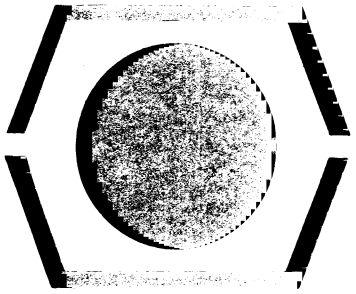

DECUS

PROGRAM LIBRARY

DECUS NO.	8-102a
TITLE	A LISP INTERPRETER FOR THE PDP-8
AUTHOR	G. van der Mey and W. L. van der Poel
COMPANY	Technical University of Delft The Netherlands
DATE	May, 1968
SOURCE LANGUAGE	

A LISP INTERPRETER FOR THE PDP-8

DECUS Program Library Write-up DECUS No. 8-102a

Summary

The present document gives a description of a system for the programming language LISP on the PDP-8. The system is designed to operate on a basic PDP-8 with 4096 words of core storage and an ASR-33 Teletype. In addition, the system will input on a high speed reader. More than half of the storage is used as list space. The system is particularly suitable for conversational use and teaching. There are very few restrictions to the language, apart from the total storage space.

Introduction

The reader is assumed to be familiar with LISP, a programming language for list manipulation. Detailed descriptions can be found in:

1. "LISP 1.5 Programmer's Manual," J. McCarthy et al, M.I.T. Press, 1965.

2. "The Programming Language LISP - Its Operation and Application," E. C. Berkeley and D. G. Bobrow, eds., Information International, Inc.

3. "LISP 1.5 Primer," Clark Weissman, Dickinson Publishing Company.

This LISP system is an interpreter-based simplification of LISP 1.5 as implemented on the IBM 7090. The severe restrictions on size in the smallest PDP-8 have caused the omission of many advanced functions, which can be defined in more basic functions. Nevertheless, the system has been designed in a very general way, e.g. no restrictions on the length of names and on the size of the push-down list, all functions C....R with as many as 11 A's or D's in between are built in, decimal input/output (can be switched to octal independently for input and output). The storage is used both for push-down space and list space. Only if the total capacity of both together is too big, the system stops. The push-down is organized itself as a list in the normal list space. A very full assortment of error stops is included. Just as in 7090 LISP, the programmer need not quote his arguments on the outside level; the system has the function EVALQUOTE. Input tapes can be in ASCII code or in CCITT2 Teleprinter five hole code. They can be mixed with the only precaution that both kind of tapes start with blank (ASCII and CCITT2) or with leader-trailer code (ASCII). Control over the printing and the choice of low or high speed reader can be exercised by a mode number instead of a function in EVALQUOTE. (See System Operation)

Inside the system a LISP cell occupies two locations in core. The first location is used for the CDR part and the next location is used for the

CAR part. As all cells start in an even location, pointers are always even. Therefore, the rightmost bit has been used for other purposes, e.g. the rightmost bit in the CAR part is the atom mark and the rightmost bit in the CDR part is used in the garbage collector. It is normally zero. As the CDR operation is more frequent than the CAR operation, this arrangement allows the CDR to be simply performed by an indirect access of a location.

Access to machine code programming is through a single function EXPR of three arguments. Several useful actions can be accessed via this function: inspection of a machine word, changing a machine word (also in the system itself), logical and, logical not.

The system is located in addresses 2 to 3725$_8$; all the rest can be used as list space. The standard binary tape will not destroy a minimum binary loader in 7700-7777. By changing a single constant, the list space can be lengthened or shortened. For example, machine code programs can be placed above the list space. These changes can be made expressed in the language LISP itself.

The Available Functions and Objects

The following list of objects and functions are built-in for PDP-8 LISP. When they are identical to the same function in 7090 LISP, they are described very summarily, but several functions deviate in some respects and are more fully described. In addition, a LISP definition of the function is given where possible. This LISP description is not always accurate, e.g. no error testing will be shown, nor will machine code function be expressed adequately.

Built-in function names are recognized by the value of their pointer. They do not have indicators SUBR or FSUBR on their property lists. In fact there are five kinds of built-in functions and objects:

1. Special functions having an arbitrary number of arguments or a very special treatment of their arguments. They are LAMBDA, NIL, FUNARG, T, APVAL, COND, FEXPR, FUNCTI, GO, LIST, MINUS, PLUS, PROG, QUOTE, RETURN, SETQ, and STOP.

2. Functions with no arguments: GENSYM, READ, TERPRI.

3. Functions with one argument: ATOM, CAR, CDR, DEFINE, NULL, NUMBER, PRINT.

4. Functions with two arguments: ASSOC, CONS, DEFLIS, EQ, EQUAL, EVAL, GET, LESSP, RPLACA, RPLACD, SET.

5. Functions with three arguments: APPLY, EXPR.

APPLY - This is a function of three arguments. It applies the function mentioned in the first argument to the list of arguments mentioned as the second argument. The association list to be used is given in the third argument (often NIL).

```
DEFINE((
   (APPLY(LAMBDA(FN ARGS ALIST)(PROG(K)
      ((NULL FN)(RETURN NIL))
      ((FN-pointer < certain address)(do machine code function))
      ((ATOM FN)(GO A))
     B((EQ(CAR FN)FUNARG)(RETURN(APPLY(CADR FN)ARGS(CADDR FN))))
      ((EQ(CAR FN)LAMBDA)
       (RETURN(EVAL(CADDR FN)(PAIRLIS(CADR FN)ARGS ALIST))) )
      (RETURN(APPLY(EVAL FN ALIST)ARGS ALIST))
     A(SETQ K(ASSOC FN ALIST))
      (COND(K(RETURN(APPLY(CDR K)ARGS ALIST))))
      (SETQ K(GET FN EXPR))
      (COND(K(RETURN(APPLY K ARGS ALIST))))
      (GO B) )))
   (PAIRLIS(LAMBDA(X Y A)(COND((NULL X)A)
      (T(CONS(CONS(CAR X)(CAR Y))(PAIRLIS(CDR X)(CDR Y)A)))) ))) ))
```

Remarks: The given definitions will take advantage of the features of PDP LISP, e.g. COND has been omitted where possible (see under COND) and standard object names need not be quoted. PAIRLIS has been defined together with APPLY but is not a standard function of the system.

APVAL - This is used as an indicator for objects having a value on the property list. But in PDP-8 LISP is can also be used as a function for sensing blank on tape. It returns T when a symbol blank is typed in or read in by the tape reader; the tape steps to the next symbol. When there is no blank under the tape reader, it returns with NIL and does not move the tape. (For use see also T.)

ASSOC - Looks up the first argument on the association list given in the second argument. Returns with the pair found. If not found then NIL.

```
DEFINE(((ASSOC(LAMBDA(X A)(COND
   ((NULL A)NIL)
   ((EQ(CAAR A)X)(CAR A))
   (T(ASSOC X(CDR A))) )))))
```

ATOM - Function of one argument. Returns with T if argument is atomic, otherwise with NIL.

CAR - Returns with first of its argument, which must be a list. If argument is atomic, then an error print follows. If a CAR function is needed which can look into the atom it can be simulated with

```
DEFINE(((CAR(LAMBDA(X)(CDR(EXPR 3172(PLUS(CONS NIL X)1)-1)) ) )))
```

CDR - Returns with the rest of its argument when the first is removed. Application to an atom is allowed to look into property lists; however, this is not advisable for built-in functions as they have no property list except OBLIST and NIL. Instead they have a pointer into a machine code program which is recognized because it is below a certain limit.

```
CDR(NIL) = NIL
```

C...R - Any function with as many as eleven A's and D's may be given. For example, (CAADDAR X) = (CAR(CAR(CDR(CDR(CAR X))))). On first appearance of such a function, it will be placed on the OBLIST with no properties. Nevertheless, it will be recognized as a function unless it has been redefined as something else.

COND - Pseudo-function having an indeterminate number of arguments, each being a pair. If the first of a pair does not have the value NIL, the value of the complete COND form has the value of the second of the pair. Otherwise the next pair is tried. An error print will follow if COND "drops out of the bottom" except in PROG.

CONS - Function of two arguments. Constructs a dotted pair from its arguments.

DEFINE - Defines a list of functions by attaching a form to the name on the property list with the indicator EXPR. A function, also a built-in function, may be redefined. The original significance is then inaccessible unless the name is removed from the OBLIST. With the help of this feature, a standard function can be redefined to print relevant information about its arguments and values, or can count its number of calls. It makes TRACE superfluous.

```
DEFINE(((DEFINE(LAMBDA(X)(DEFLIS X EXPR)))))
```

DEFLIS - Defines a list of functions by attaching a form to the name on the property list with the indicator given in the second argument. All built-in functions have names of no more than six letters, mainly for economy reasons. Its effect is exactly the same as DEFLIST in the 7090.

```
DEFINE((
   (DEFLIS(LAMBDA(L PRO)(MAPLIST L(FUNCTI(LAMBDA(J)
      (DEF1(CAAR J)(CADAR J)) )))))
   (DEF1(LAMBDA(OB L)(PROG NIL
      (RPLACA(PROP OB PRO(FUNCTI(LAMBDA NIL
         (CDDR(RPLACD OB(CONS PRO(CONS NIL(CDR OB))))) )))L)
      (RETURN OB) )))
   (PROP(LAMBDA(X Y FN)(PROG NIL
   A  ((NULL X)(RETURN(FN)))
      ((EQ(CAR X)Y)(RETURN(CDR X)))
      (SETQ X(CDR X))
      (GO A) )))
   (MAPLIST(LAMBDA(X FN)(COND((NULL X)NIL)
      (T(CONS(FN X)(MAPLIST(CDR X)FN))) ))) ))
```

EQ - Returns with T when its two arguments are identical pointers, or when they point to numbers which have the same value. Otherwise, its value is NIL.

EQUAL - Returns with T if its arguments are equal; NIL if they are unequal.

```
DEFINE(((EQUAL(LAMBDA(X Y)(COND
   ((ATOM X)(COND((ATOM Y)(EQ X Y))
      (T NIL) ))
   ((ATOM Y)NIL)
   ((EQUAL(CAR X)(CAR Y))(EQUAL(CDR X)(CDR Y)))
   (T NIL) )))))
```

EVAL - Returns with the evaluated first argument, with the association list given in the second argument.

```
DEFINE((
   (EVAL(LAMBDA(FORM ALIST)(PROG(K L)
      ((NULL FORM)(RETURN NIL))
      ((NUMBER FORM)(RETURN FORM))
      ((ATOM FORM)(GO A))
   B  (SETQ K(CAR FORM))
      ((EQ K QUOTE)(RETURN(CADR FORM)))
      ((EQ K FUNCTI)(RETURN(LIST FUNARG(CADR FORM)ALIST)))
      ((EQ K COND)(RETURN(EVCON(CDR FORM)ALIST)))
      ((EQ K PROG)(RETURN(EVPROG(CDR FORM)ALIST)))
      ((ATOM K)(GO C))
   D  (RETURN(APPLY K(EVLIS(CDR FORM)ALIST)ALIST))
   A  (SETQ K(ASSOC FORM ALIST))
      (COND(K(RETURN(CDR K)))
      (RETURN(GET FORM APVAL))
   C  (SETQ L(GET K EXPR))
      (COND(L(RETURN(APPLY L(EVLIS(CDR FORM)ALIST)ALIST))))
      (SETQ L(GET K FEXPR))
      (COND(L(RETURN(APPLY L(LIST(CDR FORM)ALIST)ALIST))))
      (RETURN(EVAL(CONS(CDR(ASSOC K ALIST))(CDR FORM))ALIST)) )))
   (EVCON(LAMBDA(C A)(COND
      ((EVAL(CAAR C)A)(EVAL(CADAR C)A))
      (T(EVCON(CDR C)A)) )))
(EVLIS(LAMBDA(L A)(COND
      ((NULL L)NIL)
      (T(CONS(EVAL(CAR L)A)(EVLIS(CDR L)A))) ))) ))
```

Remark: The functions EVCON and EVLIS are not available as standard functions under these names.

EXPR - Used as indicator on property lists for functions which have their arguments evaluated. In the PDP-8 LISP, EXPR also serves as a function of three arguments. The arguments are supposed to evaluate to numerical values (unless not used). It jumps to the machine address indicated in the first argument with the numerical value of the second argument in the accumulator. The pointer to the second argument can be found in 0037, and the pointer to the third argument in 0041. The machine code program can return to the LISP system with the instruction 5171 (=G EV-5) in case that no value need be returned. If a numerical value must be returned, this can be done by going to 3175 with the value in the AC. Back in LISP one then has a pointer to that value as value. In the system, several machine code functions are readily provided for the user. The most practical examples will be given below. Octal numbers will be shown normally, decimal numbers will be shown underlined.

EXPR(3172 X -1) or EXPR(1658 X -1) gives the contents of machine address X. When -1 is replaced by another number, this serves as a mask.

EXPR(3202 X Y) or EXPR(1666 X Y) will store in location X the number Y. This function can be used to change the system and forms the basic ingredient for a user-written LAP compiler. Applications:

EXPR(1666 1373 3584) switches the system to octal reading.

EXPR(3202 2535 1037) switches the system to decimal reading.

EXPR(3202 2034 1750) EXPR(3202 2035 144) EXPR(3202 2036 12) switches the system to decimal printing.

EXPR(3202 2034 1000) EXPR(3202 2035 100) EXPR(3202 2036 10) switches the system to octal printing.

EXPR(3202 2046 7061) switches the system to printing with sign.

EXPR(3202 2046 7000) switches the system to printing without sign. (-1 will now appear as 7777 in octal or 4095 in decimal.)

EXPR(3174 X Y) will return with the logical and operation on X and Y.

EXPR(3170 X Y) will return with switch register + X masked with Y. The logical inverse can simply be made with (MINUS -1 X).

EXPR(1306 addend mask) will return with (single character + addend) and mask from the tape reader in the code as it stands on tape.

EXPR(2160 X NIL) will print a single character whose value in internal representation is X. This can be used for outputting characters which would otherwise be regarded as syntactic symbols as space, (, or). Care must be taken, however, for the line count as e.g. carr. ret. now also counts as a printable character. The internal code is: 0 = space, 2 = space, 3 = line feed, 6 = carriage return, all other characters in the ASCII table from code 241 to 335 are found by subtracting 236 octally, e.g. 25 = digit 3, 43 = A, etc.

Sometimes it can be useful to know numerically the location of objects. There is no EXPR for this but this can be done with (PLUS(CONS NIL X)) as PLUS does not check whether the argument really is a number. Conversely (CDR number) will convert a number into a pointer.

FEXPR - Indicator on property lists to indicate special forms whose arguments are not evaluated.

FUNARG - Atom used on the association list to be able to recover previous association lists. Its operation is automatically introduced by the system whenever FUNCTI is used. The user almost never sees the atom FUNARG printed. (See APPLY and EVAL.)

FUNCTI - Same as FUNCTION in 7090 LISP. Used to quote literal functions presented as functional arguments.

GENSYM - Function with no arguments. It returns with a uniquely created new atom, which will not be attached to the OBLIST. If appear in print as: GGGG, GGHG, GGIG,..., GGVG, GGGH,GGHH,..., GGVV,...,GHGG, etc.

GET - Function of two arguments. Looks on the property list of the object given as the first argument to find the indicator given in the second argument. When the second argument is EXPR, FEXPR or APVAL, it need not be quoted as built-in names in PDP-8 LISP. A GET applied to a built-in function will always return with NIL.

```
DEFINE(((GET(LAMBDA(OBJ IND)(PROG(K)
      (SETQ K(CDR OBJ))
   A  ((NULL K)(RETURN NIL))
      ((EQ(CAR K)IND)(RETURN(CADR X)))
      (SETQ K(CDR K))
      (GO A) )))))
```

GO - Pseudo-function of a single argument. Goes to that argument when used as a label within PROG. Otherwise, an error print follows.

LAMBDA - Pseudo-function of two arguments. Binds the formal parameters given in the first argument (which must be a list) to the actual parameters. The number of actual and formal parameters must fit in the case of EXPR's. Then the form given in the second argument is evaluated. It is possible to give more than two arguments to LAMBDA. In that case, all forms given will be evaluated exactly as in PROG. There may be labels and GO statements etc. The checking on the number of formals is suppressed; instead all extra formals play the same role as PROG variables and are initialized to NIL.

LESSP - Function of two parameters. Return with T if first argument is less than second.

LIST - Evaluates an arbitrary number of arguments and returns with a list of the values.

```
DEFLIS(((LIST(LAMBDA(X A)(EVLIS X A)))FEXPR)
```

MINUS - Function with an arbitrary number of arguments. Returns with .... $-x_{n-2}+x_{n-1}-x_n$. In particular, (MINUS X) = -X and (MINUS X Y) = X - Y.

NIL - Serves as the empty list and at the same time is an atom. CDR(NIL) = NIL. As a function, it has an indeterminate number of arguments and returns NIL as value. This can be useful as comment, but all newly read words are placed on the OBLIST! False is always represented by NIL; F does not have this significance unless it has been defined so explicitly.

NULL - Function of one argument. Returns with T if argument is NIL. NULL has the same action as NOT, which is not included in the system.

```
DEFINE(((NULL(LAMBDA(X)(EQ X NIL)))))
```

NUMBER - Same as NUMBERP in 7090 LISP. Returns with T if argument is a number.

OBLIST - List of all programmer-defined objects. The object OBLIST itself always appears at the bottom of the OBLIST; for technical reasons there is an extra NIL at the top. Hence: CDR(OBLIST) (APVAL(NIL ... all defined objects .... OBLIST)). The OBLIST may be manipulated by the programmer including the element NIL. CAUTION: Great care must be taken not to remove OBLIST from the OBLIST.

PLUS - Function of an indeterminate number of arguments. Will perform a signed additon on integers only. No test on the exceeding of capacity is performed. No test is made on whether the argument(s) is(are) numerical.

PRINT - Function of one argument. Will print the value of the argument and also returns with that value. If the value is atomic, it will only print the atom, no spaces will follow. Numbers are printed with the least number of digits, non-significant zeroes are omitted and the sign when it is a plus. If the element to be printed exceeds the capacity of the line (64 characters), a carriage-return line-feed will automatically be given. Hence, PRINT(X) PRINT(Y) will cause XY to be printed. Any symbol can serve as a letter in a print name even space, carriage return, dot, left parenthesis, etc. For that purpose, they must be read in preceded by '. This will quote the next following single character. Even ' itself may serve as a character when read as ''. Hence, space may be printed with PRINT('), or on the inside level with (PRINT(QUOTE ')). When printing back such a statement from a property list, it cannot be read in again as the ' has disappeared in print. In that case, (EXPR 2160 0 NIL) is preferable. A print name which is longer than 64 characters cannot be printed back; a new line will be given because the name did not fit into the previous line, and this situation will continue to be so.

PROG - Has the same meaning as in 7090 LISP. The first argument is a list of program variables, all set to NIL on entry; then follow statements and labels. Labels are atoms, statements are not. In a PROG, a COND statement is allowed to fall out of the bottom. PROG is left on a RETURN statement, and the value returned is the value of the argument of RETURN.

QUOTE - Prevents its argument from being evaluated. The value is the unevaluated argument. Standard built-in objects need not be quoted in PDP-8 LISP.

```
DEFLIS(((QUOTE(LAMBDA(X A)(CAR X))))FEXPR)
```

READ - Function of no arguments. Reads a single S-expression from tape or keyboard. All identifiers read for the first time are put on the OBLIST. Identifiers may consist of any number of characters and any character except left parenthesis, right parenthesis, dot, space, carr. ret., line-feed, blank, and apostrophe; however, these characters can be "quoted" by preceding them with '. Then, they may again be a character of a name. A name must start with a letter. An object starting with a digit or a plus sign or a minus sign is regarded as a

number (except when preceded by '). An isolated plus sign and minus sign is also a legal name, e.g. (LEFTHANDVARIABLE := A * TEMPERATURE) is a perfectly correct S-expression and will be printed exactly the same; however, when ('234567 + 777 A')B) is read it will print as (234567 + -1 A)B). READ can read two different codes: ASCII and CCITT2 code (5 track). The five track tape must be put into the reader with the three hole side on the same side as on the eight track tape. In the high-speed reader it will need a little bit of extra guiding on the front edge. The system will recognize the code from the leader. This contains no seventh hole but blank or eighth hole for ASCII and contains five blank holes plus the seventh hole =1 for CCITT2. In case CCITT2 code is read, no immediate printing may be done (see Operation of the System). On switching back to keyboard from CCITT2, a blank or leader symbol must be typed first. (Control + shift + commerical at). For switching from low speed to high speed reader see under operating instructions.

RETURN - Function of one argument. If is used to return from a PROG with the value of the argument.

RPLACA - Replace the CAR link of the first argument by the second argument.

RPLACD - Replace the CDR link of the first argument by the second argument. Both these functions must be handled with great care as they can destroy existing list structure. In particular one must never try to replace the property list of a standard atom unless that atom is redefined with DEFINE or DEFLIS. The RPLACA will leave the atom mark intact.

SET - Function of two arguments. Sets the value of the first argument, bound on the association list equal to the value of the second argument. It returns with the value of the second argument.

```
DEFLIS((
   (SET(LAMBDA(X A)(RPLACD(ASSOC(EVAL(CAR X)A)A)
      (EVAL(CADR X)A) ))) )FEXPR)
```

SETQ - Same as SET, but automatically quotes its first argument. SET and SETQ can only effectively be used within LAMBDA or PROG.

```
DEFLIS(((SETQ(LAMBDA(X A)(RPLACD(ASSOC(CAR X)A)
   (EVAL(CADR X)A) ))) )FEXPR)
```

STOP - No arguments. Returns with NIL. Press CONTINUE to restart.

T - Has value T for True. In PDP-8 LISP, T also is a function which will return its argument as value. This is used especially in PROG. COND forms can be abbreviated to the condition pairs only. When the first element evaluates to NIL,the function NIL will return with NIL; but when the first element evaluates to T, the value of the second argument is

taken, e.g. (COND((NULL L)(GO A))) can be abbreviated to ((NULL L)(GO A)). With COND every Boolean not NIL is regarded as T. This is not the case in the abbreviated pairs. The first element must evaluate to T or NIL. Another example is skipping blank tape before READ. Although READ will do this automatically, one has no way of knowing that there has been blank other than: ...(PROG(K L)(GO B) A (SETQ K T) B ((APVAL)(GO A)) (SETQ L(READ))...) K will be NIL unless blank has been skipped by the function APVAL. When it is desirable to have T available as a normal variable one can give:

DEFINE(((T NIL))) RPLACD(T NIL)

This redefines T and then removes all its properties.

DEFINE(((T(LAMBDA(X)X)))) DEFLIS(((T T))APVAL)

TERPRI - This will print carriage return, line-feed and will set the line count to zero. Its value is NIL. PRINT(') or EXPR(2160 6 NIL) EXPR(2160 3 NIL) would not have exactly the same effect, as these do not reset the line count.

Some Important Addresses in the System

It can be necessary to inspect some system variables with EXPR(3172 X -1)

6 contains the cumulative number of times that the garbage collector has been called.

14 contains the last character read

15 contains the shift in CCITT2 code. 0 = letters, 1 = figures.

16 contains the line count, counting from -77 to 0

25 contains the pointer to the association list

27 contains the pointer to the stack

30 contains the pointer to the free list

35 contains the pointer to the first argument

37 contains the pointer to the second argument

41 contains the pointer to the third argument

45 contains the beginning point of the list space minus the end point. This must be an even difference.

0, 1 and 102 are kept free for interrupt service.

100 contains the address of the basic IN routine.

2270 contains the address of the basic OUT routine.

17 contains the GENSYM count from which the GENSYM names are derived.

Operation of the System

The system is provided on BIN code tape and can be loaded by a BIN loader which must not occupy more than the last half page of store. (7700-7777). For the high speed reader a bootstrapping BIN loader is placed in front of the BIN tape which is completely self starting when any normal BIN loader is already in the machine. If the tape is put into the reader on the first stretch of blank tape, it can be loaded by a minimal bootstrap routine:

20 LA, 6011 DEP, 5020 DEP, 6016 DEP, 7012 DEP, 7010 DEP, 7440 DEP, 2027 DEP, 3002 DEP, 5020 DEP, 22 LA, START.

The system can be restarted on

3000LA, START. This will clear the whole system. The OBLIST is emptied and the GENSYM count is zeroed.

3001 LA, START. This will initialize the system but will keep the OBLIST and all properties of the objects. In both cases EVALQUOTE will be entered now. Blank at the beginning of a tape is skipped and sensed for code (ASCII or CCITT2), a pair is read, this pair is evaluated and the value can be printed. Then the next pair is read and evaluated, etc.

Control over what is input and output can be made by a mode number. The mode number is input in the place of the first element of an EVALQUOTE pair. This mode will be set now and the system will again expect a pair (or can accept another mode number). There are four bits in the mode number which have a significance.

When 1 is absent, all characters input will be echoed immediately on low speed input (keyboard or low reader). There is no echo for the high speed reader. Also the function READ will echo.

When 1 is present, there will be no echo but each S-expression of the EVALQUOTE pair will be reproduced after it has been completely read in. If CCITT2 code is read, the direct echo will not work correctly but on mode bit 1, the output will be in ASCII.

When 2 is absent, the value of EVALQUOTE will not be printed but any explicit PRINT function in the program will print.

When 2 is present, the value of EVALQUOTE will be printed.

When 4 is absent, reading takes place on the low speed reader (or keyboard).

When 4 is present, reading takes place on the high speed reader.

When 40 (or 32 in decimal) is present, only the echo is suppressed.

Mode 2 is initially installed. Each change in mode will persist until explicitly changed again, but any error will reinstall mode 2.

An error will result in an error printout which cannot be suppressed by the mode and which has the format:

STOPnumber followed by some indication what has gone wrong, e.g. the name of the function. After an error the system will immediately go on reading the next pair for EVALQUOTE. This can give rise to further errors if, for example, the number of brackets in the input material is wrong.

There are two errors of a special kind:

? will be printed if all working space has been used, whether for stack or list. In this case pushdown list, association list, etc. are cleared and another collection is made. The system then comes to a stop. If CONTINUE is pressed, the next pair for EVALQUOTE will be read. In extreme cases, even by clearing the stack and a list, new space cannot be found and the collector will continue to print ?.

The commercial at sign will be printed if the special collector stack overflows. This will be rarely the case as this stack is only used in the CAR direction. A depth of 42 is provided. Also, a second collector run is tried, and the system stops and must be restarted with CONTINUE.

The system can change the size of the working space by executing EVAL((EXPR 3202 45(MINUS 3725 < last usable address(odd)>))NIL). This will put in 45_8 the difference between the first usable address (=3725) and the last usable address (normally 7677). The change in 45 has no direct effect but comes into play on the next restarting (or on a collector run). Therefore it is advisable to have the change in size followed immediately by EXPR(3000 NIL NIL) to reinitialize the system and to create a new free space. Changing the size gives the possibility to add machine code functions to the system or run a small LISP compiler to compile machine code functions behind the list space.

This issue will run on a PDP-8 as well as on the PDP-8/S. As this issue is an adaption of a LISP system for the 8 only, a patch had to be made for some instructions which did have a different action in the 8/S.

If working space is at premium, one can enlarge the working space to the maximum amount possible, thereby overwriting the loaders by EXPR(3202 45 3726) or decimally: EXPR(1666 37 2006).

The operation of the system could concisely be described by the following program for EVALQUOTE:

```
DEFINE(((EVALQUOTE(LAMBDA()(PROG(FN ARGS VALUE MODE ECHO HI)
START3000(CLEAR)
     (CLEARGENSYM)
ERROR(SETQ MODE 2)
START3001(TERPRI)
     (SETQ ECHO (EQ(LOGAND MODE 45)0))
     (SETQ HI (EQ(LOGAND MODE 4)4))
     (SETQ FN (READ))
     ((EQ(LOGAND MODE 1)0)(GO NOFN))
     (PRINT FN)
```

```
NOFN ((NUMBER FN)(GO NEWMODE))
     (SETQ ARGS (READ))
     ((EQ(LOGAND MODE 1)0)(GO NOARGS))
     (PRINT ARGS)
     (TERPRI)
NOARGS(SETQ VALUE (APPLY FN ARGS NIL))
     ((EQ(LOGAND MODE 2)0)(GO START3001))
     (PRINT VALUE)
     (GO START3001)
NEWMODE(SETQ MODE FN)
     (GO START3001) )))

  (CLEAR(LAMBDA()(RPLACD OBLIST(QUOTE(OBLIST)))))
  (CLEARGENSYM(LAMBDA()(EXPR 3202 17 0)))
  (LOGAND(LAMBDA(X Y)(EXPR 3174 X Y))) ))  EVALQUOTE NIL
```

Some features have not been shown adequately. The function READ will print an immediate echo if ECHO = T. The high speed reader will be used if HI = T. An error will give an error print and will go to ERROR. If anywhere inside an EVALQUOTE pair a blank or trailer code is read this is an error. Blank will only be skipped before a pair. This feature can be used to stop a tape on the high speed reader by ending the previous pair with carr. ret., line feed, blank. The function READ will then react as if the new pair had begun because of the carr. ret. and gives an error stop. The error stop then automatically returns the mode to 2 and hence switches to keyboard.

List of Error Prints

Depending on whether the system prints in decimal or in octal, the following error numbers will be printed:

octal	decimal	kind or error
213	139	value of this variable is not defined
243	163	a number is standing in the place of a function
331	217	built-in function has too few arguments
346	230	built-in function has too many arguments
422	274	this functional argument is no function
501	321	LAMBDA form has to few arguments
522	338	LAMBDA form has too many arguments
554	364	GO, RETURN or a COND with undefined value has been encountered outside of a PROG
567	375	GO has an unknown label
744	484	first argument of SET or SETQ is not atomic
1022	530	wrong number of arguments in this function
1231	665	first element of a pair in DEFINE or DEFLIS is not a name

octal	decimal	kind of error (continued)
1345	741	name in position of a function which is not a function
1501	833	the CAR of an atom has been taken
2432	1306	blank in a LISP expression cannot occur
2504	1348	closing parenthesis cannot occur here
2525	1365	blank after a number has been found
3252	1706	blank after ' has been found
?		working space is full
at sign		collector stack is full

Some Information on the Assembly Listing

The assembly of the system was done on a slightly changed PAL. The indication of an indirect bit was changed from I to Y and the Z for page zero was removed (by a binary change in PAL, 2322/3100 and 2326/0). Furthermore, the following EXPUNGE tape was given:

```
EXPUNGE
FIXMRI
E=0000      E stands for latin ET
FIXMRI
A=1000      A for Add
FIXMRI
X=2000      X for indeX
FIXMRI
B=3000      B for Bring
FIXMRI
S=4000      S for Subroutine jump
FIXMRI
G=5000      G for Go to
W=7000      W for no operation
C=7200      C for Clear acc. May be used in group 1 and 2
F=7100      F for set link False
I=7040      I for Invert acc
U=7020      U for changing the link bit
R=7010      R for Right shift
L=7004      L for Left shift
D=7002      D for Double shift
Q=7001      Q for Qount in acc
M=7500      M for skip on Minus
Z=7440      Z for skip on Zero
T=7420      T for skip on link bit True
N=7410      N for Non or skip Next
K=7404      K for Keys
H=7402      H for Halt
FIXTAB
```

This compressed code shortens the tape considerably and has the advantage of being very easily memorized, e.g. N Z = skip on Non Zero, F D L = make link bit False and shift Double Left. A load-and-go assembler, one pass, has been constructed which can read this same code and in addition has some other features as an ALGOL-like block structure for local names, relocatable loading, general use of symbolic names over page bounds, etc.

An annotated program write-up is given in Dutch. This reflects the use of this load-and-go assembler. For general use this code has been slightly modified in the above-mentioned way to be usable on PAL assemblers as well.