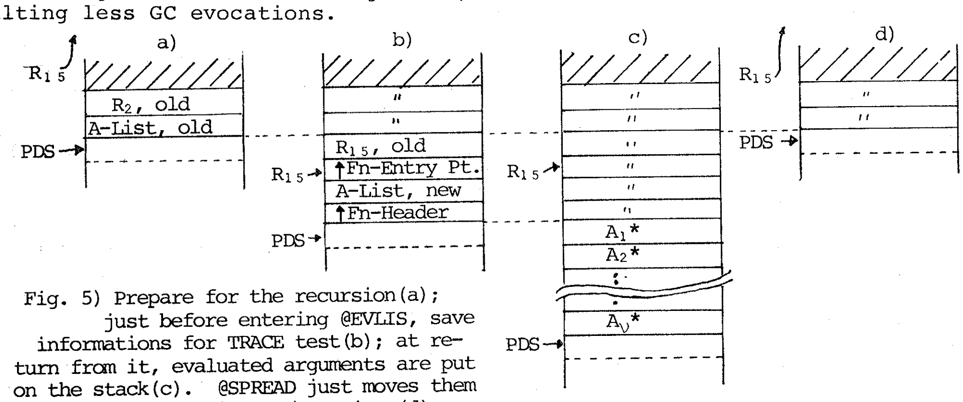


Fig. 5) Prepare for the recursion(a); just before entering @EVLIS, save informations for TRACE test(b); at return from it, evaluated arguments are put on the stack(c). @SPREAD just moves them to appropriate registers/memories (d).

4-4) Order in Function Type Search: Obviously the most frequently called functions are of type F/SUBR, so we changed the order to search them first:

(QUOTE → COND →) EXPR →FEXPR →SUBR →FSUBR →MACRO →(FLUID → in A-List)

⟹ ···→ SUBR →FSUBR →EXPR →FEXPR →··· .

Along with this change, new macros, $GET and $SASSOC, are introduced to enable direct jump to the next alternative in case of search failure, instead of BAL'ing to the internal subroutines and then comparing the result to NILR to determin it.

4-5) COND antecedents: It is very common to use long COND expressions in inner-loops. Predicates should be fast to fail if they are doomed so after all. Some predicates are changed to this end.

§ 5 -- Concluding Remarks --

The substantial part of this work begun in January, 1980 and was completed in September. In this process it was very invaluable to have Spitbol running. SLISP is a rather large system; about 8,500 lines in assembly code and 3,600 lines in SLISP for the FRONT-END including the portable compiler. For example, imagine the following tedious and error-prone task: While scanning the assembly source, marking and counting and classifying all the system or user macro calls by read-and-blue pencil! Spitbol's hash TABLE(collisions are resolved by linking all the items with the same hash index) is especially useful. It can contain anything and extends its capacity as required. TABLE can be converted to ARRAY, and vice versa, with a minor but reasonable restriction.

----- ----- -----

* Branch and Link, for subprogram linkages. It saves current PC value to a specified register and then branches.

† Move Characters, up to 256 bytes are moved from memory to memory.

Spitbol has the packed STRING datatype, which may be internally shared among them by the virture of start- and offset-fields in its SPECIFIER, and has a good repertoir of STRING PATTERN MATCH operations.

The Vector is an indispensable datatype in the modern Lisp system. If it is built-in, the open-chaining Hash Table can be easily constructed. Strings can be packed if it provides bit/byte-Vector of any size. We recently felt the necessity of bit-Vectors in writing an application program concerning the PLA optimization, where subset of a moderate sized universe is economically encoded into this bit-Vector.

-- ACKNOWLEDGEMENTS --

It is a pleasure to acknowledge the encouragement by Dr.Heitaro Nakajima, Director of Audio Technology Center, in the process of this work. We had many care-takings of MT handling from Dr.Y.Kanada and Mr.N.Inada. We also express our hearty thanks to them.

-- REFERENCES --

[1] Dewar,R.B.K.:"SPITBOL Version 2.0", Document S4D23, Illinois Institute of Tech., Chicago, 1971.
[2] Fitch,J.:"Manual for Standard Lisp on IBM 360 and 370", Technical Report TR-6, University of Utah, Aug. 1978.
[3] Goto,E.:"Monocopy and Associative Algorithms in an Extended Lisp", Technical Report 74-03, Inform. Sci. Labs. University of Tokyo, 1974.
[4] Griswold,R.E. et al.:"The SNOBOL4 Programming Language", 2nd Ed., Prentice-Hall, 1971.
[5] Hearn,A.C. et al.:"Standard Lisp Report", UPC-60, Univ. of Utah, jan. 1978, revised 1979.
[6] Hearn,A.C.(Ed.):"REDUCE Newsletter", Nos. 3,5 and 7, 1978-79.
[7] Hearn,A.C.: a private discussion with him at IFIP '80, Tokyo. The Assembly comments indicate the following personnels:
J.G.Kent, J.F.Bolce, R.I.Berns, K.R.Kay and J.Fitch.
[8] Kanada,Y.:"Implementation HLISP and Algebraic Manipulation Language REDUCE-2", Technical Report 75-01, Inform. Sci. Labs. Univ. of Tokyo, 1975.
[9] Terashima,M.:"Algorithms Used in an Implementation of HLISP", Technical Report 75-03, ibid., Jan. 1975.

-- APPENDIX A --

Notes: 1°) The time unit is milli-seconds, excluding the time of GC.
2°) I:Interpreted, C:Compiled.
3°) The first 3 bentch-mark problems are taken from that of 2nd Lisp Contest of IPSJ WGSYM [Take-uchi,I.:"The Result of Lisp Contest", Aug. 21, '78].

	Improved SLISP		Original SLISP		ratio(Imp/Orig)	
	I	C	I	C	I	C
TARAI[integer version] n=3	62	10	95	11	1.53	1.10
n=4	1,173	186	1,779	206	1.52	1.11
n=5	31,862	5,089	48,140	5,613	1.51	1.10
TARAI[list version] n=3	165	5	265	6	1.61	1.20
n=4	3,125	108	5,007	127	1.60	1.18
n=5	---	2,967	---	3,448	---	1.16
SORT[integer] n=20	80	5	145	6	1.83	1.20
n=40	286	20	535	24	1.87	1.20
n=60	461	32	848	39	1.84	1.22
n=80	690	49	1,278	60	1.86	1.22
n=100	888	64	1,643	78	1.85	1.22
REV[see below] n=5	53	3	85	4	1.60	1.33
n=6	215	12	344	16	1.60	1.33
n=7	854	51	1,375	65	1.61	1.27
n=8	3,439	206	5,521	262	1.61	1.27
n=9	13,808	825	22,119	1,047	1.60	1.27
n=10	55,221	3,306	88,430	4,186	1.60	1.27

*) The REDUCE test program takes 7.1 seconds (9.1 previously), excluding the GC which occurred 4 times and used 0.5 seconds under TSO with logon size = 1 M bytes.

The following problem made by Mr.A.Rich takes time propotional to 4^n, n beeing the length of the argument list of REV.

```
(DE REV (X)
    (COND ((NULL X) NIL)
    (T (APP (REV (CDR X))
            (LIST (CAR X))) )))
```

```
(DE APP (X Y)
    (COND ((NULL X) Y)
       (T (APP (REV (CDR (REV X)))
               (CONS (CAR (REV X)) Y))) ))
```

-- APPENDIX B --

Some examples of messages. First the initial one and top-level monitor's. The right most **is** made if functions are traced.

```
-CONTIGUOUS-
STANDARD LISP  INITIAL CORE ALLOCATION:  FREE CELLS =   339616, BPS =   349904, PDS =     4096.
```

here a page-eject occurs

```
ARGUMENT FOR EVAL ...
  (RESTORE (QUOTE REDU))

TIME      124MS,  VALUE IS ...
  NIL

ARGUMENT FOR EVAL ...
  !*COMP

TIME        0MS,  VALUE IS ...
  T

ARGUMENT FOR EVAL ...
  (SETQ !*COMP NIL)

TIME        0MS,  VALUE IS ...
  NIL

ARGUMENT FOR EVAL ...
  (BEGIN)
```

```
  REDUCE 2 (APR-15-79) ...
   ON INT;

  LISP;
  NIL



   CLEAN();
  NIL

   CLEAN();
  NIL

  END;ENTERING LISP...

TIME     269MS,  VALUE IS ...
  NIL
```

```
### ARGUMENTS OF NULL
    ("APR-15-79")

### VALUE OF NULL
    NIL

### ARGUMENTS OF VERBOS
    (NIL)

### VALUE OF VERBOS
    NIL
```

And the more pleasant ones made by the revised. The top-right is a trace example, and the bottom-right is the final one.

```
-CONTIGUOUS-
STANDARD LISP  INITIAL CORE ALLOCATION :

  FCS =  338933 CELLS

  BPS =  349202 WORDS

STACK =    4096 WORDS.

==<< MINOR MODIFICATIONS BY K.NAMBA, SEP. 09, 1980. >>==

<= EVAL ARGUMENT : (RESTORE (QUOTE RED))
=> EVAL TIME :       89 MS, ANS = NIL

<= EVAL ARGUMENT : !*COMP
=> EVAL TIME :        0 MS, ANS = NIL

<= EVAL ARGUMENT : (SETQ !*COMP NIL)
=> EVAL TIME :        0 MS, ANS = NIL

<= EVAL ARGUMENT : (BEGIN)



     REDUCE 2 (APR-15-79) ...
     ON INT;

     LISP;
     NIL


     CLEAN();
     NIL

     CLEAN();
     NIL

     END;ENTERING LISP...
 => EVAL TIME :     1441 MS, ANS = NIL
 GC.=<        2,    1185 MS>
```

```
<= EVAL ARGUMENT : (SETQ DATE!* "APR-15
<0+ SETQ (DATE!* "APR-15-79")
-0> SETQ = "APR-15-79"
=> EVAL TIME :          0 MS, ANS = "APR-1

<= EVAL ARGUMENT : (BEGIN)
<0+ BEGIN NIL
<1+ PROG (NIL (SETQ !*INT NIL) (SETQ !*
    ETQ OFL!* (SETQ OPL!* NIL)))) (COND
    !*) 500) (COND ((NULL !*COMP) (BPSM
     ...") (TERPRI) (SETQ DATE!* NIL) A
    ...") (TERPRI))
<2+ SETQ (!*INT NIL)
-2> SETQ = NIL
<2+ SETQ (!*ECHO IECHO!*)
-2> SETQ = T
<2+ SETQ (!*EXTRAECHO T)
-2> SETQ = T
<2+ SETQ (TMODE!* NIL)
-2> SETQ = NIL
<2+ SETQ (IFL!* (SETQ IPL!* (SETQ OFL!*
<3+ SETQ (IPL!* (SETQ OFL!* (SETQ OPL!*
<4+ SETQ (OFL!* (SETQ OPL!* NIL))
<5+ SETQ (OPL!* NIL)
-5> SETQ = NIL
-4> SETQ = NIL
-3> SETQ = NIL
-2> SETQ = NIL
<2+ NULL ("APR-15-79")
-2> NULL = NIL
```

```
 => EVAL TIME :        0 MS, ANS = NIL


 ### END OF DATA

 ===<< LISP RUN STATISTICS >>===

  CPU TIME :      1573 MS

   GC TIME :      1185 MS; #GC =        2.
```