L*(F)

Final Version

by

A. Newell
D. McCracken
G. Robertson
L. DeBenedetti

Department of Computer Science
Carnegie-Mellon University

January 25, 1971

This work is supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-70-C0107) and is monitored by the Air Force Office of Scientific Research.

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF APPENDICES

1. L* is a system on the PDP10 for constructing software systems, which is under development at CMU by A. Newell, D. McCracken, G. Robertson and P. Freeman. This version, L*(F), is the sixth to be designed and the third to become a running system. It is not the final version, by any matter of means. Each of these versions involves radical redesign of one or more aspects of the system. Thus, L*(G) (the next one, now in the process of being developed) is not merely a polishing of L*(F), but differs substantially from it. We are making L*(F) available in a complete form with documentation to let others see what we are doing, to let them play with it, and to submit a version to the discipline of being completed and exposed to external users. At some stage we will simply abandon L*(F) and it will have to live or die on its own.

2. This document provides a description of L*(F) without a detailed design rationale. A few principles are given when they are appropriate to orient the user toward the system.

3. L* intends to be a complete system for running and constructing software systems. It does operate within the limits of the 10-50 monitor system of the PDP10. Completeness implies that one should be able to perform and to construct systems for performing:

Processing of arbitrary data types,
e.g., symbolic structures, lists, numbers, arrays, bit strings, tables, text.

Editing

Compiling and assembling

Language interpreting

Debugging

Operating systems (within the PDP10 monitor),
e.g., resource allocation, space and time accounting, exotic control (parallel and supervisory control).

Communication between user and system,
e.g., external languages, dynamic syntax, displays, etc.

4. L* is a kernel system. It starts with a small kernel of code and data and is grown from within the system. Thus, L* does not perform all the functions above when it exists only as a kernel. It does have means to construct systems

for them all. Whether gracefully or not we'll just have to see.

5. L* is for the professional programmer. It assumes someone sophisticated in systems programming who wants to build up his own system and who will modify any presented system to his own requirements and prejudices.

L* can be used with only a small amount of sophistication in list processing, but this is mostly just for play.

6. L* is intended to be transparent. All mechanisms in the total system are open for understanding and modification. No mechanisms are under the floor.

7. L* is intended to provide complete access to the machine (the PDP10), so that all the 10's facilities can be utilized (except, again, what the monitor prevents).

8. The memory layout of the L*(F) kernel is shown in Appendix 1. The kernel consists of a collection of routines, a few small tables, a large symbol table, and an initial allotment of available space. There is also a high segment (not shown) that contains one word for each symbol in the main segment. These are for symbol descriptions and will be described later. The routines cluster into a series of subsystems, which are used in Appendix 2 to label areas of the kernel. Appendix 2 gives the names of routines and data for each of these subsystems. There are 248 names in all, and Appendix 3 and Appendix 4 list them all in alphabetical order with one line definitions.

These names are those chosen by us, the designers.

However, the names can all be changed.

The kernel is written in MACRO-10 . A listing may be obtained from disk (see Appendix 12 for details). (The names in the MACRO-10 code are not changeable, of course, unless you want to build your own kernel -- which is OK with us.)

9. Symbols. There are symbols in L*, which are addresses (18 bits) and serve to name all the data structures. The symbol for a structure is invariably the address of the first word of the structure. Symbols may be tested for equality (=S) or inequality (<S, >S). New symbols may be obtained by adding an increment to a given symbol (+IS). Conversely, the difference between two symbols (an integer)

may be obtained (-SS).

Symbols may be created (C) or erased (E), and are always tied to the creation or destruction of the structure named by the symbol. That is, symbols do not exist in abstraction from the structures they name. (This follows from the fact that a symbol is the address of some word of the structure it designates.)

10. Types. Every symbol has a type, which determines the structure of the data object the symbol designates. There are originally 6 types : cells (T/C), integers (T/I), characters (T/K), lists (T/L), machine code (T/M) and program lists (T/P). However, other types may be created and types may also be destroyed. Only the minimum necessary types have been set up initially. For instance, there are many kinds of structures in the kernel that do not have types of their own, even though it might be appropriate (e.g., external interfaces, tables of various sorts).

The type itself is abstract. That is, there is no symbol in the system that designates the type. For each type there is a characteristic symbol, which is a symbol of the given type and designates a null structure of that type. These are the symbols T/C, T/I, T/K, T/L, T/M and T/P ; they serve as names of the types.

To each type is associated a type index, which is an integer that is used to access tables organized by type (called type tables). The type tables initially hold space for 15 types, but it is possible to extend the tables to more types.

Symbols can be compared on type (=T), the characteristic symbol of the symbol's type can be obtained (T) or the type index can be obtained (TT). A symbol can also have its type replaced (RT).

The type system for L*(F) is mechanized by having associated with each address a second cell which holds the type index for the given address, hence effectively making it of a given type. These extra cells constitute the high segment. By convention of the PDP10 monitor the relationship between an address X and its corresponding cell in the high segment is an increment of 400000 octal (called TD). The symbol description word for a symbol holds the type index in the address field of the cell (called the S-field in L*). The high order 18 bit field (called the N-field in L*) is not used for anything in the kernel. However, it is available for any use the user wishes to make of it (e.g., as the holder for an attribute-value association list for each symbol).

The main import of having types is that (1) a process may respond differentially to the types of its operands and (2) the availability of type information does not impose structural constraints on the data structures, either by pre-empting bits in the structures themselves or forcing type indicators to be given explicitly with operands. A price is paid, of course, in taking half the total memory to contain the type information. (More exotic ways of holding the type information, which would conserve memory, require more processing to determine the type. There are reasons to prefer the extreme point on the memory-processing exchange to make type determination as fast as possible).

11. Cells (T/C). A cell is simply an isolated word with no specified internal structure. The two operations performable on cells are tests for equality of contents (=C) and replacing the contents of one cell by another (RC). This is the residual type, in that anything not otherwise typed is considered to be T/C .

12. Integers (T/I). An integer is a full word integer in the PDP10 format, i.e., two's-complement. The operations that can be performed on integers are tests for equality (=I) and for inequality (<I, >I), the replacement of one integer value with another (RI), and the standard four arithmetic operations (+I, -I, *I, /I, /RI), where there are two division operations, one for the integer part, one for the remainder.

Since integers are simply bit patterns in full cells, =I and RI are identical with =C and RC . However, both names are included in the kernel to make clear the sets of operations for each data type.

While internally integers are binary two's-complement, for external communication they must be taken to some base. There is a cell, WIB , that holds the base for the integers.

13. Characters (T/K). Each of the 128 characters in the PDP10's 7-bit ASCII character set has a corresponding internal symbol in L*. These make up a separate type. No operations are proper to this type. Names have been given to all non-printing characters; printing characters can use their own print name (with some addition to distinguish the character from a symbol with a one character name).

14. Lists (T/L). The main operating data type initially available in L* is the list. The structure of the lists is entirely conventional. Each list cell holds two symbols, the symbol (or content) of the list cell (S) and the name of the next list cell (N):

```
0       17 18    35
-----------------
|   N   |   S   |
-----------------
```

The null list is NIL, which is T/L like any other list cell:

```
        -----------------
NIL :   | NIL  |  NIL  |
        -----------------
```

However, the routines that erase symbols recognize NIL and will not let it be erased. NIL in the N field of a list cell terminates the list.

The name of the list is the address of the first cell of the list. Thus, there is no way to name a list with no cells on it. The "most null" list possible is:

```
        -----------------
L :     | NIL  |  NIL  |
        -----------------
```

The basic operations on a list are finding the symbol in a list cell (S), finding the next list cell (N), replacing the symbol in a list cell with another (R), and replacing the list cell to be next (RN). Besides these there are processes for inserting a symbol into a list at a point (I) and inserting it after the point (IA) ; also, for deleting a list cell (D) and deleting the cell after (DA).

Two processes exist in the kernel for creating and erasing T/L (C/L, E/L). These illustrate a point about the kernel: that all the processes in the kernel are made available to the user. The two routines above are used in other parts of the kernel, so are made available. They could easily be coded within L* itself using C and E .

15. Machine code (T/M). All the machine code used in the kernel is T/M , which allows it to be recognized. No operations exist initially for manipulating machine code directly, though of course it can be processed by operations of other types (e.g., =C, RC, R, RN, etc.). Create (C) and erase (E) of course work on T/M, just as they do on any type.

16. Program lists (T/P). Program lists are distinct from lists of T/L (i.e., from data lists), which permits one to be executed as program and the other handled as data. There is no reason why there should not be many data types which are structurally identical but are typed separately for some particular purpose.

17. The L*L language. The kernel comes with a single programming language, called L*L , ready to function with ease. The kernel also has T/M, of course, but it is not so easy at the start to create new programs of T/M or modify existing ones. L*L is a list language in the sense that the program structures are lists (i.e., T/P). It also permits the processing of lists (i.e., data structures of T/L or T/P), but it equally permits processing of all other data types. What determines the efficacy of its processing of particular data types is primarily whether the operations are available for the data types. The kernel comes with a good basis for list processing, a reasonable basis for integer processing, and only minimal or indirect bases for the others (including the as-yet-uncreated types).

L*L is a very simple language. It is not the only language that can be created in L*, nor does it even occupy a privileged position, except that one is forced to start with it. It should be possible to construct a second language within L*, such that L*L remains only as a command language, or even is excised from the system entirely.

18. Working cells. In setting up the system a number of cells are required to hold symbols, either temporarily or to define the current context. All these cells are called W cells and their names start with W (mnemonic aid, no structural significance). These cells are T/L, since they are all stacks (i.e., lists which can be pushed and popped).

19. Operand communication. When processes are executed they must acquire their operands and provide their results in some fashion so that the appropriate data can flow within the entire set of processes making up a total activity. In L*L this communication takes place via a single symbol stack of T/L , called W (for the working stack). Thus, each process expects to find its operands in the W stack, and pushes its results into the W stack (after removing its inputs, naturally). Of course, processes can communicate with each other in any other way they wish (e.g., via some set of mutually understood cells or lists), but such arrangements are not part of the conventions of L*L.

20. Interpretation. Each type has an interpreter for symbols of

that type that are to be interpreted. Thus, to define a system of interpretation it suffices to give the interpreters for each type. The initial interpreters are as follows:

Type	Interpreter	Action
T/C	.I/S	Push symbol into W
T/I	.I/S	Push symbol into W
T/K	.I/S	Push symbol into W
T/L	.I/S	Push symbol into W
T/M	.I/M	Execute symbol as machine language subroutine.
T/P	.I/P	Sequence down program list, interpreting each symbol in turn.

Thus, the distinction between program and data is carried by the type of the symbol -- data (T/C, T/I, T/K, T/L) gets put into the operand stack, program (T/M and T/P) gets interpreted further.

The L*L language is essentially as simple as it can be and still provide unrestricted phrase structure. There is no syntax in the program list other than sequencing. Each symbol is interpreted in isolation from its fellows, fore and aft, though of course it is interpreted in the context of the data stack, W, and all the other cells and lists with their current values. But these constitute the semantic context, not the syntactic context, of the symbol.

The act of interpretation occurs not only on a symbol of some type, but in the context of some symbolic structure. For example, a program list can occur for interpretation within another program list, or it can occur for interpretation within a machine language routine. The interpretation is to be the same in some abtract sense. But the total processing is not the same, for the symbolic context is not the same. In particular, the interpreter (for T/P, the type of the symbol in question) cannot find the symbol to be interpreted without knowledge of the symbolic context. Thus, there must be separate interpreters, not only for each type, but for each context in which interpretation can occur. In the initial situation this is only T/P and T/M, although the number of such contexts could increase. For example, one might have L*ALGOL and want to execute T/P programs in it as procedures. The set of contexts in which interpretation can occur is not even necessarily limited to one per type; one could have a polish prefix language (e.g., T/PP) in which routines were written as (F X Y Z) so that the first position (where the F is) is a distinguished context from the others (where the X, Y, Z operands are). Different interpreters would be required for the two contexts. (The remarks of this paragraph may seem abstruse; they are meant to explain the double sets of interpreters that occur throughout Appendices 2, 3 and 10.)

The above interpreters are not the only ones that occur in the kernel. Special interpreters are used for T/K for both reading and writing to an external interface. These operate in conjunction with interpreters for other types, since interpretation is always the result of a set of interpreters (over types and contexts of interpretation).

21. Control. Control operations manipulate the sequence of symbols ultimately interpreted. They do this by manipulating the stacks which contain the information about what symbols and lists remain to be interpreted (WXS, WXN, WHS, WHN). These stacks are T/L and are open to inspection and modification by the user, as well as by the initial control operations provided in the kernel. As a mnemonic guide, all (and only those) routines that affect these stacks start with a period (.). All of these control actions occur in program lists. Control in T/M code occurs according to the conventions of machine coding.

The control actions make use of the structure of the program in terms of lists and sublists, and there is no conditional transfer to another location. Termination (.) stops interpretation of the current program list and ascends to the next higher list. Double termination (..) stops interpretation of both the current program list and the one immediately above it, thus ascending two levels. Repeat (.R) starts over on the present program list (at the same level, thus forming an iterative loop). These control actions can be dependent on data, to wit, on whether the symbol in W is NIL (-) or not NIL (+). (Note: the symbol in W is an input to these processes; hence, it is no longer in W after they have been interpreted.)

Besides termination and repeat, there are two execute operations. .X executes the symbol in W (after popping it to make the operands for it available); .XCX executes the symbol one down in W after going into a new context given by the symbol in the top of W.

The last control action is .Q which is the quote operation. It is the one kernel operation that is not totally context free. It outputs to W the symbol that follows it (the occurrence of .Q, that is) in the program list. Thus, the symbol following a .Q in the program list is never interpreted.

22. External interface. The PDP10 Monitor provides a way for data to move across the interface to and from the various peripheral devices of the PDP10. To use this way requires accepting the data formats specified by the monitor. Thus there are small tables, called interfaces, and buffers to receive and hold sequences of bits for transmission. The kernel comes provided with two such

interfaces, TTY for communicating with a teletype, and DSK for communicating with the disk. Additional communication (to printers, dectapes, etc.) takes place outside of L*, via say PIP. At a later stage of development new interfaces can be built; but the two provided make it possible to get started conveniently.

The DSK interface is set up to read a file called BOOT.LSF and to write a file called FILE.LSF . The TTY is set up to read and write the user's teletype.

23. Name table. A mechanism must be provided right at the start for making correspondences between external names and internal symbols. This is the name table (NT). It consists of a sequential table with pairs of words, the first holding a string representation of the external name in 7-bit characters, the second holding the corresponding symbol (in the S-field). The limitation to one word for the name implies a limitation to 5 characters, where any 7-bit characters are permissible. The three operations that are appropriate with the name table are locating the symbol given the name (LSNT), locating the name given the symbol (LNNT), and creating a symbol given a name (CSNT). In the latter case the type of the new structure must be given (in WTC).

The kernel itself is coded in MACRO-10 assembly language, so that its symbols (on the MACRO-10 listing) are in the MACRO-10 symbol table. All of the symbols of interest in this table are mapped into the initial L* name table (NT1), and appear in Appendices 2, 3 and 4.

24. Read. In reading from an external interface, the interface itself is activated, filling the buffer, as dictated by the PDP10 Monitor conventions. This buffer is scanned to create a list of characters (according to the specifications of the interface). RD, the basic read, simply creates this list (of type in WTCKL) and outputs it to W. Reading of this list in order to extract information from it is done by interpreting it in Read Context with the reading interpreters (.I/K and .TP/K) for T/K . These interpreters execute an action associated with each character. The actions are processes stored in a character table (in WAKT), which has an entry for each of the 128 characters. Thus, reading the list is an active process that executes an arbitrary process for each character (including blank). What actually happens depends entirely on the nature of these actions.

AKT1 holds a set of character actions which serves as the initial interpretation of the input stream. These actions are described in Appendix 10. Essentially they produce the following:

(1) Strings of characters corresponding to names result in their corresponding internal symbols being pushed onto W.

(2) Strings of digits (possibly preceded by + or -) result in an integer being defined according to the base in WIB, with its name input to W.

(3) Semicolon (;) immediately terminates the line, after which normally the next line is read in and interpreted.

(4) Exclamation mark (!) immediately executes the process in the top of W, i.e., it does a .X .

(5) Quote (') immediately executes .Q, so that it puts into W the next character (even if it is the space character) in the input stream.

It can be seen that the last three actions are simply the immediate evocation of three of the control actions available for a program list. The character actions taken together essentially define a simple postfix system, such that one puts the operands first into W following with the process to be executed and then fires it (!). Comments can be hidden behind the semicolon.

The executive (EXEC) continues to read lines from the input interface until an end-of-file is reached. RD itself breaks the input stream into shorter lists on the occurrence of a break character (in WRDBK). This is initially the line-feed character (KLF). (This is needed to avoid getting an entire disk bufferful back as one 640 character list, which could cause initial space problems).

25. Write. Writing to an external interface is done by interpreting symbols in a special context of interpreters. In this Write Context, T/L and T/P symbols are both interpreted with .I/P (or .IP/P). i.e., by sequencing down the list interpreting each symbol in turn. T/K is interpreted by .IWR (or .IPWR) which lays down in the output buffer the 7-bit ASCII code corresponding to the character symbol being interpreted. Buffers are given to the PDP10 Monitor for output to the actual interface as soon as they have been filled up, and also at the end of a complete writing operation (interpretation). Thus, lists of character symbols are mapped by the writing interpreters into the corresponding strings of characters at the actual interface.

26. Assembly. The assembly operations are provided to allow access to the basic machine by depositing (assembly write) and extracting (assembly read) bit patterns in memory.

Any symbol A has associated with it a bit string defined as the low order K bits of A - B where B is taken from the type table in WBTT (the current base type table) and K from the type table in WNBTT (the current number of bits type table) according to the type of A. This association can be two-way, i.e., for a given type one can reconstruct a symbol by adding B to the value of a bit string of length K.

The W cell WPTR is used to hold a machine byte pointer (T/C) for assembly operations. Byte pointers can be created and initialized to point to a given location (CPTR) and can be erased (E). Byte pointers can be moved a given number of bits to the right or left within the current word (MVPTR). There are no special operations for changing the word address of a byte pointer; however, the "replace symbol" list process (R) will accomplish this since the word address field of a byte pointer corresponds to the S-field.

The two assembly operations, reading and writing, are both done by interpretation in a special context of interpreters. The key interpreters for Assembly Read are ones which extract a bit string according to the type of the interpreted symbol (using the byte pointer in WPTR) and push the associated symbol into W (.IEX and .IPEX). For Assembly Write there are interpreters (.IDP and .IPDP) which deposit the bit string associated with the interpreted symbol into memory at the location specified by the byte pointer in WPTR .

There are sets of interpreters in the kernel for both Assembly Read and Assembly Write. They are identical to those in the initial interpreter set (see section 20), except that the interpreters for T/K are changed to .IEX and .IPEX for Assembly Read, or .IDP and .IPDP for Assembly Write.

27. Operating system. Grouped under what we call the L* operating system are processes which perform the following functions :

(1) Error handling and recovery (ERROR).

(2) Debugging capabilities (DEBUG).

(3) Saving of core images for later restarting (SV).

(4) Resetting I/O interfaces for reuse (RSIF, RSIFB,

RSIFR).

(5) Entering monitor mode from L* (HALT).

(6) Entering L* from monitor mode ("CONTINUE" , "START 140" (ST140) , "START 141" (ST141) , "START 142" (ST142)).

(7) Context-changing (PCX, RCX, UCX, SWPCX).

(8) Obtaining core from the Monitor, and returning core (CSP).

(9) Space-exhausted condition handling (routines in SPXTT).

See Appendix 10 for detailed descriptions of the processes appearing above within parentheses.

Definition of the space-exhausted processes (function (9) above) is delayed until the bootstrap; initial available space lists for each type suffice until the bootstrap sequence can define CSP/C , CSP/I , CSP/L , CSP/M and CSP/P and store them into type table SPXTT . See Appendix 12 for details of these create-space processes.

Under function (5) above, there are several other ways of getting into monitor mode from L*, and users may very well discover yet others. The following is a list of conditions we know will cause entry into monitor mode from L* :

(a) Control-C. One will suffice if L* is doing I/O, otherwise two are required.

(b) A PDP10 monitor-detected error. E.g., "ILLEGAL UUO AT USER 000732".

(c) The L* process HALT .

(d) The L* process SV .

(e) Returning from the call on DEBUG in ST141 .

(f) Returning from the call on EXEC in ST140 .

(g) Exiting from the very first call on EXEC made by L* when it first comes up.

28. Structure of the kernel processes. In order that they might be used in many different contexts, most of the kernel processes were coded as independent little units which obtain their inputs and pass back their outputs via machine registers. We call these units the stems of the

processes. Calls on the processes from machine code (e.g., from other processes in the kernel, or possibly from compiled code) are made directly to the stems with registers (R1,R2,etc.) used for input-output communication. (R6 is the highest register available for this purpose; hence, it would not be possible to have a process expecting more than six inputs without adopting some additional conventions). These process stems are all called via a "PUSHJ MSTKP,<stem addr>" instruction, and return to their caller with a "POPJ MSTKP," instruction; i.e., the linkage is always done through the machine stack MSTK .

When kernel processes are called by the machine code interpreters .I/M and .IP/M , input-output communication must be done through W . To handle this, the kernel processes must have "prefixes" which surround the process stem to transfer inputs from W to registers for the stem, and outputs from registers back to W when the stem has completed. ("prefix" is actually somewhat of a misnomer since the prefix does often surround the stem).

The input-output characteristics of the kernel processes are such that only 8 different types of prefix are needed. To conserve space, 8 prefix subroutines (P01, P10, P11, P12, P20, P21, P22 and P33) were created which take a nonstandard input (in R6) telling which process stem is being interfaced with. These prefix subroutines operate by first transferring inputs from W to registers (R1 for W(0), R2 for W(1), etc.). If no output handling is necessary (as it is not in P10 and P20), the process stem (address in R6) is transferred to, and it will return to the caller of the entire process. When output handling is necessary, the stem is called as a subroutine of the prefix subroutine. Then when the stem returns control, outputs are put back into W from the registers, and control is returned to the caller of the entire process.

The name of a kernel process names the entire process including the profix. The name of the stem in the MACRO-10 listing is obtained by putting a "%" in front of the process name. Below is an example of the code for a typical kernel process with a prefix :

```
<prefix>    RN:    JSP    R6,P20     ; call prefix subroutine P20
                                     ;   with R6 pointing to stem
<stem>      %RN:   HRLM   R2,(R1)    ; replace next of W(0) input
                                     ;   by W(1) input
<stem>             RETURN            ; return to caller
```

Note that prefixes of kernel processes are in a preferred position in that they always immediately precede their stem. This will of course not always be the case, particularly if a stem is ever to have more than one prefix.

Appendix 1 - Map of L* Kernel

Decimal	Octal	Contents	Notes
0	0	Registers and Program Status	T/C (except NIL, WPTR, WITT, WIPTT, W, WXS, WHS, WHN : T/L)
96	140	Operating System start locations error locations system initialization prefix routines operations	<-- .START 140 (return) <-- .START 141 (debug) <-- .START 142 (continue after save)
417	641	Symbol Operations symbols types	
519	1007	Data Type Operations cells T/C integers T/I lists T/L T/P	T/M
667	1233	L*L Operations control operand communication interpreters	
814	1456	External Interface Operations name table read write	
1127	2147	Assembly Operations	
1174	2226		

```
           |------------------------------|_
1174  2226 | Symbols of Various Types     |  T/C,T/I,T/K,
           |     T/C,T/I,T/K,T/L,T/M,T/P  |  T/L,T/M,T/P
           |------------------------------|_
1366  2526 | Interfaces                   |
           |     DSK TTY                  |
           |------------------------------|
2002  3722 | Tables                       |
           |     type tables              |  T/C
           |     action character table   |
           |     name table               |
           |     save areas               |
           |     machine stack            |
           |------------------------------|_
3683  7143 | T/C Initial Available Space  |
           |     256 (decimal) cells      |  T/C
           |------------------------------|_
3939  7543 | T/M Initial Available Space  |
           |     256 (decimal) cells      |  T/M
           |------------------------------|_
4195 10143 | T/I Initial Available Space  |
           |     256 (decimal) cells      |  T/I
           |------------------------------|_
4451 10543 | T/L Initial Available Space  |
           |     1280 (decimal) cells     |  T/L
           |     + 64 reserved cells      |
           |------------------------------|_
5795 13243 | T/P Initial Available Space  |
           |     1344 (decimal) cells     |  T/P
           |------------------------------|_
7139 15743
```

```
SYMBOLS -
   SYMBOLS -
        OPERATIONS:     =S <S >S +IS -SS C E
        SYMBOLS:        TRUE NIL

   TYPES -
        OPERATIONS:     =T T TI RT
        SYMBOLS:        T/C T/I T/K T/L T/M T/P TD TTN TTT
        W CELLS:        WTTT


DATA TYPES -
   CELLS T/C -
        OPERATIONS:     =C RC
        SYMBOLS:        T/C R1 R2 R3 R4 R5 R6

   INTEGERS T/I -
        OPERATIONS:     =I <I >I +I -I *I /I PI
        SYMBOLS:        T/I

   CHARACTERS T/K -
        OPERATIONS:     (NONE)
        SYMBOLS:        T/K KBELL KBSP KLF KVT KFF KTAB
                        KCR KSP KALT KTN KBS

   LISTS T/L -
        OPERATIONS:     S N R RN T IA D DA C/L E/L EL
        SYMBOLS:        T/L

   MACHINE CODE T/M -
        OPERATIONS:     (NONE)
        SYMBOLS:        T/M P01 P10 P11 P12 P20 P21 P22 P31

   PROGRAM LISTS T/P -
        OPERATIONS:     (SAME AS FOR LISTS)
        SYMBOLS:        T/P


L*L: INITIAL LANGUAGE AVAILABLE IN L*

CONTROL -
        OPERATIONS:     . .+ .- .. ..+ ..- .R .R+ .R- .X .XCX .Q
                        NOP
OPERAND COMMUNICATION -
        OPERATIONS:     P U V
        W CELLS:        W

INTERPRETERS -
        OPERATIONS:     .I/M .I/P .I/S .IP/M .IP/P .IP/S
        SYMBOLS:        .ITT .IPTT STOP
        W CELLS:        WXS WXN WHS WHN WITT WIPTT


EXTERNAL INTERFACE -
   NAME TABLE -
```

```
        OPERATIONS:     LSNTW LNNTW CSNTW LSNT LNNT CSNT
        SYMBOLS:        NT1 NT1I NT1N
        W CELLS:        WNT WTC

    READ -
        OPERATIONS:     RD .I/K .IP/K ABND ANK ADK A+K A-K ACCD ACCK
        SYMBOLS:        AKT1 NACC ISGN INUM INUMF OCTAL DECML
                        RDCX RDTT RDPTT
        W CELLS:        WRD WTCKL WRDBK WK WAKT WIB

    WRITE -
        OPERATIONS:     WR CVNKL CVIDL .IWR .IPWR
        SYMBOLS:        OCTAL DECML WRCX WRTT WRPTT
        W CELLS:        WWR WTCKL WIB

    INTERFACES AND FILES -
        SYMBOLS:        TTY DSK


ASSEMBLY -
        OPERATIONS:     .IDP .IEX .IPDP .IPEX CPTR MVPTR
        SYMBOLS:        ARTT ARPTT AWTT AWPTT BTT NBTT SEVEN
        W CELLS:        WPTR WBTT WNBTT


OPERATING SYSTEM -
        OPERATIONS:     EXEC DEBUG ERROR PCX RCX UCX SWPCX HALT
                        SV RSTF RSTFB RSTFR CSP
        SYMBOLS:        DBCX SPTT SPXTT ST140 ST141 ST142
                        N/C N/I N/L N/M N/P N/RL
                        MSTK MSTKP MSTKN MSTKM
                        R1SV R2SV R3SV R4SV R5SV MSPSV
        W CELLS:        WDBG WDBCX WSPTT WSPXT WSPRL
```

In the column following the name is the name of the prefix subroutine used by the process, indicating the number of standard inputs and outputs the process has. Although there is no prefix subroutine named P00 , it is used to indicate a process has no inputs and no outputs. (Such processes actually have a no-op as a prefix). A blank entry indicates that the process does not have a prefix for standard handling of inputs and outputs.

*I	P31	MULTIPLY W(1) TIMES W(2), RESULT TO W(0) (T/I)
+I	P31	ADD W(1) TO W(2), RESULT TO W(0) (T/I)
+IS	P21	ADD INTEGER W(0) TO SYMBOL W(1), SYMBOL RESULT W(0)
-I	P31	SUBTRACT W(1) FROM W(2), RESULT TO W(0) (T/I)
-SS	P31	SUBTRACT SYMBOL W(1) FROM SYMBOL W(2), RESULT TO W(0) (T/I)
.	P00	EXIT UNCONDITIONALLY
.+	P10	EXIT IF W(0) NOT = NIL (POP W)
.-	P10	EXIT IF W(0) = NIL (POP W)
..	P00	EXIT TWO LEVELS UNCONDITIONALLY
..+	P10	EXIT TWO LEVELS IF W(0) NOT = NIL (POP W)
..-	P10	EXIT TWO LEVELS IF W(0) = NIL (POP W)
.I/K		INTERPRETER FOR READING (T/K)
.I/M		INITIAL INTERPRETER FOR T/M
.I/P		INITIAL INTERPRETER FOR T/P
.I/S		INITIAL INTERPRETER FOR T/C,T/I,T/L
.IDP		INTERPRETER FOR DEPOSITING (T/K)
.IEX		INTERPRETER FOR EXTRACTING (T/K)
.IP/K		INTERPRETER FOR READING IN T/P CONTEXT (T/K)
.IP/M		INITIAL INTERPRETER FOR T/M IN T/P CONTEXT
.IP/P		INITIAL INTERPRETER FOR T/P IN T/P CONTEXT
.IP/S		INITIAL INTERPRETER FOR T/C,T/I,T/L,T/K IN T/P CONTEXT
.IPDP		INTERPRETER FOR DEPOSITING IN T/P CONTEXT (T/K)
.IPEX		INTERPRETER FOR EXTRACTING IN T/P CONTEXT (T/K)
.IPWR		INTERPRETER FOR WRITING IN T/P CONTEXT (T/K)
.IWR		INTERPRETER FOR WRITING (T/K)
.O	P01	INPUT NEXT SYMBOL TO W AND ADVANCE PAST IT
.R	P00	REPEAT CURRENT LEVEL
.R+	P10	REPEAT CURRENT LEVEL IF W(0) NOT = NIL, POP W
.R-	P10	REPEAT CURRENT LEVEL IF W(0) = NIL, POP W
.X	P10	EXECUTE W(0) AFTER REMOVING IT
.XCX	P20	EXECUTE W(1) IN CONTEXT W(0)
/I	P31	DIVIDE W(2) BY W(1), INTEGER QUOTIENT TO W(0) (T/I)
/RI	P31	DIVIDE W(2) BY W(1), REMAINDER TO W(0) (T/I)
<I	P21	TEST INTEGER W(0) < INTEGER W(1)
<S	P21	TEST SYMBOL W(0) < SYMBOL W(1)
=C	P21	TEST CONTENTS OF CELL W(0) = CONTENTS OF CELL W(1)
=I	P21	TEST INTEGER W(0) = INTEGER W(1)
=S	P21	TEST SYMBOL W(0) = SYMBOL W(1)
=T	P21	TEST IF W(0) IS SAME TYPE AS W(1)
>I	P21	TEST INTEGER W(0) > INTEGER W(1)
>S	P21	TEST SYMBOL W(0) > SYMBOL W(1)
A+K		ACTION FOR CHARACTER +
A-K		ACTION FOR CHARACTER -
ABND		BOUNDARY ACTION
ACCD	P10	ACCUMULATE DIGIT CHARACTER INTO T/I INUM
ACCK	P10	ACCUMULATE NAME CHARACTER INTO T/C NACC
ADK		ACTION FOR DIGIT CHARACTERS

ANK		ACTION FOR NAME CHARACTERS
C	P11	COPY W(0)
C/L	P01	CREATE T/L SYMBOL
CPTR	P11	CREATE BYTE POINTER FOR LOCATION W(0)
CSNT	P11	CREATE SYMBOL WITH NAME W(0) IN NAME TABLE
CSNTW	P21	CREATE SYMBOL WITH NAME W(1) IN NAME TABLE W(0)
CSP	P21	CREATE SPACE FROM MONITOR OF LENGTH W(1) OF TYPE OF W(0)
CVIDL	P11	CONVERT INTEGER W(0) TO DIGIT LIST
CVNKL	P11	CONVERT NAME W(0) TO CHARACTER LIST
D	P10	DELETE CELL W(0)
DA	P10	DELETE CELL AFTER W(0)
DEBUG	P00	ENTER DEBUGGING MODE
E	P10	ERASE SYMBOL W(0)
E/L	P10	ERASE T/L SYMBOL W(0)
EL	P10	ERASE LIST W(0)
ERR0		MACHINE STACK UNDERFLOW ERROR
ERR1		CENTRAL PROCESSOR TRAP ERROR
ERR2		NON-EXISTENT .IPTT ENTRY ERROR
ERR3		NON-EXISTENT .ITT ENTRY ERROR
ERR4		NON-EXISTENT ARPTT ENTRY ERROR
ERR5		NON-EXISTENT ARTT ENTRY ERROR
ERR6		NON-EXISTENT AWPTT ENTRY ERROR
ERR7		NON-EXISTENT AWTT ENTRY ERROR
ERR8		NON-EXISTENT RDPTT ENTRY ERROR
ERR9		NON-EXISTENT RDTT ENTRY ERROR
ERR10		NON-EXISTENT SPXTT ENTRY ERROR
ERR11		NON-EXISTENT WRPTT ENTRY ERROR
ERR12		NON-EXISTENT WRTT ENTRY ERROR
ERR13		SETUWP ERROR RETURN DURING A RESTART
ERR14		CORE UUO ERROR RETURN IN CSP
ERR15		OUT OF SPACE IN NAME TABLE - CSNTW
ERR16		ERROR RETURN FROM OPEN - RD
ERR17		ERROR RETURN FROM LOOKUP - RD
ERR18		ERROR RETURN FROM IN - RD
ERR19		ERROR RETURN FROM OPEN - WR
ERR20		ERROR RETURN FROM ENTER - WR
ERR21		ERROR RETURN FROM OUT - WR
ERR22		ERROR RETURN FROM OUT - .IWR OR .IPWR
ERROR	P00	INTERPRET ERROR ROUTINE IN WERR AFTER DEBUG SWAP
EXEC	P00	MAIN EXECUTIVE : READ AND INTERPRET LINES FROM TTY
HALT	P00	GO INTO MONITOR MODE
I	P20	INSERT W(1) AT W(0) (PUSH AND REPLACE)
IA	P20	INSERT W(1) AFTER W(0) (PUSH, ADVANCE AND REPLACE)
LNNT	P11	LOCATE NAME FOR SYMBOL W(0) IN NAME TABLES
LNNTW	P21	LOCATE NAME FOR SYMBOL W(1) IN NAME TABLE W(0)
LSNT	P11	LOCATE SYMBOL FOR NAME W(0) IN NAME TABLES
LSNTW	P21	LOCATE SYMBOL FOR NAME W(1) IN NAME TABLE W(0)
MVPTR	P20	MOVE BYTE POINTER W(0) W(1) BITS WITHIN CURRENT WORD
N	P11	GET NEXT OF W(0)
NOP	P00	NO OPERATION
P	P12	PUSH W
P01		PREFIX RTN FOR PROCESSES WITH NO INPUT AND 1 OUTPUT
P10		PREFIX RTN FOR PROCESSES WITH 1 INPUT AND NO OUTPUT
P11		PREFIX RTN FOR PROCESSES WITH 1 INPUT AND 1 OUTPUT
P12		PREFIX RTN FOR PROCESSES WITH 1 INPUT AND 2 OUTPUTS
P20		PREFIX RTN FOR PROCESSES WITH 2 INPUTS AND NO OUTPUT

P21		PREFIX RTN FOR PROCESSES WITH 2 INPUTS AND 1 OUTPUT
P22		PREFIX RTN FOR PROCESSES WITH 2 INPUTS AND 2 OUTPUTS
P31		PREFIX RTN FOR PROCESSES WITH 3 INPUTS AND 1 OUTPUT
PCX	P10	PUSH CONTEXT ACCORDING TO CONTEXT LIST W(0)
R	P20	REPLACE SYMBOL OF W(0) BY W(1)
RC	P20	REPLACE CONTENTS OF CELL W(0) BY CONTENTS OF CELL W(1)
RCX	P10	REPLACE CONTEXT ACCORDING TO CONTEXT LIST W(0)
RD	P11	READ FROM INTERFACE W(0). RESULT W(0) = CHARACTER LIST
RI	P20	REPLACE VALUE OF INTEGER W(0) BY VALUE OF INTEGER W(1)
RN	P20	REPLACE NEXT OF W(0) BY W(1)
RSIF	P10	RESET INTERFACE W(0)
RSIFB	P10	RESET INTERFACE BUFFERS (W(0) IS BUFFER HEADER)
RSIFR	P10	RESET INTERFACE RING (W(0) POINTS INTO BUFFER RING)
RT	P20	REPLACE TYPE OF SYMBOL W(0) WITH TYPE INDEX W(1) (T/I)
S	P11	GET SYMBOL OF W(0)
ST140		REENTER EXEC
ST141		ENTER DEBUGGING MODE
ST142		CONTINUE AFTER SAVE
SV	P01	SET UP TO SAVE FOR RESTART
SWPCX	P10	SWAP CONTEXT ACCORDING TO CONTEXT LIST W(0)
T	P11	OUTPUT CHARACTERISTIC SYMBOL FOR TYPE OF W(0)
TI	P21	SET VALUE OF INTEGER W(0) = TYPE INDEX OF W(1)
U	P10	POP W
UCX	P10	POP CONTEXT ACCORDING TO CONTEXT LIST W(0)
V	P22	REVERSE W(0) AND W(1)
WR	P20	WRITE W(1) TO INTERFACE W(0)

Appendix 4 - L*(F) Kernel Data

.IPTT	STANDARD INTERPRETER TYPE TABLE FOR T/P CONTEXT
.ITT	BASIC INTERPRETER TYPE TABLE
AKT1	INITIAL ACTION CHARACTER TABLE
ARPTT	ASSEMBLY READ INTERPRETER TYPE TABLE FOR T/P CONTEXT
ARTT	ASSEMBLY READ INTERPRETER TYPE TABLE
AWPTT	ASSEMBLY WRITE INTERPRETER TYPE TABLE FOR T/P CONTEXT
AWTT	ASSEMBLY WRITE INTERPRETER TYPE TABLE
B/K	INTEGER WHOSE VALUE IS BASE OF CHARACTER SYMBOLS
BTT	BASE TYPE TABLE
DECML	T/I CONSTANT FOR DECIMAL RADIX
DSK	INTERFACE FOR DISK
INUM	T/I NUMBER ACCUMULATOR FOR DIGIT CHARACTER ACTION
INUMF	T/I NUMBER FLAG FOR DIGIT CHARACTER ACTION
ISGN	T/I SIGN INDICATOR FOR DIGIT ACTION
JBAPR	:
JBCNI	:
JBCOR	:
JBFF	: JOB DATA AREA LOCATIONS
JBHRL	: SEE PDP-10 REFERENCE HANDBOOK
JBOPC	: (LOOK IN INDEX).
JBREL	:
JBREN	:
JBSA	:
JBTPC	:
KALT	ALTMODE CHARACTER
KBELL	BELL CHARACTER
KBSP	BACKSPACE CHARACTER
KCR	CARRIAGE RETURN CHARACTER
KFF	FORM FEED CHARACTER
KLF	LINE FEED CHARACTER
KSP	SPACE CHARACTER
KTAB	TAB CHARACTER
KTN	CHARACTER TABLE NUMBER (SIZE)
KVT	VERTICAL TAB CHARACTER
MSPSV	CELL FOR MSTKP CONTENTS AT TIME OF ERROR
MSTK	MACHINE STACK
MSTKM	MACHINE STACK MAXIMUM
MSTKN	MACHINE STACK NUMBER (OPERATING SIZE)
MSTKP	MACHINE STACK POINTER
N/C	NUMBER OF INITIAL T/C AV.SP. CELLS
N/I	NUMBER OF INITIAL T/I AV.SP. CELLS
N/L	NUMBER OF INITIAL T/L AV.SP. CELLS
N/M	NUMBER OF INITIAL T/M AV.SP. CELLS
N/P	NUMBER OF INITIAL T/M AV.SP. CELLS
N/RL	NUMBER OF INITIAL T/L RESERVED AV.SP. CELLS
NACC	NAME ACCUMULATOR FOR NAME CHARACTER ACTION
NBTT	NUMBER OF BITS TYPE TABLE
NIL	NULL LIST (LIST TERMINATOR)
NT1	INITIAL NAME TABLE
NT1I	INITIAL NAME TABLE INDEX (NO. OF ENTRIES)
NT1N	INITIAL NAME TABLE SIZE
OCTAL	T/I CONSTANT FOR OCTAL RADIX
R1	MACHINE REGISTER 1
R1SV	CELL FOR R1 CONTENTS AT TIME OF ERROR
R2	REG. 2
R2SV	CELL FOR R2 CONTENTS AT TIME OF ERROR

R3	REG. 3
R3SV	CELL FOR R3 CONTENTS AT TIME OF ERROR
R4	REG. 4
R4SV	CELL FOR R4 CONTENTS AT TIME OF ERROR
R5	REG. 5
R5SV	CELL FOR R5 CONTENTS AT TIME OF ERROR
R6	REG. 6
RDCX	CONTEXT LIST FOR READ INTERPRETATION
RDPTT	READ INTERPRETER TYPE TABLE FOR T/P CONTEXT
RDTT	READ INTERPRETER TYPE TABLE
SEVEN	T/I CONSTANT =7
SP/C	INITIAL T/C AVAILABLE SPACE LIST
SP/I	INITIAL T/I AVAILABLE SPACE LIST
SP/L	INITIAL T/L AVAILABLE SPACE LIST
SP/M	INITIAL T/M AVAILABLE SPACE LIST
SP/P	INITIAL T/P AVAILABLE SPACE LIST
SP/RL	INITIAL T/L RESERVED AVAILABLE SPACE LIST
SPTT	SPACE TYPE TABLE (HOLDS AV.SP. LISTS)
SPXCX	SPACE EXHAUSTED CONTEXT SWAP LIST
SPXTT	SPACE EXHAUSTED TYPE TABLE (HOLDS SPACE EXHAUSTED PROCESSES)
STOP	T/P EXECUTION CONTEXT DELIMITER FOR WHN STACK
T/C	CHARACTERISTIC SYMBOL FOR TYPE CELL (= 0)
T/I	CHARACTERISTIC SYMBOL FOR TYPE INTEGER (= 0)
T/K	CHARACTERISTIC SYMBOL FOR TYPE CHARACTER (NULL CHARACTER)
T/L	CHARACTERISTIC SYMBOL FOR TYPE LIST (= NIL,NIL)
T/M	CHARACTERISTIC SYMBOL FOR TYPE MACHINE (= RETURN)
T/P	CHARACTERISTIC SYMBOL FOR TYPE PROGRAM (= (NOP))
TD	TYPE DISPLACEMENT (= 400000 OCTAL)
TRUE	SYMBOL FOR POSITIVE RESULT FROM TESTS
TTN	TYPE TABLE SIZE (ALSO MAXIMUM NO. OF TYPES)
TTT	CHARACTERISTIC SYMBOL TYPE TABLE
TTY	INTERFACE FOR USER'S TELETYPE
W	OPERAND COMMUNICATION STACK
WAKT	W CELL FOR CHARACTER ACTION TABLE
WBTT	W CELL FOR BASE TYPE TABLE
WDB	W CELL FOR DEBUG ROUTINE
WDBCX	W CELL FOR DEBUG CONTEXT SWAP LIST
WERR	W CELL FOR ERROR HANDLING ROUTINE
WERRL	W CELL FOR ERROR LOCATION
WHN	HIGHER ROUTINE NEXT STACK
WHS	HIGHER ROUTINE SYMBOL STACK
WIR	W CELL FOR INTEGER RADIX
WIPTT	W CELL FOR PROGRAM CONTEXT INTERPRETER TYPE TABLE
WITT	W CELL FOR INTERPRETER TYPE TABLE
WK	W CELL FOR CHARACTER BEING INTERPRETED
WNBTT	W CELL FOR NUMBER OF BITS TYPE TABLE
WNT	W CELL FOR NAME TABLES
WPTR	W CELL FOR BYTE POINTER
WRCX	CONTEXT LIST FOR WRITE INTERPRETATION
WRD	W CELL FOR READ INTERFACE
WRDBK	W CELL FOR READ BREAK CHARACTER
WRPTT	WRITE INTERPRETER TYPE TABLE FOR T/P CONTEXT
WRTT	WRITE INTERPRETER TYPE TABLE
WSPRL	W CELL FOR RESERVED T/L SPACE
WSPTT	W CELL FOR SPACE TYPE TABLE
WSPXT	W CELL FOR SPACE EXHAUSTED TYPE TABLE

Appendix 4 - L*(F) Kernel Data

WTC	W CELL FOR TYPE BEING CREATED
WTCKL	W CELL FOR TYPE OF CHARACTER LISTS BEING CREATED
WTTT	W CELL FOR CHARACTERISTIC SYMBOL TYPE TABLE
WWR	W CELL FOR WRITE INTERFACE
WXN	CURRENT INSTRUCTION NEXT CELL
WXS	CURRENT INSTRUCTION SYMBOL CELL
ZERO	T/I CONSTANT =0

AR	ASSEMBLY READ STARTING AT W(0) ACCORDING TO LIST W(1)
AW	ASSEMBLY WRITE STARTING AT W(0) ACCORDING TO LIST W(1)
AW6BI	ASSEMBLY-WRITE SIXBIT INITIALIZATION
AWRS	ASSEMBLY-WRITE RESET
CR.LF	WRITE KCR AND KLF TO CURRENT WRITE INTERFACES
CSP/C	ADD 2000 CELLS OF T/C AVAILABLE SPACE
CSP/I	ADD 2000 CELLS OF T/I AVAILABLE SPACE
CSP/L	ADD 2000 CELLS OF T/L AVAILABLE SPACE
CSP/M	ADD 2000 CELLS OF T/M AVAILABLE SPACE
CSP/P	ADD 2000 CELLS OF T/P AVAILABLE SPACE
CSPT	ADD 2000 CELLS TO AVAILABLE SPACE FOR TYPE W(0)
CVSI	CONVERT SYMBOL W(0) TO INTEGER
DCKA	DELETE CURRENT CHARACTER ACTION FOR CHARACTER W(0)
DEF/I	SET WTC TO T/I FOR DEFINING INTEGERS
DEF/L	SET WTC TO T/L FOR DEFINING LISTS
DEF/P	SET WTC TO T/P FOR DEFINING PROGRAM LISTS
DETT	DELETE ENTRY FOR W(1) IN TYPE TABLE W(0)
ENDKL	END CHARACTER LIST
ENDL	ACTION FOR ")" - END LIST
ENDL1	SUBPROGRAM OF ENDL
ENDL2	SUBPROGRAM OF ENDL
ENDL3	SUBPROGRAM OF ENDL
ICKA	INSERT W(1) AS CURRENT CHARACTER ACTION FOR CHARACTER W(0)
IETT	INSERT W(2) AS CURRENT ENTRY OF TYPE TABLE W(0) FOR W(1)
LNKUP	LINK UP W(1) CELLS STARTING WITH W(0) INTO A LIST
PR	PRINT W(0)
PRI	PRINT INTEGER W(0)
PRL	PRINT LIST W(0)
PRLS	PRINT LIST USING PRSTX FOR ELEMENTS
PRN	PRINT NAME W(0)
PRN1	SUBPROGRAM OF PRN
PRN2	SUBPROGRAM OF PRN
PRS	PRINT SYMBOL W(0)
PRST1	PRSTX ROUTINE USED FOR PRSTR
PRSTR	PRINT STRUCTURE W(0)
PRSTX	CURRENT PRINT ROUTINE USED BY PRLS TO PRINT LIST ELEMENTS
RCKA	REPLACE W(1) AS CURRENT CHARACTER ACTION FOR CHARACTER W(0)
RDF	READ DSK FILE NAMED W(0) (WITH EXTENSION "LSF")
RETT	REPLACE ENTRY FOR W(1) IN TYPE TABLE W(0) BY W(2)
RSTRW	RESTORE W(0) FROM WSAVE
SAVE	SAVE FOR RESTART
SAVEW	SAVE W(0) IN WSAVE
SCKA	GET CURRENT CHARACTER ACTION FOR CHARACTER W(0)
SETRD	SET DSK INPUT TO READ FROM FILE NAMED W(0) (EXTENSION "LSF")
SETT	GET ENTRY OF W(1) IN TYPE TABLE W(0)
SETWR	SET DSK OUTPUT TO WRITE TO FILE NAMED W(0) (EXTENSION "LSF")
SPACE	WRITE A BLANK CHARACTER TO CURRENT WRITE INTERFACES
STRKL	START CHARACTER LIST
STRL	ACTION FOR "(" - START LIST
STRL1	SUBPROGRAM OF STRL
STRL2	SUBPROGRAM OF STRL
USEN	ACTION FOR ":" - USE NAME IN W(0)
WRF	WRITE DSK FILE NAMED W(0) (WITH EXTENSION "LSF")
WRWWP	WRITE W(0) TO CURRENT WRITE INTERFACES IN STACK WWR

.TCX	CONTEXT LIST FOR STANDARD INTERPRETATION
ARCX	CONTEXT LIST FOR ASSEMBLY READ INTERPRETATION
AWCX	CONTEXT LIST FOR ASSEMBLY WRITE INTERPRETATION
DBCX	DEBUG SWAP CONTEXT LIST (IN WDBCX)
DNIL	CELL FOR DEBUG SWAP OF NIL
DWIPT	CELL FOR DEBUG SWAP OF WIPTT
DWITT	CELL FOR DEBUG SWAP OF WITT
DWRD	CELL FOR DEBUG SWAP OF WRD
DWRDB	CELL FOR DEBUG SWAP OF WRDBK
DWWR	CELL FOR DEBUG SWAP OF WWR
ITC	TEMPORARY T/T CELL
SPCLI	T/I WORK CELL USED BY CSP/L WHEN RESTORING RESERVED SPACE
T0	TEMPORARY WORK CELL (UNSAFE)
T1	TEMPORARY WORK CELL (UNSAFE)
T2	TEMPORARY WORK CELL (UNSAFE)
T3	TEMPORARY WORK CELL (UNSAFE)
T4	TEMPORARY WORK CELL (UNSAFE)
T5	TEMPORARY WORK CELL (UNSAFE)
TYPL	ASSOCIATION LIST OF TYPES FOR "@" ACTION
W0	WORK CELL (SAFE)
W1	WORK CELL (SAFE)
W2	WORK CELL (SAFE)
W3	WORK CELL (SAFE)
W4	WORK CELL (SAFE)
W5	WORK CELL (SAFE)
WARPT	W CELL FOR T/P CONTEXT ASSEMBLY READ INTERPRETER TYPE TABLE
WARTT	W CELL FOR ASSEMBLY READ INTERPRETER TYPE TABLE
WAWPT	W CELL FOR T/P CONTEXT ASSEMBLY WRITE INTERPRETER TYPE TABLE
WAWTT	W CELL FOR ASSEMBLY WRITE INTERPRETER TYPE TABLE
WC	W CELL TO HOLD CURRENT LIST BEING CREATED
WFLR	W CELL TO HOLD W FLOOR
WSAVE	W CELL USED BY SAVEW
WUSEN	W CELL TO HOLD USEN SIGNAL

Meaning of Code Letters: K=Kernel, B=Bootstrap, P=Process, D=Data .

*I	KP	MULTIPLY W(1) TIMES W(2), RESULT TO W(0) (T/I)
+I	KP	ADD W(1) TO W(2), RESULT TO W(0) (T/I)
+IS	KP	ADD INTEGER W(0) TO SYMBOL W(1), SYMBOL RESULT W(0)
-I	KP	SUBTRACT W(1) FROM W(2), RESULT TO W(0) (T/I)
-SS	KP	SUBTRACT SYMBOL W(1) FROM SYMBOL W(2), RESULT TO W(0) (T/I)
.	KP	EXIT UNCONDITIONALLY
.+	KP	EXIT IF W(0) NOT = NIL (POP W)
.-	KP	EXIT IF W(0) = NIL (POP W)
..	KP	EXIT TWO LEVELS UNCONDITIONALLY
..+	KP	EXIT TWO LEVELS IF W(0) NOT = NIL (POP W)
..-	KP	EXIT TWO LEVELS IF W(0) = NIL (POP W)
.I/K	KP	INTERPRETER FOR READING (T/K)
.I/M	KP	INITIAL INTERPRETER FOR T/M
.I/P	KP	INITIAL INTERPRETER FOR T/P
.I/S	KP	INITIAL INTERPRETER FOR T/C,T/I,T/L
.ICX	BD	CONTEXT LIST FOR STANDARD INTERPRETATION
.IDP	KP	INTERPRETER FOR DEPOSITING (T/K)
.IEX	KP	INTERPRETER FOR EXTRACTING (T/K)
.IP/K	KP	INTERPRETER FOR READING IN T/P CONTEXT (T/K)
.IP/M	KP	INITIAL INTERPRETER FOR T/M IN T/P CONTEXT
.IP/P	KP	INITIAL INTERPRETER FOR T/P IN T/P CONTEXT
.IP/S	KP	INITIAL INTERPRETER FOR T/C,T/I,T/L,T/K IN T/P CONTEXT
.IPDP	KP	INTERPRETER FOR DEPOSITING IN T/P CONTEXT (T/K)
.IPEX	KP	INTERPRETER FOR EXTRACTING IN T/P CONTEXT (T/K)
.IPTT	KD	STANDARD INTERPRETER TYPE TABLE FOR T/P CONTEXT
.IPWR	KP	INTERPRETER FOR WRITING IN T/P CONTEXT (T/K)
.ITT	KD	STANDARD INTERPRETER TYPE TABLE
.IWR	KP	INTERPRETER FOR WRITING (T/K)
.O	KP	INPUT NEXT SYMBOL TO W AND ADVANCE PAST IT
.R	KP	REPEAT CURRENT LEVEL
.R+	KP	REPEAT CURRENT LEVEL IF W(0) NOT = NIL, POP W
.R-	KP	REPEAT CURRENT LEVEL IF W(0) = NIL, POP W
.X	KP	EXECUTE W(0) AFTER REMOVING IT
.XCX	KP	EXECUTE W(1) IN CONTEXT W(0)
/I	KP	DIVIDE W(2) BY W(1), INTEGER QUOTIENT TO W(0) (T/I)
/RI	KP	DIVIDE W(2) BY W(1), REMAINDER TO W(0) (T/I)
<I	KP	TEST INTEGER W(0) < INTEGER W(1)
<S	KP	TEST SYMBOL W(0) < SYMBOL W(1)
=C	KP	TEST CONTENTS OF CELL W(0) = CONTENTS OF CELL W(1)
=I	KP	TEST INTEGER W(0) = INTEGER W(1)
=S	KP	TEST SYMBOL W(0) = SYMBOL W(1)
=T	KP	TEST IF W(0) IS SAME TYPE AS W(1)
>I	KP	TEST INTEGER W(0) > INTEGER W(1)
>S	KP	TEST SYMBOL W(0) > SYMBOL W(1)
A+K	KP	ACTION FOR CHARACTER +
A-K	KP	ACTION FOR CHARACTER -
ABND	KP	BOUNDARY ACTION
ACCD	KP	ACCUMULATE DIGIT CHARACTER INTO T/I INUM
ACCK	KP	ACCUMULATE NAME CHARACTER INTO T/C NACC
ADK	KP	ACTION FOR DIGIT CHARACTERS
AKT1	KD	INITIAL ACTION CHARACTER TABLE
ANK	KP	ACTION FOR NAME CHARACTERS
AR	BP	ASSEMBLY READ STARTING AT W(0) ACCORDING TO LIST W(1)

ARCX	BD	CONTEXT LIST FOR ASSEMBLY READ INTERPRETATION
ARPTT	KD	ASSEMBLY READ INTERPRETER TYPE TABLE FOR T/P CONTEXT
ARTT	KD	ASSEMBLY READ INTERPRETER TYPE TABLE
AW	BP	ASSEMBLY WRITE STARTING AT W(0) ACCORDING TO LIST W(1)
AW6BI	BP	ASSEMBLY-WRITE SIXBIT INITIALIZATION
AWCX	BD	CONTEXT LIST FOR ASSEMBLY WRITE INTERPRETATION
AWPTT	KD	ASSEMBLY WRITE INTERPRETER TYPE TABLE FOR T/P CONTEXT
AWRS	BP	ASSEMBLY-WRITE RESET
AWTT	KD	ASSEMBLY WRITE INTERPRETER TYPE TABLE
B/K	KD	INTEGER WHOSE VALUE IS BASE OF CHARACTER SYMBOLS
BTT	KD	BASE TYPE TABLE
C	KP	COPY W(0)
C/L	KP	CREATE T/L SYMBOL
CPTR	KP	CREATE BYTE POINTER FOR LOCATION W(0)
CR.LF	BP	WRITE KCR AND KLF TO CURRENT WRITE INTERFACES
CSNT	KP	CREATE SYMBOL WITH NAME W(0) IN NAME TABLE
CSNTW	KP	CREATE SYMBOL WITH NAME W(1) IN NAME TABLE W(0)
CSP	KP	CREATE SPACE FROM MONITOR OF LENGTH W(1) OF TYPE OF W(0)
CSP/C	BP	ADD 2000 CELLS OF T/C AVAILABLE SPACE
CSP/I	BP	ADD 2000 CELLS OF T/I AVAILABLE SPACE
CSP/L	BP	ADD 2000 CELLS OF T/L AVAILABLE SPACE
CSP/M	BP	ADD 2000 CELLS OF T/M AVAILABLE SPACE
CSP/P	BP	ADD 2000 CELLS OF T/P AVAILABLE SPACE
CSPT	BP	ADD 2000 CELLS TO AVAILABLE SPACE FOR TYPE W(0)
CVIDL	KP	CONVERT INTEGER W(0) TO DIGIT LIST
CVNKL	KP	CONVERT NAME W(0) TO CHARACTER LIST
CVST	BP	CONVERT SYMBOL W(0) TO INTEGER
D	KP	DELETE CELL W(0)
DA	KP	DELETE CELL AFTER W(0)
DBCX	BD	DEBUG SWAP CONTEXT LIST (IN WDBCX)
DCKA	BP	DELETE CURRENT CHARACTER ACTION FOR CHARACTER W(0)
DEBUG	KP	ENTER DEBUGGING MODE
DECML	KD	T/I CONSTANT FOR DECIMAL RADIX
DEF/I	BP	SET WTC TO T/I FOR DEFINING INTEGERS
DEF/L	BP	SET WTC TO T/L FOR DEFINING LISTS
DEF/P	BP	SET WTC TO T/P FOR DEFINING PROGRAM LISTS
DETT	BP	DELETE ENTRY FOR W(1) IN TYPE TABLE W(0)
DNIL	BD	CELL FOR DEBUG SWAP OF NIL
DSK	KD	INTERFACE FOR DISK
DWIPT	BD	CELL FOR DEBUG SWAP OF WIPTT
DWITT	BD	CELL FOR DEBUG SWAP OF WITT
DWRD	BD	CELL FOR DEBUG SWAP OF WRD
DWRDB	BD	CELL FOR DEBUG SWAP OF WRDBK
DWWR	BD	CELL FOR DEBUG SWAP OF WWR
E	KP	ERASE SYMBOL W(0)
E/L	KP	ERASE T/L SYMBOL W(0)
EL	KP	ERASE LIST W(0)
ENDKL	BP	END CHARACTER LIST
ENDL	BP	ACTION FOR ")" - END LIST
ENDL1	BP	SUBPROGRAM OF ENDL
ENDL2	BP	SUBPROGRAM OF ENDL
ENDL3	BP	SUBPROGRAM OF ENDL
ERR0	KP	MACHINE STACK UNDERFLOW ERROR
ERR1	KP	CENTRAL PROCESSOR TRAP ERROR
ERR2	KP	NON-EXISTENT .IPTT ENTRY ERROR
ERR3	KP	NON-EXISTENT .ITT ENTRY ERROR

ERR4	KP	NON-EXISTENT ARPTT ENTRY ERROR
ERR5	KP	NON-EXISTENT ARTT ENTRY ERROR
ERR6	KP	NON-EXISTENT AWPTT ENTRY ERROR
ERR7	KP	NON-EXISTENT AWTT ENTRY ERROR
ERR8	KP	NON-EXISTENT RDPTT ENTRY ERROR
ERR9	KP	NON-EXISTENT RDTT ENTRY ERROR
ERR10	KP	NON-EXISTENT SPXTT ENTRY ERROR
ERR11	KP	NON-EXISTENT WRPTT ENTRY ERROR
ERR12	KP	NON-EXISTENT WRTT ENTRY ERROR
ERR13	KP	SETUWP ERROR RETURN DURING A RESTART
ERR14	KP	CORE UUO ERROR RETURN IN CSP
ERR15	KP	OUT OF SPACE IN NAME TABLE - CSNTW
ERR16	KP	ERROR RETURN FROM OPEN - RD
ERR17	KP	ERROR RETURN FROM LOOKUP - RD
ERR18	KP	ERROR RETURN FROM IN - RD
ERR19	KP	ERROR RETURN FROM OPEN - WR
ERR20	KP	ERROR RETURN FROM ENTER - WR
ERR21	KP	ERROR RETURN FROM OUT - WR
ERR22	KP	ERROR RETURN FROM OUT - .IWR OR .IPWR
ERROR	KP	INTERPRET ERROR ROUTINE IN WERR AFTER DEBUG SWAP
EXEC	KP	MAIN EXECUTIVE : READ AND INTERPRET LINES FROM TTY
HALT	KP	GO INTO MONITOR MODE
I	KP	INSERT W(1) AT W(0) (PUSH AND REPLACE)
IA	KP	INSERT W(1) AFTER W(0) (PUSH, ADVANCE AND REPLACE)
ICKA	BP	INSERT W(1) AS CURRENT CHARACTER ACTION FOR CHARACTER W(0)
IETT	BP	INSERT W(2) AS CURRENT ENTRY OF TYPE TABLE W(0) FOR W(1)
INUM	KD	T/I NUMBER ACCUMULATOR FOR DIGIT CHARACTER ACTION
INUMF	KD	T/I NUMBER FLAG FOR DIGIT CHARACTER ACTION
ISGN	KD	T/I SIGN INDICATOR FOR DIGIT ACTION
ITO	BD	TEMPORARY T/I CELL
JBAPR	KD	:
JBCNT	KD	:
JBCOR	KD	:
JBFF	KD	: JOB DATA AREA LOCATIONS
JBHRL	KD	: SEE PDP-10 REFERENCE HANDBOOK
JBOPC	KD	: (LOOK IN INDEX).
JBREL	KD	:
JBREN	KD	:
JBSA	KD	:
JBTPC	KD	:
KALT	KD	ALTMODE CHARACTER
KBELL	KD	BELL CHARACTER
KBSP	KD	BACKSPACE CHARACTER
KCR	KD	CARRIAGE RETURN CHARACTER
KFF	KD	FORM FEED CHARACTER
KLF	KD	LINE FEED CHARACTER
KSP	KD	SPACE CHARACTER
KTAB	KD	TAB CHARACTER
KTN	KD	CHARACTER TABLE NUMBER (SIZE)
KVT	KD	VERTICAL TAB CHARACTER
LNKUP	BP	LINK UP W(1) CELLS STARTING WITH W(0) INTO A LIST
LNNT	KP	LOCATE NAME FOR SYMBOL W(0) IN NAME TABLES
LNNTW	KP	LOCATE NAME FOR SYMBOL W(1) IN NAME TABLE W(0)
LSNT	KP	LOCATE SYMBOL FOR NAME W(0) IN NAME TABLES
LSNTW	KP	LOCATE SYMBOL FOR NAME W(1) IN NAME TABLE W(0)
MSPSV	KD	CELL FOR MSTKP CONTENTS AT TIME OF ERROR

MSTK	KD MACHINE STACK
MSTKM	KD MACHINE STACK MAXIMUM
MSTKN	KD MACHINE STACK NUMBER (OPERATING SIZE)
MSTKP	KD MACHINE STACK POINTER
MVPTR	KP MOVE BYTE POINTER W(0) W(1) BITS WITHIN CURRENT WORD
N	KP GET NEXT OF W(0)
N/C	KD NUMBER OF INITIAL T/C AV.SP. CELLS
N/I	KD NUMBER OF INITIAL T/I AV.SP. CELLS
N/L	KD NUMBER OF INITIAL T/L AV.SP. CELLS
N/M	KD NUMBER OF INITIAL T/M AV.SP. CELLS
N/P	KD NUMBER OF INITIAL T/M AV.SP. CELLS
N/RL	KD NUMBER OF INITIAL T/L RESERVED AV.SP. CELLS
NACC	KD NAME ACCUMULATOR FOR NAME CHARACTER ACTION
NBTT	KD NUMBER OF BITS TYPE TABLE
NIL	KD NULL LIST (LIST TERMINATOR)
NOP	KP NO OPERATION
NT1	KD INITIAL NAME TABLE
NT1I	KD INITIAL NAME TABLE INDEX (NO. OF ENTRIES)
NT1N	KD INITIAL NAME TABLE SIZE
OCTAL	KD T/I CONSTANT FOR OCTAL RADIX
P	KP PUSH W
P01	KP PREFIX RTN FOR PROCESSES WITH NO INPUT AND 1 OUTPUT
P10	KP PREFIX RTN FOR PROCESSES WITH 1 INPUT AND NO OUTPUT
P11	KP PREFIX RTN FOR PROCESSES WITH 1 INPUT AND 1 OUTPUT
P12	KP PREFIX RTN FOR PROCESSES WITH 1 INPUT AND 2 OUTPUTS
P20	KP PREFIX RTN FOR PROCESSES WITH 2 INPUTS AND NO OUTPUT
P21	KP PREFIX RTN FOR PROCESSES WITH 2 INPUTS AND 1 OUTPUT
P22	KP PREFIX RTN FOR PROCESSES WITH 2 INPUTS AND 2 OUTPUTS
P31	KP PREFIX RTN FOR PROCESSES WITH 3 INPUTS AND 1 OUTPUT
PCX	KP PUSH CONTEXT ACCORDING TO CONTEXT LIST W(0)
PR	BP PRINT W(0)
PRI	BP PRINT INTEGER W(0)
PRL	BP PRINT LIST W(0)
PRLS	BP PRINT LIST USING PRSTX FOR ELEMENTS
PRN	BP PRINT NAME W(0)
PRN1	BP SUBPROGRAM OF PRN
PRN2	BP SUBPROGRAM OF PRN
PRS	BP PRINT SYMBOL W(0)
PRST1	BP PRSTX ROUTINE USED FOR PRSTR
PRSTR	BP PRINT STRUCTURE W(0)
PRSTX	BP CURRENT PRINT ROUTINE USED BY PRLS TO PRINT LIST ELEMENTS
R	KP REPLACE SYMBOL OF W(0) BY W(1)
R1	KD MACHINE REGISTER 1
R1SV	KD CELL FOR R1 CONTENTS AT TIME OF ERROR
R2	KD REG. 2
R2SV	KD CELL FOR R2 CONTENTS AT TIME OF ERROR
R3	KD REG. 3
R3SV	KD CELL FOR R3 CONTENTS AT TIME OF ERROR
R4	KD REG. 4
R4SV	KD CELL FOR R4 CONTENTS AT TIME OF ERROR
R5	KD REG. 5
R5SV	KD CELL FOR R5 CONTENTS AT TIME OF ERROR
R6	KD REG. 6
RC	KP REPLACE CONTENTS OF CELL W(0) BY CONTENTS OF CELL W(1)
RCKA	BP REPLACE W(1) AS CURRENT CHARACTER ACTION FOR CHARACTER W(0)
RCX	KP REPLACE CONTEXT ACCORDING TO CONTEXT LIST W(0)

RD	KP	READ FROM INTERFACE W(0). RESULT W(0) = CHARACTER LIST
RDCX	KD	CONTEXT LIST FOR READ INTERPRETATION
RDF	BP	READ DSK FILE NAMED W(0) (WITH EXTENSION "LSF")
RDPTT	KD	READ INTERPRETER TYPE TABLE FOR T/P CONTEXT
RDTT	KD	READ INTERPRETER TYPE TABLE
RETT	BP	REPLACE ENTRY FOR W(1) IN TYPE TABLE W(0) BY W(2)
RI	KP	REPLACE VALUE OF INTEGER W(0) BY VALUE OF INTEGER W(1)
RN	KP	REPLACE NEXT OF W(0) BY W(1)
RSIF	KP	RESET INTERFACE W(0)
RSIFB	KP	RESET INTERFACE BUFFERS (W(0) IS BUFFER HEADER)
RSIFR	KP	RESET INTERFACE RING (W(0) POINTS INTO BUFFER RING)
RSTRW	BP	RESTORE W(0) FROM WSAVE
RT	KP	REPLACE TYPE OF SYMBOL W(0) WITH TYPE INDEX W(1) (T/I)
S	KP	GET SYMBOL OF W(0)
SAVE	BP	SAVE FOR RESTART
SAVEW	BP	SAVE W(0) IN WSAVE
SCKA	BP	GET CURRENT CHARACTER ACTION FOR CHARACTER W(0)
SETRD	BP	SET DSK INPUT TO READ FROM FILE NAMED W(0) (EXTENSION "LSF")
SETT	BP	GET ENTRY OF W(1) IN TYPE TABLE W(0)
SETWR	BP	SET DSK OUTPUT TO WRITE TO FILE NAMED W(0) (EXTENSION "LSF")
SEVEN	KD	T/I CONSTANT =7
SP/C	KD	INITIAL T/C AVAILABLE SPACE LIST
SP/I	KD	INITIAL T/I AVAILABLE SPACE LIST
SP/L	KD	INITIAL T/L AVAILABLE SPACE LIST
SP/M	KD	INITIAL T/M AVAILABLE SPACE LIST
SP/P	KD	INITIAL T/P AVAILABLE SPACE LIST
SP/RL	KD	INITIAL RESERVED T/L AVAILABLE SPACE LIST
SPACE	BP	WRITE A BLANK CHARACTER TO CURRENT WRITE INTERFACES
SPCLI	BD	T/I WORK CELL USED BY CSP/L WHEN RESTORING RESERVED SPACE
SPTT	KD	SPACE TYPE TABLE (HOLDS AV.SP. LISTS)
SPXCX	KD	SPACE EXHAUSTED CONTEXT SWAP LIST
SPXTT	KD	SPACE EXHAUSTED TYPE TABLE (HOLDS SPACE EXHAUSTED PROCESSES)
ST140	KP	REENTER EXEC
ST141	KP	ENTER DEBUGGING MODE
ST142	KP	CONTINUE AFTER SAVE
STOP	KD	T/P EXECUTION CONTEXT DELIMITER FOR WHN STACK
STRKL	BP	START CHARACTER LIST
STRL	BP	ACTION FOR "(" - START LIST
STRL1	BP	SUBPROGRAM OF STRL
STRL2	BP	SUBPROGRAM OF STRL
SV	KP	SET UP TO SAVE FOR RESTART
SWPCX	KP	SWAP CONTEXT ACCORDING TO CONTEXT LIST W(0)
T	KP	OUTPUT CHARACTERISTIC SYMBOL FOR TYPE OF W(0)
T/C	KD	CHARACTERISTIC SYMBOL FOR TYPE CELL (= 0)
T/I	KD	CHARACTERISTIC SYMBOL FOR TYPE INTEGER (= 0)
T/K	KD	CHARACTERISTIC SYMBOL FOR TYPE CHARACTER (NULL CHARACTER)
T/L	KD	CHARACTERISTIC SYMBOL FOR TYPE LIST (= NIL,NIL)
T/M	KD	CHARACTERISTIC SYMBOL FOR TYPE MACHINE (= RETURN)
T/P	KD	CHARACTERISTIC SYMBOL FOR TYPE PROGRAM (= (NOP))
T0	BD	TEMPORARY WORK CELL (UNSAFE)
T1	BD	TEMPORARY WORK CELL (UNSAFE)
T2	BD	TEMPORARY WORK CELL (UNSAFE)
T3	BD	TEMPORARY WORK CELL (UNSAFE)
T4	BD	TEMPORARY WORK CELL (UNSAFE)
T5	BD	TEMPORARY WORK CELL (UNSAFE)
TD	KD	TYPE DISPLACEMENT (= 400000 OCTAL)

TT	KP	SET VALUE OF INTEGER W(0) = TYPE INDEX OF W(1)
TRUE	KD	SYMBOL FOR POSITIVE RESULT FROM TESTS
TTN	KD	TYPE TABLE SIZE (ALSO MAXIMUM NO. OF TYPES)
TTT	KD	CHARACTERISTIC SYMBOL TYPE TABLE
TTY	KD	INTERFACE FOR USER'S TELETYPE
TYPL	BD	ASSOCIATION LIST OF TYPES FOR "@" ACTION
U	KP	POP W
UCX	KP	POP CONTEXT ACCORDING TO CONTEXT LIST W(0)
USEN	BP	ACTION FOR ":" - USE NAME IN W(0)
V	KP	REVERSE W(0) AND W(1)
W	KD	OPERAND COMMUNICATION STACK
W0	BD	WORK CELL (SAFE)
W1	BD	WORK CELL (SAFE)
W2	BD	WORK CELL (SAFE)
W3	BD	WORK CELL (SAFE)
W4	BD	WORK CELL (SAFE)
W5	BD	WORK CELL (SAFE)
WAKT	KD	W CELL FOR CHARACTER ACTION TABLE
WAPPT	BD	W CELL FOR T/P CONTEXT ASSEMBLY READ INTERPRETER TYPE TABLE
WARTT	BD	W CELL FOR ASSEMBLY READ INTERPRETER TYPE TABLE
WAWPT	BD	W CELL FOR T/P CONTEXT ASSEMBLY WRITE INTERPRETER TYPE TABLE
WAWTT	BD	W CELL FOR ASSEMBLY WRITE INTERPRETER TYPE TABLE
WBTT	KD	W CELL FOR BASE TYPE TABLE
WC	BD	W CELL TO HOLD CURRENT LIST BEING CREATED
WDB	KD	W CELL FOR DEBUG ROUTINE
WDBCX	KD	W CELL FOR DEBUG CONTEXT SWAP LIST
WERR	KD	W CELL FOR ERROR HANDLING ROUTINE
WERRL	KD	W CELL FOR ERROR LOCATION
WFLR	BD	W CELL TO HOLD W FLOOR
WHN	KD	HIGHER ROUTINE NEXT STACK
WHS	KD	HIGHER ROUTINE SYMBOL STACK
WIB	KD	W CELL FOR INTEGER RADIX
WIPTT	KD	W CELL FOR PROGRAM CONTEXT INTERPRETER TYPE TABLE
WITT	KD	W CELL FOR INTERPRETER TYPE TABLE
WK	KD	W CELL FOR CHARACTER BEING INTERPRETED
WNBTT	KD	W CELL FOR NUMBER OF BITS TYPE TABLE
WNT	KD	W CELL FOR NAME TABLES
WPTR	KD	W CELL FOR BYTE POINTER
WR	KP	WRITE W(1) TO INTERFACE W(0)
WRCX	KD	CONTEXT LIST FOR WRITE INTERPRETATION
WRD	KD	W CELL FOR READ INTERFACE
WRDBK	KD	W CELL FOR READ BREAK CHARACTER
WRF	BP	WRITE DSK FILE NAMED W(0) (WITH EXTENSION "LSF")
WRPTT	KD	WRITE INTERPRETER TYPE TABLE FOR T/P CONTEXT
WRTT	KD	WRITE INTERPRETER TYPE TABLE
WRWWR	BP	WRITE W(0) TO CURRENT WRITE INTERFACES IN STACK WWR
WSAVE	BD	W CELL USED BY SAVEW
WSPRL	KD	W CELL FOR RESERVED T/L SPACE
WSPTT	KD	W CELL FOR SPACE TYPE TABLE
WSPXT	KD	W CELL FOR SPACE EXHAUSTED TYPE TABLE
WTC	KD	W CELL FOR TYPE BEING CREATED
WTCKL	KD	W CELL FOR TYPE OF CHARACTER LISTS BEING CREATED
WTTT	KD	W CELL FOR CHARACTERISTIC SYMBOL TYPE TABLE
WUSEN	BD	W CELL TO HOLD USEN SIGNAL
WWR	KD	W CELL FOR WRITE INTERFACE
WXN	KD	CURRENT INSTRUCTION NEXT CELL

Appendix 7 - Complete List of System Names

```
WXS     KD CURRENT INSTRUCTION SYMBOL CELL
ZERO    KD T/I CONSTANT =0
```

Appendix 8 - Abbreviations Used For Names

A	AFTER ACTION ASSEMBLY
ACC	ACCUMULATE ACCUMULATOR
AKT	ACTION-CHARACTER-TABLE
ALT	ALTMODE
AR	ASSEMBLY-READ
AW	ASSEMBLY-WRITE
B	BASE BIT BREAK BUFFER
BND	BOUNDARY
BSP	BACKSPACE
C	CELL COPY CREATE
CR	CARRIAGE-RETURN
CV	CONVERT
CX	CONTEXT
D	DELETE DISPLACEMENT DIGIT
DB	DEBUG
DP	DEPOSIT
E	ERASE
EX	EXTRACT
F	FLAG
FF	FORM-FEED
H	HIGHER
I	INTEGER INSERT INDEX
IF	INTERFACE
JP	JOB-DATA-AREA-LOCATION
K	CHARACTER
KT	CHARACTER-TABLE
L	LIST LOCATE LOCATION
LF	LINE-FEED
M	MACHINE MAXIMUM
N	NEXT NAME NUMBER
NT	NAME-TABLE
NUM	NUMBER
P	PROGRAM PUSH PREFIX POINTER
PTR	POINTER
Q	QUOTE
R	REPLACE REPEAT RESERVE REGISTER RING
RD	READ
RS	RESET
S	SYMBOL STACK
SGN	SIGN
SP	SPACE
ST	START
STK	STACK
SV	SAVE
SWP	SWAP
T	TYPE TABLE
TT	TYPE-TABLE
U	POP-UP
V	REVERSE
VT	VERTICAL-TAB
W	WORKING-CELL
WR	WRITE
X	EXECUTE EXHAUSTED
/X	OF-TYPE-X
.I	INTERPRETER
.IP	INTERPRETER-FOR-T/P-CONTEXT
%X	STEM-OF-PROCESS-X

I. OUTLINE OF INITIAL BOOTSTRAP

1. DEFINE RCKA - REPLACE CURRENT CHARACTER ACTION
   W(0)=CHARACTER, W(1)=ACTION

2. DEFINE A:(...) BY CHARACTER ACTIONS FOR : ( )
   USEN STRL ENDL

3. ADD ABND TO CHARACTER ACTIONS FOR ! : ( )

4. SET UP DEBUG SWAP LIST

5. DEFINE WORKING CELLS (T'S AND W'S) AND SAVE AND RESTORE PROCESSES (SAVEW RSTRW).

6. DEFINE TYPE DECLARATION PROCESSES AND @ ACTION
   DEF/P DEF/L DEF/I
   A@T TO MAKE A OF TYPE T BY CHARACTER ACTION FOR @

7. DEFINE "..." TO CREATE LIST OF CHARACTERS (OF TYPE WTCKL.S) BY CHARACTER ACTION FOR "

8. DEFINE PRINT PROCESSES
   WRWWR SPACE CR.LF PRN PRT PRS PR PRL

9. DEFINE TYPE TABLE AND CURRENT CHARACTER ACTION PROCESSES
   RETT IETT DETT SETT
   ICKA DCKA SCKA (RCKA DEFINED IN 1.)

10. DEFINE ELEMENTARY SPACE PROCESSES
    CSPT LNKUP CSP/I CSP/L CSP/M CSP/P CSP/C

11. DEFINE ASSEMBLY PROCESSES
    AW AR

12. DEFINE FILE NAMING PROCESSES
    AW6BI AWRS
    SETRD SETWR (W(0)=SYMBOL AND USES EXTERNAL NAME OF IT .LSP)
    RDF WRF (READ AND WRITE FROM DSK FILE W(0))

II. CHARACTER ACTIONS AFTER BOOTSTRAP

```
A - Z   - ANK - NAME ACTION
0 - 9   - ADK - DIGIT ACTION
+       - A+K - PLUS ACTION
-       - A-K - MINUS ACTION
'       - .Q - QUOTE ACTION
;       - . - COMMENT ACTION (EXITS LINE)
!       - (ABND .ICX .XCX) - EXECUTE ACTION (ALSO BOUNDARY ACTION)
:       - (ABND USEN) - NAMING ACTION
(       - (ABND STRL) - START LIST ACTION
)       - (ABND ENDL) - END LIST ACTION
@       - TYPE ACTION
"       - CHARACTER LIST ACTION
KSP,KCR,KLF,KFF,KVT,KTAB   - ABND - BOUNDARY ACTION
OTHER PRINTING CHARACTERS - ANK
ALL OTHERS - NOP
```

character action table (in W cell WAKT) with the 7-bit code for the character being interpreted. .I/K exits upon return from the character action.

.I/M .I/M is the interpreter for T/M used in all interpretation contexts defined in the kernel and bootstrap. It appears as the entry for T/M in all the following interpreter type tables: .ITT, ARTT, AWTT, RDTT and WRTT. .I/M's only action is to call the symbol to be interpreted (input in R1) as a machine code subroutine.

.I/P .I/P is the interpreter for T/P used in all interpretation contexts defined in the kernel and bootstrap. It appears as the entry for T/P in all the following interpreter type tables: .ITT, ARTT, AWTT, RDTT and WRTT. .I/P operates as follows:

Descend: Push WXS onto WHS and WXN onto WHN.
Put STOP into WXN to delimit scope of current T/P execution.
Put symbol to be interpreted (input to .I/P in R1) into WXS.

Interpret: Interpret symbol in WXS by calling the appropriate interpreter obtained from the current interpreter type table contained in W cell WIPTT. (This is interpretation within the scope of a T/P list, hence WIPTT is used rather than WITT).
Upon return, continue.

Advance: If WXN.S = NIL, go to Ascend.
If WXN.S= STOP, go to Exit.
Otherwise, put the symbol of the cell pointed to by WXN into WXS, and advance WXN to point to the next cell on the list (by putting the link of the cell pointed to by WXN back into WXN).
Then go to Interpret.

Ascend: Pop WHS into WXS and WHN into WXN.
Go to Advance.

Exit: Pop WHS into WXS and WHN into WXN.
Exit from T/P execution context by returning to the original caller of .I/P.

+IS

+IS adds the value of W(0) to symbol W(1). The symbol result is output W(0).

-I

-I subtracts the value of W(1) from the value of W(2), storing the result as the value of W(0). W(0) is left as output.

-SS

-SS subtracts the symbol W(1) from the symbol W(2), storing the integer result as the value of W(0). W(0) is left as output.

.

. exits one level unconditionally by putting NIL into WXN.

.+

.+ exits one level if input W(0) is not NIL by putting NIL into WXN. The input is always removed.

.-

.- exits one level if input W(0) is NIL by putting NIL into WXN. The input is always removed.

..

.. exits two levels unconditionally by putting NIL into both WXN and WHN.

..+

..+ exits two levels if the input W(0) is not NIL by calling .. . The input is always removed.

..-

..- exits two levels if the input W(0) is NIL by calling .. . The input is always removed.

.I/K

.I/K is a reading interpreter used for T/K in Read Context. It appears as the entry for T/K in the interpreter type table RDTT. .I/K obtains the (character) symbol to be interpreted from R1, stores it in cell WK, and then interprets the appropriate character action. The character action is obtained by indexing into the current character action table (in W cell WAKT) with the 7-bit code for the character being interpreted. .I/K exits upon return from the character action.

.I/M .I/M is the interpreter for T/M used in all interpretation contexts defined in the kernel and bootstrap. It appears as the entry for T/M in all the following interpreter type tables: .ITT, ARTT, AWTT, RDTT and WRTT. .I/M's only action is to call the symbol to be interpreted (input in R1) as a machine code subroutine.

.I/P .I/P is the interpreter for T/P used in all interpretation contexts defined in the kernel and bootstrap. It appears as the entry for T/P in all the following interpreter type tables: .ITT, ARTT, AWTT, RDTT and WRTT. .I/P operates as follows:

Descend: Push WXS onto WHS and WXN onto WHN.
Put STOP into WXN to delimit scope of current T/P execution.
Put symbol to be interpreted (input to .I/P in R1) into WXS.

Interpret: Interpret symbol in WXS by calling the appropriate interpreter obtained from the current interpreter type table contained in W cell WIPTT. (This is interpretation within the scope of a T/P list, hence WIPTT is used rather than WITT).
Upon return, continue.

Advance: If WXN.S = NIL, go to Ascend.
If WXN.S= STOP, go to Exit.
Otherwise, put the symbol of the cell pointed to by WXN into WXS, and advance WXN to point to the next cell on the list (by putting the link of the cell pointed to by WXN back into WXN).
Then go to Interpret.

Ascend: Pop WHS into WXS and WHN into WXN.
Go to Advance.

Exit: Pop WHS into WXS and WHN into WXN.
Exit from T/P execution context by returning to the original caller of .I/P.

.I/S .I/S is the data interpreter and appears in .ITT for T/L, T/I, T/K and T/C, and in ARTT, AWTT, RDTT and WRTT for T/L, T/I and T/C. The operation of .I/S is simply to push onto W the symbol being interpreted.

.IDP .IDP is the interpreter for depositing, and appears as the entry for T/K in interpreter type table AWTT. Let A be the symbol being interpreted (input to .IDP in R1).

.IDP first obtains the entry for A in the current bit number type table (in W cell WNBTT). This entry is an integer whose value is now deposited into the S-field (bits 6-11) of the PDP10 byte pointer in W cell WPTR.

Next the entry in the current base type table (in W cell WBTT) for A is obtained. (It is an integer; call its value B).

Finally, the value A - B is deposited using an IDPB (Increment and DePosit Byte) instruction on the byte pointer in WPTR.

.IEX .IEX is the interpreter for extracting, and appears as the entry for T/K in interpreter type table ARTT. Let A be the symbol being interpreted (input to .IEX in R1).

.IEX first obtains the entry for A in the current bit number type table (in W cell WNBTT). This entry is an integer whose value is now deposited into the S-field (bits 6-11) of the PDP10 byte pointer in W cell WPTR.

Next, a bit pattern is extracted using an ILDB (Increment and LoaD Byte) instruction on the byte pointer in WPTR. Then the entry in the current base type table (in W cell WBTT) for A is obtained. (It is an integer; call its value B).

Finally, the symbol obtained by adding to B the bit pattern extracted above is pushed onto W.

.IP/K .IP/K is the reading interpreter that appears as the entry for T/K in interpreter type table .TPTT. .IP/K is identical to .I/K except that it gets its input (the symbol to be interpreted) from WXS rather than R1 since .IP/K is used within the scope of T/P interpretation.

.IP/M .IP/M is the machine code interpreter for interpreting T/M symbols appearing in T/P lists. It is the entry for

T/M in all the following interpreter type tables: .IPTT, ARPTT, AWPTT, RDPTT and WRPTT. It operates identically with .I/M except that its input (the symbol to be interpreted) is gotten from WXS rather than R1.

.IP/P .IP/P is the interpreter for T/P symbols appearing in T/P lists. It is the entry for T/P in all the following interpreter type tables: .IPTT, ARPTT, AWPTT, RDPTT and WRPTT.

Its operation is not like the operation of .I/P; in fact, all .IP/P does is to push WXS onto WHS and WXN onto WHN, move the contents of WXS to WXN, and then exit.

In the initial L* system, .IP/P will always be called by the "Interpret" part of .I/P, and hence when .IP/P returns to .I/P, WXN will be set up so that "Advance" will start down the new T/P list.

.IP/S .IP/S is the interpreter for data appearing in T/P lists. It appears in .IPTT for T/L, T/T, T/K and T/C, and in ARPTT, AWPTT, RDPTT and WRPTT for T/L, T/T and T/C. It operates identically with .I/S except that its input (the symbol to be interpreted) is obtained from WXS rather than R1.

.IPDP .IPDP is the interpreter for depositing within the scope of a T/P list, and is the entry for T/K in interpreter type table AWPTT. It operates identically with .IDP except that it obtains the symbol to interpret from WXS rather than R1.

.IPEX .IPEX is the interpreter for extracting within the scope of a T/P list, and is the entry for T/K in interpreter type table ARPTT. It operates identically with .IEX except that it obtains the symbol to interpret from WXS rather than R1.

.IPWR .IPWR is an interpreter for writing and is used to interpret T/K appearing in T/P lists when in Write Context. It appears in interpreter type table WRPTT as the entry for T/K. .IPWR is identical with .IWR except that its input (the symbol to be interpreted) is gotten from WXS rather than R1.

.IWR .IWR is a writing interpreter used for T/K in Write

Context. It appears as the entry for T/K in the interpreter type table WRTT.

.IWR subtracts the base for the symbol being interpreted (obtained as an integer from the type table in W cell WBTT) from the symbol itself. The resulting bit pattern is deposited into the appropriate output buffer if there is room, otherwise an output operation is done first.

There are two implicit inputs to .IWR, both related to the particular interface being written to. These are the channel number (from the right half of the first word of the interface block) and the buffer header address (from the left half of the fourth word), and they are set up by WR before it interprets its W(1) input.

Note that the size of the above bit pattern deposited in the buffer is determined by the particular output interface, and not by any L* mechanism.

If .IWR must do an output operation (because the buffer is full), and an error return occurs, error location ERR22 is called.

.Q .Q (Quote) outputs the next symbol in the current program list and causes interpretation to skip over it. If .Q appears as the last symbol on a program list, it will cause control to ascend until the following symbol is found.

.R .R repeats execution of the current level by putting the top element of WHS, which is the higher routine symbol stack, into WXN, which holds the next operation on the current level.

.R+ .R+ repeats the current level if input W(0) is not NIL by putting the top element of WHS into WXN. The input is always removed.

.R- .R- repeats the current level if input W(0) is NIL by putting the top element of WHS into WXN. The input is always removed.

.X .X interprets (eXecutes) the symbol W(0) (after removing it from the stack) by calling the appropriate interpreter obtained from the interpreter type table contained in W cell WITT.

.XCX .XCX interprets (executes) the symbol W(1) in the context specified by context list W( ), which is in the form expected by PCX, RCX, and UCX.

The operations PCX and then RCX are performed on the context list W(0). The symbol W(1) is then interpreted by calling the appropriate interpreter from the interpreter type table contained in W cell WITT. Upon return from the interpreter, the original context is restored by performing UCX on the context list which was input W(0).

/I /I divides the value of W(2) by the value of W(1), storing the quotient as the value of W(0). W(0) is left as output.

/RI /RI divides the value of W(2) by the value of W(1), storing the remainder as the value of W(0). W(0) is left as output.

<I <I tests if the value of W(0) < the value of W(1). If not, the output is NIL. If so, the output is W(1) (unless W(1) is NIL, in which case the output is TRUE).

<S <S tests if the symbol W(0) is less than the symbol W(1). If not, the output is NIL. If so, the output is W(1) (unless W(1) = NIL, in which case the output is TRUE).

=C =C tests if the value of W(0) = the value of W(1). If not, the output is NIL. If so, the output is W(1) (unless W(1) is NIL, in which case the output is TRUE).

=I =I is identical to =C .

=S =S tests if the symbol W(0) = the symbol W(1). If not, the output is NIL. If so, the output is W(1) (unless W(1) = NIL, in which case the output is TRUE).

=T   =T gets the type indexes of W(0) and W(1) which are stored as the contents of the cells whose addresses are W(0) + TD and W(1) + TD respectively. Then it tests if the type of W(0) is the same as the type of W(1). If not, the result is NIL. If so, the output is W(1) (unless W(1) = NIL, in which case the output is TRUE).

>T   >T tests if the value of W(0) > the value of W(1). If not, the output is NIL. If so, the output is W(1) (unless W(1) = NIL, in which case the output is TRUE).

>S   >S tests if the symbol W(0) is greater than the symbol W(1). If not, the output is NIL. If so, the output is the symbol W(1) (unless W(1) is NIL, in which case the output is TRUE).

A+K   A+K is the initial character action (entry in AKT1) for the character '+'. It operates by first testing if the number flag integer INUMF = 1 (indicating that only digit characters have occurred since the last boundary character), and if so sets INUMF = -1 to indicate a name rather than an integer is to be recognized. Thus, for example, a string like "_13+_" (where '_' is a boundary character) will be recognized as a name "13+" rather than the integer +13. A+K completes its operation by always calling ACCK to accumulate the current character being interpreted (obtained from cell WK) into NACC, the name accumulator cell.

A-K   A-K is the initial character action (entry in AKT1) for the character '-'. It operates identically to A+K except for the additional action of updating the integer sign indicator ISGN. ISGN is used to keep a (mod 2) count of the number of '-' characters since the last boundary character, and thus represents sign for integers.

ABND   ABND is the initial character action (entry in AKT1) for tab (KTAB), line feed (KLF), vertical tab (KVT), form feed (KFF), carriage return (KCR), and space (KSP). These characters are called "boundary characters" since they act as boundaries for the recognition of names and integers.

ABND first tests the name accumulator cell NACC to see if the previous character was also a boundary action, and if so it exits. Next it tests the value of INUMF to see whether it is a name or an integer that should be recognized.

If INUMF = 1, a new T/I cell is created and given the value of the integer INUM if ISGN = 0, or the complement of that value if ISGN = 1.

If INUMF doesn't = 1 (hence is -1), then ABND calls LSNT (Locate Symbol in Name Table) with the address of the name accumulator cell NACC as input. If the symbol is not located, CSNT (Create Symbol in Name Table) is called to create an entry for the name accumulated in NACC.

Finally, ABND pushes onto W as output the integer created in the first case, or the symbol located or created. Then the contents of the four cells NACC, INUM ,INUMF and ISGN are set to zero, and ABND exits.

ACCD ACCD is used (by ADK) to accumulate digit characters for recognition of integers. It has one standard input which is a digit character symbol whose digit value (0,1,...,9) it accumulates into integer INUM by first multiplying INUM by the current radix in W cell WIB and then adding to INUM the new digit value. A special check is made for overflow in the multiplication, and the high order bit of INUM is set on if overflow occurs. This was necessary to make recognition of negative numbers written in twos complement form work (e.g. octal 400000000000).

ACCK ACCK is used by ANK, ADK, A+K and A-K to accumulate characters into the name accumulator cell NACC. It has one standard input which is a character symbol whose 7-bit code is shifted into the low order position of cell NACC. If more than five characters are accumulated (between boundary actions), the first ones are shifted out the left of the accumulator and are lost. Bit 0 (leftmost bit in cell) of NACC is not reset after the shift so that it may retain a spurious setting if characters are shifted out the left of the accumulator.

ADK ADK is the initial character action (entry in AKT1) for all the digit characters (0,1,...,9). Its operation is as follows:

If INUMF = 0, indicating that the previous character was a boundary character, INUMF is set = 1 to indicate that an integer is provisionally to be recognized. Then ACCK is called, with the current digit character being interpreted (from cell WK) as input, to accumulate the character into the name accumulator NACC.

Next, a test is made to see if INUMF = -1, and if so, ADK exits since there is no chance for an integer to be recognized; otherwise ACCD is called to accumulate the current digit character into the integer accumulator INUM.

ANK ANK is the "name" action, and is the initial character action (entry in AKT1) for all printing characters except : (TAB,LF,VT, FF,CR,SP,!,',",1,2,3,4,5,6,7,8,9,+,-,;). It operates by setting INUMF = -1 to indicate a name is to be recognized and then calling ACCK to accumulate the current character (from cell WK) into name accumulator NACC.

C C (Copy) first accesses the space type table in W cell WSPTT to find the available space list for the type of input W(0).

If the available space list is NIL, C first checks to see if input W(0) is T/L, and if so stores the reserved space list from W cell WSPRL as the available space list. (This is necessary for execution of the space-exhausted routine, which is responsible for restoring the reserved space after it has allocated more list space). Next, C swaps into space-exhausted context by calling SWPCX with context swap list SPXCX as input. Then it executes (interprets) the space exhausted process obtained as the entry for the type of W(0) in the type table in W cell WSPXT. Upon return, SWPCX is called again with input SPXCX to swap back to the previous context, and control transfers to the beginning of C for another try.

If the available space list is not found to be exhausted, C unlinks the top cell, copies the full-word contents of input W(0) into it, and leaves it as output W(0).

C/L C/L (Create type List) is similar to C except that it has no input telling what type of cell to obtain from available space and how to initialize it. It always outputs a T/L cell which has not been initialized (and hence still links into the available space list for T/L).

CPTR CPTR is to be used to create PDP10 byte pointers and initialize them to point at the start of a given location (input W(0)). Most common usage of CPTR will be to create a pointer to put into W cell WPTR for use with .IDP and .IEX.

CPTP calls C with the symbol T/C as input to obtain a cell for the pointer. It initializes S=0, P=36, I=X=0, Y=(W(0) input) in the pointer and leaves it as output W(0).

CSNT — CSNT is used to add a new entry in the current name table (specified by W cell WNT) for an input name W(0). It merely calls CSNTW to create the symbol for the input name in the name table residing in WNT.

CSNTW — CSNTW adds a new entry to a particular name table (input W(0)) for an input name W(1). The name input is a cell containing right-justified ASCII characters (as in the name accumulator NACC into which ACCK accumulates characters).

CSNT first gets the current index for the input name table, which is located in the word immediately preceding the table itself. The index is compared with the table size from the next preceding word, and EPR15 is called at this point if the index is not less than the size. Otherwise, the table is not full, so the new entry is made as follows:

C is called with input WCT.S to create a new symbol of the type of the symbol in W cell WCT. At the location of the new entry, which is determined by adding twice the index to the input table address, are stored the packed characters of the input name in the first word and the newly created symbol in the second word (right half). Then the index of the table is incremented by one, and CSNTW exits with the new symbol as output W(0).

CSP — CSP is the routine which allows additional space to be allocated from the monitor, or space to be returned. Input W(1) is the size (T/I) of the change in allocation; positive if space is to be obtained, negative if space is to be returned to the monitor. Since the monitor only allocates in 1K blocks (2000 octal), the value of W(1) should be a multiple of that size. Any new space obtained from the monitor is made to have the same type as that of input W(0). W(0) is not used if the value of W(1) is negative.

Output from CSP is the address of the block of space obtained from the monitor if the value of input W(1) was positive, otherwise the output is NIL since no space was obtained.

CSP does some housekeeping in updating the Job Data Area locations JOBFF, JOBSA and JOBHRL (L* symbols JBFF, JBSA and JBHRL) to ensure that the monitor SAVE function saves the correct amount of core in both the low and high segments.

If an error return from the CORE UUO occurs, indicating that the requested additional core is not currently available, error location ERR14 is called.

CVIDL CVIDL expects an integer (T/I) as input W(0) and outputs a list of the same type as WTCKL.S of character symbols which are the digit characters for the representation of the integer in the current radix (in W cell WIB). If the value of integer W(0) is negative, CVIDL outputs a list with a minus sign character followed by the digits of the absolute value of the input.

CVIDL operates by successive divisions by the current radix (from W cell WIB), using the remainders to build the list of digit characters.

CVNKL CVNKL expects a cell containing packed right-justified ASCII characters as input W(0). (The same form as the name cells in the name table). By successive shifting, masking, and testing for null characters, CVNKL builds a list of the same type as WTCKL.S of character symbols for the packed characters (in left to right order from the packed cell) and outputs it.

D D deletes the symbol in list cell W(0). If W(0) is not the last cell in a list, then the full-word contents of the next cell is copied into it, and the next cell is erased (using E). If W(0) is the last cell in the list, the symbol in that cell is replaced by NIL.

DA DA (Delete After) deletes the symbol in the cell after W(0) by replacing the link of W(0) by the link of the cell after W(0). Then it calls E to erase the cell which was previously after W(0).

DEBUG DEBUG is used to swap into "debugging context" for execution of diagnostics, etc. when something has gone wrong. It operates as in the following T/P list:

(WBDCX S SWPCX WDB S .X WBDCX S SWPCX)

I.e., it swaps into Debug Context, executes the contents of cell WBD, and then upon return swaps back to the previous context.

DEBUG is called when a "START 141" is done in monitor mode.

E E (Erase) first checks if W(0) = NIL, and if so exits without erasing. Otherwise it returns W(0) to the front of the available space list which is the entry for the type of W(0) in the type table in W cell WSPTT.

E/L E/L (Erase type List) assumes that input W(0) is T/L and returns it to the front of the available space list which is the entry for T/L in the type table in W cell WSPTT.

EL EL assumes that input W(0) is a list and iterates down the list erasing (via E) each cell on the list.

ERR0 - ERR22 These error locations are called at the site where an error is detected in a kernel process to initiate handling of the error. There is a unique error location for each of the 23 different errors which can be detected in the kernel. Each error location is a "JSP R6,ERROR" instruction which transfers control to the central error routine ERROR with R6 retaining the identity of the particular error location.

Below are listed the separate error locations with a description of the conditions causing each error.

ERR0 This symbol is put on the bottom of the machine stack (MSTK) so that an attempt to do a RETURN with an empty stack will give control to the ERR0 error location. Of course, the ERR0 at the bottom of the stack will not be "seen" if it is popped off as data rather than being treated as a return link.

ERR1 This symbol sits in the right half of Job Data Area location JOBAPR (L* symbol JBAPR), and thus is where control is passed when one of the conditions enabled by an APRENB UUO is detected by the monitor. The traps enabled by L* are pushdown overflow, memory protection violation, and non-existent memory flag.

When an ERR1 occurs, the Job Data Area locations JOBCNI and JOBTPC (L* symbols JBCNI and JBTPC) contain useful information.

JBCNI contains the state of the APR (Arithmetic PRocessor) when the trap occurred, and can be used to discover which of the three possible conditions actually caused the trap, as follows:

In the right half of the JBCNI word,

bit 19 (200000 octal) indicates pushdown overflow,
bit 22 (20000 octal) indicates memory protection flag,
bit 23 (10000 octal) indicates non-existent memory.

JBTPC contains the PC (Program Counter) of the next instruction to be executed when the trap occurred. (Thus the right half is the address of the next instruction). This will help locate the offending instruction.

ERR2 - ERR12 These are error locations called by an interpreter when it attempts to interpret some symbol with a type table which has no valid interpreter for the type of that symbol. Error locations ERR2 - ERR12 are merely used to fill in the unused entries in kernel type tables, one error location per type table as follows:

ERR2	.IPTT
ERR3	.ITT
ERR4	ARPTT
ERR5	ARTT
ERR6	AWPTT
ERR7	AWTT
ERR8	RDPTT
ERR9	RDTT
ERR10	SPXTT
ERR11	WRPTT
ERR12	WRTT

SPXTT is really an exception since it is not an interpreter type table, but holds processes. Thus ERR10 will be interpreted as a T/M process, while the other error locations above will be called directly as if they were interpreters.

ERR13 A part of the cleanup SV has to do after return from the SAVE in monitor mode is to reissue the SETUWP UUO to reenable writing in the high segment (the monitor SAVE command sets write protection back on as a side effect). Error location ERR13 is called if an error return from the SETUWP UUO occurs, which is an indication that either the monitor system does not have a two-register capability (impossible on our system) or that the user has been meddling without write privileges (see PDP-10 Reference Handbook, under "meddling").

ERR14 This error location is called in CSP if an error return occurs from the CORE UUO attempting to allocate core from the monitor. This indicates that the additional amount of core requested is not available, either because of hardware limitations or because a large load of other users is on the system.

ERR15 This is the "out of space in name table" error detected by CSNTW (Create Symbol in Name Table W(0)) when the index for the input name table is not less than the size of the name table.

ERR16 This error occurs if RD gets an error return when attempting to OPEN the interface to be read from, indicating that the device (specified by the device name in word 3 of the input interface block) does not exist or is allocated to another job.

ERR17 This error occurs if RD gets an error return when attempting to LOOKUP the file to be read from, indicating that the user's directory was not found or that the file (specified in words 5 and 6 of the interface block) was not found or was read protected.

ERR18 This error occurs if RD gets an error return while doing an input (IN instruction) from the interface. The error detected will be one represented by one of the file status bits (see PDP-10 Reference Handbook, under "File: status bits"). Due to an oversight in the L*F system, the file status is not made readily available when an ERR18 does occur.

ERR19 This is an error detected by WR which corresponds to ERR16 detected by RD, i.e. it indicates the specified device trying to be OPENed does not exist or is allocated to another job.

ERR20 This error occurs if WR gets an error return from an ENTER UUO (which is analogous to LOOKUP, but for output files). It indicates one of several possible error conditions:

The user's directory was not found (if the device has a directory).

The file to be written was found to already exist and was being currently written or renamed, or was write protected.

ERR21 This error occurs if WR gets an error return from doing the final output (OUT instruction) of a write operation. The possible errors are those which can be reflected in the file status bits (see PDP-10 Reference Handbook, under "File: status bits"), although due to an oversight the status is not readily available when an ERR21 occurs.

ERR22 This error occurs if .IWR or .IPWP (the writing interpreters) gets an error return from output operations. The conditions are identical to those for ERR21.

ERROR ERROR uniformly handles kernel errors represented by error locations ERR0 - ERR22 by initiating appropriate context-swapping and executing an arbitrary user-written error routine, while preserving the identity of the particular error. Its detailed operation is as follows:

ERROR expects a non-standard input in R6 which is the address of the current error location +1. ERROR uses this input to store the current error location in W cell WERRL. Next, the contents of R1 - R5 and MSTKP are copied into cells R1SV - R5SV and MSPSV respectively, and reserved machine stack space is opened up by increasing effective stack size from MSTKN to MSTKM. Then the symbol ERROR is replaced by HALT in the current error location so that a recursive error will execute HALT rather than call ERROR recursively. Next, a swap into Debug Context is made by executing SWPCY with input WDBCX.S, and then the symbol in W cell WERR is executed. Upon return, SWPCX is called to swap back out of Debug Context, the symbol ERROR is put back into the current error location, and R1 - R5 and MSTKP are restored from the save cells R1SV - R5SV and MSPSV. (Note that this effectively closes off reserved machine stack space since MSTKP was copied into MSPSV before reserved space was opened above.) Control will return to the caller of the current error location (errors are initiated by calling the appropriate error location), unless of course the machine stack pointer was altered in cell MSPSV before it was restored from there.

EXEC EXEC is the main executive which is called when the L* kernel is run for the first time. It reads from the current read interface (WRD.S) and executes the resulting list in Read Context. If an end-of-file is detected from the read interface, it is reset and EXEC exits. Specifically, EXEC operates as in the following T/P list:

```
(( WRD S RD P .- P WEXEC I RDCX .XCX WEXEC S WEXEC D EL .R)
WRD S RSIF)
```

where WEXEC is a save cell private to EXEC.

Calls on EXEC can of course be nested within other executions of EXEC to any level. In fact, the "START 140" command in monitor mode causes such a nested call on EXEC.

HALT HALT goes into monitor mode (without releasing I/O devices currently in use). A "CONTINUE" command from monitor mode will cause control to return to the caller of HALT. A "START 140" or "START 141" command may also be issued from monitor mode. (See ST140 and ST141).

I I inserts symbol W(1) in front of the symbol in cell W(0). It creates a new cell of the same type as W(0). The full-word contents of W(0) is copied into the new cell, then the link of W(0) is linked to the new cell, and finally the symbol W(1) is stored as the contents of W(0).

IA IA inserts symbol W(1) after the symbol in cell W(0). It first creates a new cell of the type of W(0). Then it stores the link of W(0) as the link of the new cell and stores the address of the new cell in the link of W(0). Finally the symbol W(1) is stored as the contents of the new cell.

LNNT LNNT searches the current name tables for symbol W(0) by calling LNNTW to search for W(0) in particular name tables from W stack WNT. It starts with the top name table in WNT (WNT.S) and will continue to make calls on LNNTW for successive name tables from the WNT stack until either the symbol is located, or all the name tables in WNT have been searched in vain. In the former case, the address of the name cell in the name table for the located symbol is output; in the latter case, the output is NIL.

LNNTW LNNTW searches backwards through the entries of name table W(0) for one with the symbol W(1). By searching backwards, LNNTW will find the most recent entry for the symbol W(1) if more than one exist. If the search is successful, LNNTW outputs the location of the name cell found (i.e. the cell containing the packed ASCII characters of the external name). If the search is unsuccessful, LNNTW outputs NIL.

LSNT LSNT is directly analogous to LNNT, except that it has a name cell as input W(0) and is searching for a corresponding symbol, rather than vice-versa.

LSNTW LSNTW is directly analogous to LNNTW, except that its input W(1) is a name cell and it searches for the symbol with that name in name table W(0), rather than searching for the name given the symbol as LSNTW does.

MVPTR MVPTR is an operation on PDP10 byte pointers (which are just T/C initially in L*), to be used in conjunction with CPTR and the depositing interpreters (.IDP and .IPDP) and extracting interpreters (.IEX and .IPEX). The input cell W(0) is the byte pointer; the W(1) input is an integer which designates the number of bit positions (within the current word) the pointer is to be moved (positive for right, negative for left). There is no primitive process in the L* kernel for moving a byte pointer a number of words, but this may be accomplished by operating on the right half of the pointer (which corresponds to the address field for byte pointers) with integer-symbol conversion and integer processes.

MVPTR operates by subtracting the value of W(1) from the value of the P field of the byte pointer W(0). It checks for one special case: if the P value comes out negative, it is zeroed instead.

MVPTR has no output.

N N outputs the next of cell W(0) (W(0).N).

NOP No operation.

P P Pushes W.

P01 P01 is the prefix routine for processes with no inputs and 1 output. It has a nonstandard input in R6 which is the location of the stem of the calling process. (The stem is the central machine code portion of the process divorced from special input-output considerations. It accepts inputs and returns outputs in registers). P01 operates by first calling the process stem as a subroutine, then upon return it pushes the output in R1 into W and returns to the caller of the process.

P10 P10 is the prefix routine for processes with 1 input and no outputs. It operates by first popping W(0) into input register R1, then passes control to the process stem (input to P10 in R6), which will itself return to the caller of the process.

P11 P11 is the prefix routine for processes with 1 input and 1 output. It operates by copying W(0) into input register R1, calling the process stem (input to P11 in R6), and upon return copying the output from register R1 into W and returning to the caller of the process.

P12 P12 is the prefix routine for processes with 1 input and 2 outputs. It operates by first copying W(0) into input register R1 and calling the process stem (input to P12 in R6). Upon return it copies the output from register R2 into W, pushes the output from register R1 onto W, and then returns to the caller of the process.

P20 P20 is the prefix routine for processes with 2 inputs and no outputs. It operates by popping W(0) into input register R1, W(1) into input register R2, and then passing control to the process stem (input to P20 in R6), which itself returns to the caller of the process.

P21 P21 is the prefix routine for processes with 2 inputs and 1 output. It operates by first popping W(0) into input register R2, and calling the process stem (input to P21 in R6). Upon return, P21 copies the output from register R1 into W and returns to the caller of the process.

P22 P22 is the prefix routine for processes with 2 inputs and 2 outputs. It operates by first copying W(0) into input register R1, W(1) into input register R2 and W(0) with the output from register R1, and calling the process stem (input to P22 in R6). Upon return, P22 replaces W(1) with the output from register R2, and returns to the caller of the process.

P31 P31 is the prefix routine for processes with 3 inputs and 1 output. It operates by first popping W(0) into input register R1, popping W again into register R2 to get the W(1) input, and then copying the W(2) input from W into register R3. P31 then calls the process stem (input to P31 in R6), and upon return copies the output from register R1 into W and returns to the caller of the process.

PCX PCX (Push ConteXt) pushes every other symbol in the input list W(0) starting with the second.

If the input list W(0) is (A1 B1 ... An Bn) , each Bk is operated on as in the program : (Bk S Bk I) . The Bk are normally thought of as cells whose contents specify current context in some way, hence PCX is a process which saves the current context prior to changing to a new context.

R R replaces the contents of W(0) by the symbol W(1).

RC RC (Replace Cell) replaces the full-word contents of W(0) by the full-word contents of W(1).

RCX The input to RCX (Replace ConteXt) should be a list of pairs : (A1 B1 ... An Bn) . Each pair is operated on as in the program : (Ak S Bk R) , i.e. the contents of each Bk is replaced by the contents of the corresponding Ak.

The Bk are normally thought of as cells whose contents specify current context in some manner, hence RCX is one of the basic context-changing mechanisms in the system (see also SWPCX).

RD Reads characters from interface W(0) and produces a list of the type of WTCKL.S which it outputs. It opens the interface and selects a file for the input if necessary.

As each character is read in from the buffer, RD adds the base for the characters to the character code to obtain a character symbol. (Null characters (code=0) are ignored). It then finds the type of list to be created from W cell WTCKL and calls C to create a cell. The symbol is put into the new cell and the new cell is linked to the rest of the list.

Characters are read until the current "break" character (in W cell WRDKB, initially KLF) is encountered. At this point readiing is terminated and the "next" of the last list cell is set to NIL. The created list (which contains the "break" character as its last symbol) is output W(0).

RI RI (Replace Integer) is identical to RC .

RN RN replaces the next of cell W(0) by the symbol W(1).

RSIF RSIF resets an I/O interface and will be used most often in the following situations:

(1) To reset interfaces closed by the monitor when a SAVE was done.

(2) To reset an interface that has gotten an end-of-file indication and is now to be reused (EXEC does this).

(3) To reset the DSK interface when a new file is to be read or written (see RDF, WRF in the Bootstrap Process Descriptions).

The operation of RSIF is as follows:

First the three flag bits (OPEN done, ENTER done, LOOKUP done) in the left half of the first word of the interface block are set off, the project, programmer numbers are zeroed (indicating user's own are to be used), the channel number used for the interface is RELEASed (thus ensuring that a file previously open on the interface is now closed), and finally both input and output buffers for the interface are reset using RSIFB.

RSIFB The input to RSIFB is the address of a three word block called a buffer header (input or output). (The input and output buffer header addresses are contained in the interface block. TTY and DSK are the two interface blocks defined in the kernel). RSIFB sets the use bit on in the buffer header (high order bit of first word), and then calls RSIFR to reset the buffer ring whose address is contained in the right half of the first word of the buffer header.

RSIFR The input to RSIFR is the address of the second word of one buffer in a ring of buffers. (I.e. a circular list of buffers. The right half of the second word of each buffer is a link to the next buffer in the ring). Each buffer in the ring is reset by zeroing its flag bit, which is the high order bit of the second word in the buffer.

RT RT (Replace Type) takes as input a symbol W(0) and a type index as the value of W(1) . It sets the type index of W(0) to the value in W(1) by replacing the contents of the cell whose address is W(0) + TD by the low order half of the value of W(1).

S S outputs the symbol of cell W(0) (W(0).S).

ST140 ST140 is the entry point at which L* is entered when a "START 140" command is issued in monitor mode. Entry at ST140 causes a recursive call on EXEC; exiting from this call on EXEC returns one to monitor mode. If then "CONTINUE" is typed, control returns to the caller of the routine which caused the original entry into monitor mode (i.e. before the "START 140"). Normally, this routine which caused the original entry into monitor mode will be HALT.

ST141 ST141 is the point at which L* is entered when a "START 141" command is issued in monitor mode. Entry at ST141 causes the following to happen:

The contents of working registers R1 - R5 are copied into cells R1SV - R5SV, and MSTKP is copied into cell MSPSV. Then reserved machine stack space is opened up (i.e. the effective size of the machine stack is increased from MSTKN to MSTKM), and DEBUG is called. Upon return from DEBUG, the machine stack space is closed off again (i.e. the effective size reduced from MSTKM back to MSTKN), and monitor mode is entered. If then "CONTINUE" is typed, control returns to the caller of the routine which caused the original entry into monitor mode (before the "START 141").

Note that due to an oversight, changing the integers MSTKN amd MSTKM will not affect the way ST141 manipulates the machine stack pointer since ST141 obtains the size of reserved stack space from a source other than MSTKN and MSTKM.

ST142 ST142 is an entry point to the middle of the SV routine which is meant to be used when saving for restart to reenter L* after the monitor SAVE command has been completed. Issuing a "START 142" outside of an execution of SV will result in an unpredictable context switch since the register contents are clobbered.

SV SV does the set-up work to allow the monitor SAVE command to be used, then does the necessary cleanup to continue after the SAVE is done.

It first saves the registers (NIL, R1 - R6, WPTR, WIPTT, WITT, W, WXS, WXN, WHS, WHN, MSTKP) and the first eight words of the high segment (the "Vestigial Job Data Area", clobbered by the monitor), and then goes into monitor mode. At this point the user is expected to issue a SAVE command (e.g. "SAVE DSK LSFB") and then reenter L* by the monitor command "START 142". The reentry point is inside SV where a SETUWP UUO is issued to reallow writing in the high segment, the first eight words in the high segment are restored from their save area, an APRENB UUO is issued to reenable central processor traps, the PC (Program Counter) flags are reset with a "JFCL 17,.+1" instruction, the registers are restored from their save area, and control returns to the caller of SV with the output TRUE in W(0) to indicate execution is continuing after a save.

If the saved files are RUN at some later time (e.g. "RUN DSK LSFB"), the same cleanup occurs as above after the "START 142", except that the output in W(0) is NIL to indicate a saved program is being restarted.

One of the side effects of the monitor SAVE command not handled by SV's cleanup is that all the I/O interfaces currently in use are closed. The L* program must reset those interfaces (with RSTF) before attempting to use them again.

SWPCX SWPCX (SWaP ConteXt) expects a list of pairs (A1 B1 ... An Bn) as input W(^) and exchanges the full-word contents of each Ak with the full-word contents of the corresponding Bk. SWPCX is used instead of PCX, RCX and UCX in cases where full-word contents must be changed and where context changes with respect to the particular context cells (the Ak's and Bk's) are not potentially recursive.

T T outputs the characteristic symbol for input W(0), which is the entry for W(^) in the type table in W cell WTTT.

TI TI outputs the type index of symbol W(1) as the value of W(0). The type index is found as the contents of the cell whose address is W(0) + TD .

U U (Up (pop)) pops W (i.e. it removes W(0)).

UCX UCX (Up (pop) ConteXt) is the inverse of PCX. It pops every other symbol in the list W(0) starting with the second.

If the input list W(0) is (A1 B1 ... An Bn) , each Bk is operated on as in the program : (Bk D) . The Bk are normally cells whose contents specify current context in some way, hence UCX has the effect of restoring some previously pushed context.

V V reverses W(^) and W(1) in W.

WR WR writes the list in W(1) to the interface W(0), opening the interface and selecting a file for the output if necessary. Writing to the interface is done by interpreting W(1) in Write Context (via .XCX) .

The interpreters .IWR and .IPWR do the actual work of depositing characters into the output buffers and writing them out when they become filled. WR does one last output operation to write out the last partially filled buffer when control returns from interpretation of W(1).

.IPTT Standard interpreter type table for execution (interpretation) of symbols appearing within T/P lists. Initially contains the following entries : .IP/M for T/M , .IP/P for T/P , and .IP/S for T/C, T/I, T/K and T/L .

.ITT Standard interpreter type table for execution outside of T/P lists (e.g., by .X and .XCY and from T/M routines). Initially contains the following entries : .I/M for T/M , .I/P for T/P , and .I/S for T/C, T/I, T/K and T/L .

AKT1 The character action table which is initially in W cell WAKT . It initially contains the following character actions :

ABND	for	KCR KLF KVT KFF KCR KSP
.X	for	!
ANK	for	" # $ % & ( ) * , . /
		: < = > ? @ [\ ] ↑ ←
		all upper and lower case letters
.Q	for	'
ADK	for	0 1 2 3 4 5 6 7 8 9
A+K	for	+
A-K	for	-
.	for	;
NOP	for	all others

ARPTT Interpreter type table to be used when in Assembly Read context to interpret symbols occurring within a program list. Its initial entries are : .IPEX for T/K, .IP/M for T/M, .IP/P for T/P, and .IP/S for T/C, T/I and T/L .

ARTT Interpreter type table to be used when in Assembly Read context to interpret symbols not occurring within a program list. Its initial entries are : .IEX for T/K, .I/M for T/M, .I/P for T/P, and .I/S for T/C, T/I and T/L .

AWPTT Interpreter type table to be used when in Assembly Write context to interpret symbols occurring within a program list. Its initial entries are : .IPDP for T/K, .IP/M for T/M, .IP/P for T/P, and .IP/S for T/C, T/I and T/L .

AWTT Interpreter type table to be used when in Assembly Write context to interpret symbols not occurring within a program list. Its initial entries are : .IDP for T/K, .I/M for

T/M, .T/P for T/P, and .I/S for T/C, T/I and T/L .

B/K Integer whose value is the null character symbol (T/K) which is the base of the 128 cell block of character symbols. Appears as the entry for T/K in the base type table BTT .

BTT Initial current base type table in W cell WBTT. Its only initial entry is integer B/K for T/K . The current bases are accessed via WBTT by .IDP , .IPDP , .IEX and .IPEX . The current character base is accessed via WBTT by .IWR , .IPWR and RD .

DECML An integer (T/I) with value = decimal ten. Not initially used anywhere in kernel, but intended to be used to change current integer radix in W cell WIB to decimal.

DSK Interface block for reading and writing the disk. Uses two 200 octal word buffers for both input and output. Initially set to read from file "BOOT.LSF" and write to file "FILE.LSF" . Uses channel 1 .

INUM Integer accumulator used by ADK (the digit character action) to accumulate a value (via process ACCD) as digit characters are being interpreted. Also referenced by ABND (the boundary action) to actually create an integer (when appropriate), and to clear the integer value.

INUMF Integer flag used by ABND, ANK, ADK, A+K and A-K to distinguish between integers and names being accumulated. Cleared by ABND .

TSGN Integer flag used by A-K and ABND to record whether an integer is positive or negative when one occurs. Cleared by ABND .

JBAPR PDP10 Job Data Area location (JOBAPR) which contains trap location for central processor interrupts. Initially set by L* to contain ERR1 .

JBCNI Job Data Area location JOBCNI. Contains state of arithmetic processor as stored by CONI APR when an enabled trap occurs. (See process description for ERR1).

JBCOR Job Data Area location JOBCOR. Left Half contains highest location in low segment with non-zero data (set by LOADER). Right Half contains user argument on last SAVE or GET command (set by Monitor). Not referenced by kernel.

JBFF Job Data Area location JOBFF. Right Half contains address of first free location following the low segment. Maintained by CSP to point to the top of core in the low segment so that the SAVE command will work correctly.

JBHRL Job Data Area location JOBHRL. Left Half contains first free location in high segment relative to high segment origin. Right Half contains highest legal user address in the high segment. Left Half is updated by CSP when additional core is obtained so that the SAVE command will work correctly. Right Half is used by CSP to locate the current top of the high segment when additional core is to be obtained.

JBOPC Job Data Area location JOBOPC. Used by monitor to store previous contents of the user's program counter when a DDT, REENTER, START or CSTART command is issued.

JBREL Job Data Area location JOBREL. Contains highest low segment core address available to the user.

JBREN Job Data Area location JOBREN. Contains starting address used by REENTER command. Can be set by user to provide an alternate entry point.

JBSA Job Data Area location JOBSA. Left Half contains first free location in low segment. Right Half contains starting address of user's program. Left Half is updated by CSP to ensure that the SAVE command will work correctly. Right Half is set by L* to start execution at the proper location within the process SV so that saved segments will continue when they are RUN .

JBTPC Job Data Area location JOBTPC. Where Monitor stores program counter of next instruction to be executed when an enabled central processor trap occurs.

KALT Altmode character. Code = 175 octal.

KBELL Bell character. Code = 007 octal.

KBSP Backspace character. Code = 010 octal.

KCR Carriage return character. Code = 015 octal.

KFF Form feed character. Code = 014 octal.

KLF Line feed character. Code = 012 octal.

KSP Space character (blank). Code = 040 octal.

KTAB Horizontal tab character. Code = 011 octal.

KTN Integer whose value is the size of character tables (initially 128 decimal). Not referenced by kernel.

KVT Vertical tab character. Code = 013 octal.

MSPSV Cell used by ST141 to read out and by ERROR to read out and restore the contents of the machine stack pointer MSTKP .

MSTK Contiguous block of cells of length MSTKM appearing in the kernel immediately before initial T/C available space. Used throughout the kernel for T/M routine linkage and saving of register contents over machine code subroutine calls.

MSTKM Integer whose value is the actual maximum size of the machine stack MSTK .

MSTKN Integer whose value is the stack size used in the machine stack pointer MSTKP under normal conditions. When an attempt is made to push more than MSTKN entries onto the stack, a pushdown overflow error trap occurs (see process description for ERR1). ST141 and ERROR increase the operating stack size in MSTKP from MSTKN to MSTKM over the scope of their execution to provide reserved stack space for temporary use.

MSTKP Register (17 octal) containing the PDP10 stack pointer for the machine stack MSTK. The Left Half contains the negative count of unused words left in the stack, the Right Half contains the address of the current top entry on the stack.

N/C Integer whose value is the count of cells on initial T/C available space.

N/I Integer whose value is the count of cells on initial T/I available space.

N/L Integer whose value is the count of cells on initial T/L available space (not counting reserved T/L space).

N/M Integer whose value is the count of cells on initial T/M available space.

N/P Integer whose value is the count of cells on initial T/P available space.

N/RL Integer whose value is the count of cells on initial reserved T/L space (in W cell WSPRL).

NACC Cell used by ACCK to accumulate characters being interpreted into packed form for use by ABND if a name is to be looked up or entered into the name table. ABND also clears NACC before exiting.

NBTT Initial current number-of-bits type table in W cell WNBTT . Its initial entries are SEVEN for T/K and ZERO for all other types. The bit sizes for each type are used via WNBTT by the Deposit and Extract interpreters .IDP , .IPDP , .IEX and .IPEX .

NIL Special T/L symbol used throughout the kernel as the list terminator and as the negative signal from tests. NIL happens to be the symbol 0 (register zero), but this is mainly for convenience.

NT1 The initial name table in W cell WNT which contains all the names listed in Appendices 3 and 4 . The name table is a contiguous block of cells of length twice the value of integer NT1N . Each name entry is two cells long and contains the right-justified packed ASCII characters of the external name in the first cell, and the corresponding internal symbol in the right half of the second word. NT1I , which is an integer whose value gives the current number of entries in the table, is assumed to be the cell immediately preceding the first cell of the table itself (NT1). NT1N , which is an integer whose constant value gives the maximum number of entries the table will hold, is assumed to occupy the cell immediately preceding NT1I . The current name table is accessed via W cell WNT by the kernel processes LSNT , LNNT and CSNT .

NT1I Integer whose value specifies the current number of entries in name table NT1 . Used to locate the current last entry in the table for searching and making new entries . Occupies cell immediately preceding NT1 .

NT1N Integer whose value specifies the maximum number of entries name table NT1 will hold. Compared with NT1I when new entries are being made to detect overflow of NT1 . Occupies cell immediately preceding NT1I .

OCTAL Integer (T/I) with value = decimal eight. Used as initial contents of W cell WIR to indicate octal integer radix .

R1 Register 1 . Used in the kernel as an input-output register for machine code subroutine calls, and as a work register .

R1SV Cell used by ST141 to read out and by ERROR to read out and restore the contents of R1 .

R2 Register 2 . Used in the kernel as a second input register for machine code subroutine calls, and as a work register .

R2SV Cell used by ST141 to read out and by ERROR to read out and restore the contents of R2 .

R3 Register 3 . Used in the kernel as a third input register for machine code subroutine calls, and as a work register.

R3SV Cell used by ST141 to read out and by ERROR to read out and restore the contents of R3 .

R4 Register 4 . Used in the kernel as a work register .

R4SV Cell used by ST141 to read out and by ERROR to read out and restore the contents of R4 .

R5 Register 5 . Used in the kernel as a work register .

R5SV Cell used by ST141 to read out and by ERROR to read out and restore the contents of R5 .

R6 Register 6 . Used in the kernel as a work register, by the error locations ERR? - ERR22 to transmit to the common error routine ERROR the identity of the particular error, and by machine process prefixes to transmit the location of the process stem to the prefix subroutine (P?1, etc.).

RDCX Context list used by EXEC as input to .XCX when executing the character list obtained from RD . RDCX is defined as : ((RDTT) WITT (RDPTT) WIPTT) , which causes the current interpreter type tables to become RDTT and RDPTT over the scope of the execution of the character list.

RDPTT Interpreter type table to be used when in Read Context to interpret symbols occurring within program lists. Its initial entries are : .IP/K for T/K , .IP/M for T/M , .IP/P for T/P , and .IP/S for T/C, T/I, and T/L . The context list RDCX , when used as input to .XCX , will cause interpretation to occur in Read Context (i.e., using RDPTT and RDTT).

RDTT Interpreter type table to be used when in Read Context to interpret symbols not occurring within a program list. Its initial entries are : .I/K for T/K , .I/M for T/M , .I/P for T/P , and .I/S for T/C, T/I and T/L . The context list RDCX , when used as input to .XCX , will cause interpretation to occur in Read Context (i.e., using RDTT and RDPTT).

SEVEN Integer (T/I) constant with value = 7 . Used as initial entry for T/K in type table NBTT .

SP/C Initial available space list for T/C . Appears as initial entry for T/C in type table SPTT .

SP/I Initial available space list for T/I . Appears as initial entry for T/I in type table SPTT .

SP/L Initial available space list for T/L . Appears as initial entry for T/L in type table SPTT .

SP/M Initial available space list for T/M . Appears as initial entry for T/M in type table SPTT .

SP/P Initial available space list for T/P . Appears as initial entry for T/P in type table SPTT .

SP/RL Initial reserved available space list for T/L . Appears as initial contents of W cell WSPRL .

SPTT Initial available space type table in W cell WSPTT . Its initial entries are : SP/C for T/C , SP/I for T/I , SP/L for T/L , SP/M for T/M , and SP/P for T/P .

SPXCX Context list used by C and C/L to execute space-exhausted routines in Space-Exhausted Context. SPXCX is defined as : ((.ITT) WITT (.IPTT) WIPTT) , which means that the standard interpreter type tables .ITT and .IPTT are used in Space-Exhausted Context.

SPXTT Initial space-exhausted routine type table in W cell WSPXT . SPXTT has no entries initially ; it is the responsibility of the L* bootstrap to define space-exhausted routines and put them into SPXTT before initial available space of any type is exhausted.

STOP T/L symbol used to mark the level in the higher routine stack WHN where a T/P symbol occurring outside of another T/P list was interpreted. Each STOP mark in WHN parallels a return link in the machine stack MSTK to the point where a T/P symbol was interpreted from a machine code routine. .I/P causes the STOP to be pushed onto WHN when first called, then watches for the STOP each time it Ascends and exits when the STOP reappears. Interpretation of T/P symbols within other T/P lists is done by .IP/P , a closed subroutine which causes a Descend and returns to .I/P .

T/C Entry for Type Cell in the initial characteristic symbol type table TTT . Represents a null symbol of Type Cell ; its initial full-word contents are zero . Can be used where some arbitrary symbol of Type Cell is needed, or for creating null Type Cell symbols with process C (as in CPTR).

T/I Entry for Type Integer in the initial characteristic symbol type table TTT . Represents a null symbol of Type Integer ; its initial value is zero . Can be used where some arbitrary symbol of Type Integer is needed, or for creating null Type Integer symbols with process C (as in ABND when an integer is recognized).

T/K Entry for Type Character in the initial characteristic symbol type table TTT . Also symbol for null character, and base symbol for characters. Not used for creating Type Character symbols since that is normally not allowed.

T/L Entry for Type List in the initial characteristic symbol type table TTT . Represents a null symbol of Type List ; its initial symbol and next (T/L.S and T/L.N) are both = NIL . Can be used when an arbitrary symbol of Type List is needed, or for creating null Type List symbols with process C . (Note that process C/L does not create null Type List symbols since it doesn't initialize the cells it outputs).

T/M Entry for Type Machine Code in the initial characteristic symbol type table TTT . Represents a null symbol of Type Machine Code ; its initial contents (full-word) are a RETURN (POPJ MSTKP,) instruction. Can be used where an arbitrary Type Machine Code symbol is needed, or possibly for creating null Type Machine Code symbols with process C .

T/P Entry for Type Program List in the initial characteristic symbol type table TTT . Represents a null Program List ; its initial contents are T/P.S = NOP and T/P.N = NIL (i.e., T/P : (NOP)). Used as initial contents of W cells WTC and WTCKL . Can be used when an arbitrary Type Program List symbol is needed, or for creating null program lists with process C .

TD Integer whose value is the Type Displacement, which is the displacement from a symbol to the symbol-description word for that symbol (i.e., the word holding the symbol's Type Index). The value used for L*(T) is 400000 octal, which puts all the symbol-description words into the high segment provided by the PDP10 Monitor. Changing the value of TD will not effectively change the Type Displacement since it is assembled into machine code instructions throughout the kernel.

TRUE T/L symbol output as a positive result from kernel test processes when the W(1) input was NIL and merely leaving the W(1) input as output would result in confusion. The processes which do this are : =S , <S , >S , =T , =C , =I , <I and >I . TRUE is also output by SV when continuing just after a SAVE has been done.

TTN Integer whose value is the size of existing type tables (number of cells) , which is also the maximum number of types allowed. The value does not control any processing in the kernel (e.g., no checks are made when accessing type tables to see if an index > the value of TTN is being used) ; it is only for information.

TTT Initial characteristic symbol type table in W cell WTTT . Holds null symbols of each type, initially as follows : T/C for Type Cell, T/I for Type Integer, T/K for Type Character, T/L for Type List, T/M for Type Machine Code, and T/P for Type Program List. Used via WTTT by kernel process T .

TTY Interface block for reading and writing the user's teletype. Uses two 20 octal word buffers for both input and output. Operates on channel 2 in ASCII Line mode.

W T/L cell used to communicate inputs and outputs between successively interpreted processes. The prefix subroutines (P01 - P31) handle the transfer of inputs from W to rege List ind outputs from registers back to W for calls on machine code processes. The processes .T/S , .TP/S , ABND , .IEX and .IPEX are all processes which don't use the standard prefixes and thus push their outputs directly onto W .

WAKT W cell which holds current character action table (initially AKT1) used by .I/K and .IP/K .

WBTT W cell which holds current base type table (initially BTT) used by .IDP , .IPDP , .IEX , .IPEX , .IWR , .IPWR and RD .

WDB W cell which holds Debug routine (initially EXEC) executed by DEBUG .

WDBCX W cell for holding current Debug Context Swap List used as input to SWPCX by DEBUG to swap contexts before and after executing the Debug routine in WDB, and by ERROR to swap contexts before and after executing the error routine in WERR . WDBCX is empty in the kernel ; the L* bootstrap is responsible for setting up a swap list and putting it into WDBCX .

WERR W cell for holding the current general error handling routine executed in Debug Context by ERROR . Initially holds HALT .

WERRL W cell set by ERROR to hold address of particular error location which made call to ERROR . Used to identify nature of error when one occurs.

WHN W cell (register 15 octal) used in L*L interpretation as stack to hold address of next cell in program list to be interpreted at each higher level. When a Descend occurs (as in .I/P and .IP/P) the current next program contained in W cell WXN is pushed onto WHN to preserve it. When an Ascend occurs (as in .Q and .I/P) the contents of WHN is popped into WXN . Setting the contents of WHN to NIL (i.e., the contents of the top cell) has the effect of terminating execution of the next higher program list. This fact is used by .. , ..+ and ..- .

WHS W cell (register 16 octal) used in L*L interpretation as stack to hold addresses of higher level programs being interpreted. When a Descend occurs (as in .I/P and .IP/P) the current program being interpreted contained in W cell WXS is pushed onto WHS to preserve it. When an Ascend occurs (as in .Q and .I/P) the contents of WHS is popped into WXS . .R , .R+ and .R- work by copying the contents of WHS into WXN , thus making the higher level program next at the current level.

WIB W cell which holds integer whose value is the current radix for integers. Used by processes ACCD and CVIDL .

WIPTT W cell (register 12 octal) which holds current interpreter type table for symbols occurring within T/P lists. Initially contains .IPTT . Changes of interpretation context are effected by changing the contents of WIPTT (and WITT). Referenced in the kernel by .I/P .

WITT W cell (register 11 octal) which holds current interpreter type table for interpretation of symbols not occurring within T/P lists. Initially contains .ITT . Changes of interpretation context are effected by changing the contents of WITT (and WIPTT). Referenced in the kernel by DEBUG , ERROR , C , C/L , .X , .XCX , .I/K and .IP/K .

WK W cell set by .I/K and .IP/K to contain the current character being interpreted. Used by character action routines to get the character they are interpreting, (e.g., by ANK , ADK , A+K and A-K).

WNBTT W cell which holds current number of bits type table (initially NBTT). Used by .IDP , .IPDP , .IEX and .IPEX .

WNT W cell for stack of current name tables. Initially contains only NT1 . LSNT will search each name table in the stack starting with the top until it locates its input symbol or has searched all name tables in vain. LNNT searches similarly trying to locate its input name. CSNT creates an entry for its input name in the name table in the top of the WNT stack (i.e., the contents of cell WNT).

WPTR W cell (register 7) to hold the address of a PDP10 byte pointer used for depositing and extracting bit patterns. When .IDP , .IPDP , .IEX and .IPEX use WPTR they assume that it contains a byte pointer which points to the field to be operated upon. WPTR is initially empty, but pointers can be created by CPTR and then stored into WPTR for use in depositing and extracting. Byte pointers can also be moved a number of bits within the current word pointed to by MVPTR .

WRCX Context list used by WR as input to .XCX when executing the list input to WR as W(1). WRCX is defined as : ((WRTT) WITT (WRPTT) WIPTT) , which causes WRTT and WRPTT to become the current interpreter type tables over the execution of the list being written.

WRD W cell which holds current read interface (initially TTY). Used by EXEC to obtain the interface to be read from (i.e., the interface to be the W(0) input to RD). EXEC also resets (via RSIF) the current interface in WRD when an end-of-file is detected.

WRDBK W cell containing current read break character (initially KLF). Used by RD to determine when to stop reading characters from the actual external interface and return with its output character list. RD will continue to read characters until it encounters one that is the same as the one currently in WRDBK ; thus, the last character on the list output by RD will always be the current break character, and will be the only occurrence of the break character on the list.

WRPTT Interpreter type table to be used when in Write Context to interpret symbols occurring within program lists. Its initial entries are : .IPWR for T/K , .IP/M for T/M , .IP/P for T/P , and .IP/S for T/C , T/I and T/L . The context list WRCX , when used as input to .XCX , will cause interpretation to occur in Write Context (i.e., using WRPTT and WRTT).

WRTT Interpreter type table to be used when in Write Context to interpret symbols not occurring within a program list. Its initial entries are : .IWR for T/K , .I/M for T/M , .I/P for T/P , and .I/S for T/C , T/T and T/L . The context list WRCX , when used as input to .XCX , will cause interpretation to occur in Write Context (i.e., using WRTT and WRPTT).

WSPRL W cell to hold the reserved available space list for T/L (necessary since execution of space-exhausted routines requires some T/L space as working space before additional space can be obtained from the monitor). When C or C/L detects that T/L space has been exhausted, it will make the reserved space list from WSPRL the current available space list for T/L in the type table in WSPTT before calling the space-exhausted process from the type table in WSPXT . The space-exhausted process is given the responsibility of building a fresh reserved available space list and storing it into WSPRL .

WSPTT W cell for current available space type table (initially contains SPTT). Used by C and C/L for obtaining cells from available space lists, and by E and E/L for returning cells.

WSPXT W cell for current space-exhausted process type table (initially contains SPXTT). Used by C and C/L to obtain the current space-exhausted process when available space of some type is exhausted.

WTC W cell which specifies current type being created (initially contains T/P). Used by CSNTW which is called by CSNT , which is called by ABND when a name has come across the input interface which isn't defined in the current name tables. CSNTW creates a new symbol of the same type as the symbol currently in WTC to go with the new name it enters into the name table.

WTCKL W cell which contains the type to be used for creating character lists (initially contains T/P). Used by RD , CVNKL and CVIDL .

WTTT W cell which holds the current characteristic symbol type table (initially TTT). Used by process T .

WWR W cell which holds current output interfaces (to be treated as a stack of interfaces, all of which would receive output). Not referenced in kernel, but to be used by print routines defined in bootstrap.

WXN W cell (register 14 octal) which holds next operation at current level during L*L interpretation. When an Advance occurs, the contents of WXN is replaced by the link of the cell pointed to by the original contents. During a Descend, WXN is pushed onto WHN ; during an Ascend, WHN is popped into WXN . In .I/P , WXN = NIL when attempting to Advance signals the end of the current level and triggers an Ascend; WXN = STOP when attempting an Advance triggers an Ascend followed by a return to the caller of .I/P . Setting WXN = NIL has the effect of terminating interpretation of the current level; this fact is used by the control operations . , .+ , .- , .. , ..+ and ..- . The repeat operations .R , .R+ and .R- operate by copying the current contents of WHS (the higher routine cell) into WXN .

WXS W cell (register 13 octal) which holds current symbol being interpreted. During Advance, before WXN is stepped ahead, WXS gets the symbol of the cell pointed to by the contents of WXN (i.e., the next symbol to be interpreted). During a Descend, WXS is pushed onto WHS ; during an Ascend, WHS is popped into WXS . Interpreters for symbols occurring within program lists (.IP/K , .IP/M , .IP/P , .IP/S , .IPDP , .IPEX and .IPWR) all get the symbol to be interpreted as input from WXS . (The remaining interpreters receive the symbol to be interpreted in R1).

ZERO T/I constant with value = 0. Used as initial entry in type tables BTT and NBTT for all types except T/K .

(1) TO RUN THE VERSION OF L*(F) WHICH INCLUDES THE BOOTSTRAP, DO (IN COMMAND MODE):

```
R LSFA
```

THE SYSTEM WILL RESPOND WITH "VXX RESTARTED" AND PUT YOU IN CONTROL BY READING FROM THE TTY .

(2) TO GET A COPY OF THE L*(F) KERNEL MACRO-10 LISTING, USE THE FOLLOWING PIP COMMAND:

```
LPT:←DSK:LSF.LST[167,77374]
```

(3) TO GET COPIES OF THE BOOTSTRAP FILE, ON-LINE EDITOR FILE AND STEPPING MONITOR FILE INTO YOUR DSK AREA SO THAT YOU CAN RUN THROUGH THE BOOTSTRAP, USE THE FOLLOWING PIP COMMAND:

```
DSK:/X←SYS:BOOT.LSF,EDITF.LSF,STPMF.LSF
```

(4) TO RUN THROUGH THE LOADING OF THE BOOTSTRAP DO:

```
R LSF

DSK WRD R !
```

THE SYSTEM WILL RESPOND WITH "INITIAL BOOTSTRAP LOADED". IF YOU THEN DO IN L* THE FOLLOWING:

```
EDITF RDF!

STPMF RDF!
```

THIS WILL BRING YOU TO THE SAME POINT WHERE YOU WOULD BE AFTER RUNNING LSFA .

(5) TO DO A SAVE FOR RESTART (ONLY IF YOU HAVE THE BOOTSTRAP ROUTINES LOADED) DO:

```
SAVE!
```

THIS WILL PUT YOU INTO MONITOR MODE. NOW DO:

SAVE DSK <FILE NAME>

THIS WILL CREATE THE TWO FILES <FILE NAME>.LOW AND <FILE NAME>.HGH WHICH CONSTITUTE A SAVED VERSION OF YOUR L* SYSTEM. NOW DO THE MONITOR COMMAND:

START 142

THE L* SYSTEM (I.E. THE PROCESS CALLED "SAVE") WILL RESPOND WITH "VXX CONTINUING", AND YOU ARE BACK IN L*.

AT SOME LATER TIME YOU MAY RESTART YOUR SAVED L* SYSTEM BY ISSUING THE MONITOR COMMAND:

RUN DSK <FILE NAME>

WHERE <FILE NAME> IS OF COURSE THE SAME NAME YOU USED WHEN YOU DID THE "SAVE" COMMAND. THE SYSTEM WILL RESPOND WITH "VXX RESTARTED" AND CONTINUE WHERE YOU LEFT OFF. THIS IS IN FACT HOW THE SYSTEM LSFA IS CREATED: BY RUNNING LSF (THE BARE KERNEL), LOADING THE BOOTSTRAP, EDITOR AND STEPPING MONITOR, AND THEN SAVING IT WITH <FILE NAME> LSFA .

```
; INITIAL BOOTSTRAP - L*(F)


; DEFINE ROUTINE FOR REPLACING CHARACTER ACTION
; RCKA : (WAKT S .O T/K ITO -SS +IS P)

T/I WTC R !
ITO U ! ; DEFINE TEMP INTEGER CELL FOR BOOTSTRAP
T/P WTC R !
P RCKA R !  +IS RCKA I !  -SS RCKA I !  ITO RCKA I !  T/K RCKA I !
.O RCKA I !  S RCKA I !  WAKT RCKA I !


; ** DEFINE CHARACTER ACTIONS FOR  NAME : ( ... )  **

; DEFINE ACTION FOR  :
; USEN : (WUSEN S)

T/L WTC R !
T/L C ! WUSEN R !
T/P WTC R !
S USEN R !  WUSEN USEN I !
USEN ': RCKA !

; DEFINE ACTION FOR (
; STRL : (P WUSEN S =S STRL1 P N EL P NTL V RN WC I WFLR S)
; STRL1 : (STRL2 V)
; STRL2 : (.+ WTC S C ..)

T/L WTC R !
T/L C ! WFLR R !
WC U !

T/P WTC R !
S STRL R !  WFLR STRL I !  I STRL I !  WC STRL I !
RN STRL I !  V STRL I !  NTL STRL I !  P STRL I !
EL STRL I !  N STRL I !  P STRL I !
STRL1 STRL I !  =S STRL I !  S STRL I !  WUSEN STRL I !
P STRL I !

V STRL1 R !  STRL2 STRL1 I !

.. STRL2 R !  C STRL2 I !  S STRL2 I !  WTC STRL2 I !
.+ STRL2 I !

STRL '( RCKA !

; DEFINE ACTION FOR  )
; ENDL : (ENDL1 U WC S D P WUSEN S =S ENDL2 WC D)
; ENDL1 : (P WFLR S =S .+ WC S IA .R)
; ENDL2 : (ENDL3 U)
```

```
; ENDL3 : (.+ WC S ..)

D ENDL R !  WC ENDL I !  ENDL2 ENDL I !
=S ENDL I !  S ENDL I !  WUSEN ENDL I !
P ENDL I !  D ENDL I !  S ENDL I !
WC ENDL I !  U ENDL I !  ENDL1 ENDL I !

.R ENDL1 R !  TA ENDL1 I !  S ENDL1 I !
WC ENDL1 I !  .+ ENDL1 I !  =S ENDL1 I !
S ENDL1 I !  WELR ENDL1 I !  P ENDL1 I !

U ENDL2 R !  ENDL3 ENDL2 I !

.. ENDL3 R !  S ENDL3 I !  WC ENDL3 I !
.+ ENDL3 I !

ENDL ') RCKA !


; ADD BOUNDARY ACTION TO SOME SPECIAL CHARACTERS

T/L WTC R !
.ICX : ((.ITT ) WITT (.IPTT ) WIPTT )
T/P WTC R !
(ABND .ICX .XCX ) '! RCKA !
(ABND USEN ) ': RCKA!
(ABND STRL ) '( RCKA!
(ABND ENDL ) ') RCKA!


; SET UP DEBUG SWAP LIST TO FORCE READ FROM TTY

T/L WTC R!
DWRD: (TTY)
DWRDB: (KLF)
DWWR: (TTY)
DNIL: (NIL)
DWITT: (.ITT)
DWIPT: (.IPTT)
DBCX: (DWRD WRD DWRDB WRDBK DWWR WWR DNIL NIL DWITT WITT DWIPT WIPTT)
DBCX WDBCX R!


; DEFINE WORKING CELLS AND SAVING UTILITY RTNS

W0 U! W1 U! W2 U! W3 U! W4 U! W5 U!
T0 U! T1 U! T2 U! T3 U! T4 U! T5 U!
WSAVE U!
T/P WTC R!
SAVEW: (WSAVE I)
RSTRW: (WSAVE S WSAVE D)
```

```
; DEFINE TYPE DECLARATION ROUTINES

DEF/L: (T/L WTC R)
DEF/P: (.O T/P WTC R)
DEF/I: (T/I WTC R)
; DEFINE @ ACTION - A@T MAKES A OF TYPE T
DEF/L!
TYPL: ('I T/I 'L T/L 'P T/P 'M T/M 'C T/C)
DEF/P!
(WHN N S S WO I WHN N P S N V R   ; GET NEXT CHAR. AND ADVANCE
  TYPL (P S WO S =S .+ N N P .R+ HALT)  ; FIND CHAR. SYMBOL
  N S WTC I ABND P WTC S TTO TI V RT   ; MAKE SURE OF ITS TYPE
  WTC D WC D)
'@ RCKA!


; DEFINE " ACTION - "..." CREATES LIST OF CHARACTERS

STRKL: (WTCKL S C WC I WFLR S)
ENDKL: (ENDL1 U WC S P D WC D)

(ABND WHN N P SAVEW S    ; GET INPUT LIST
  STRKL V                ; START K-LIST
  (P S .O '" =S .+ P S V N P .R+)        ; TERMINATE ON " OR EOL
  N RSTRW R ENDKL)       ; ADVANCE BEYOND " AND END LIST
'" RCKA!


; DEFINE OUTPUT ROUTINES

WRWWR: (WO I WWR (P S WO S V WR N P .R+ U) WC D)
CVSI: (SAVEW T/I C P RSTRW V R)
PRN: (P LNNT P PRN1 P WRWWR EL)
PRN1: (PRN2 V U CVNKL)
PRN2: (.+ U CVSI P CVIDL V E ..)
PRI: (.O "(=" WRWWR CVIDL P WRWWR EL .O ")" WRWWR)
PRS: (P PRN P T/I =T (.- PRI ..) U)
PR: (P PRN .O ": " WRWWR PRSTR CR.LF)
PRSTR: (P T/I =T (.- PRI ..)
  P T/L =T (.- PRLS ..)
  P .O T/P =T (.- PRLS ..)
  P .O T/K =T (.- P LNNT P (.- U ..) U .O "'" WRWWR WRWWR ..)
  PRN)   ; PRINT NAME ONLY OF ALL OTHER TYPES
PRLS: (.O "(" WRWWR
  (P S PRSTX N P .- SPACE .R)
  U .O ")" WRWWR)
PRST1: ( P LNNT P (.- V U CVNKL P WRWWR EL ..) U PRSTR)
PRST1 PRSTX R!
PRL: (.O PRS .O PRSTX R PR .O PRST1 .O PRSTX R)
SPACE: (.O " " WRWWR)
CR.LF: (.O (KCR KLF) WRWWR)
```

```
; DEFINE TYPE TABLE AND CHARACTER ACTION TABLE PROCESSES

SETT:  (V IT0 TI +IS S)
RETT:  (V IT0 TI +IS R)
IETT:  (V IT0 TI +IS I)
DETT:  (V IT0 TI +IS D)

SCKA:  (WAKT S .Q T/K IT0 -SS +IS S)
ICKA:  (WAKT S .Q T/K IT0 -SS +IS I)
DCKA:  (WAKT S .Q T/K IT0 -SS +IS D)


; DEFINE ELEMENTARY SPACE PROCESSES

; CSPT - ADD 2000 CELLS TO AV.SP FOR TYPE W(0)
CSPT:  (P            ;SAVE TYPE SYMBOL
        2000 V       ; GET NO. OF CELLS
        CSP          ; GET CELLS OF CORRECT TYPE FROM MONITOR
        P            ; COPY START ADDR
        2000 V
        LNKUP        ; LINK UP THE 2000 CELLS
        V WSPTT S RETT)   ; PUT IN AV.SP TYPE TABLE

; LNKUP - LINK W(1) CELLS STARTING AT W(0) INTO A LIST
LNKUP: (P W0 I   ; SAVE START ADDR
        V +IS W1 I        ; SAVE END ADDR +1
          (W0 S           ; GET CURRENT CELL
          P 1 +IS P       ; GET NEXT CELL
          W1 S =S .+      ; EXIT IF END
          P W0 R          ; SAVE NEXT AS CURRENT
          V RN .R)        ; STORE NEXT AS LINK OF CURRENT
        U NIL V RN        ; LINK OF LAST CELL NIL
        W0 D W1 D)

CSP/P:  (.Q T/P CSPT)       ; SPXTT RTN FOR T/P
CSP/L:  (T/L CSPT T/L C P WSPRL R N/RL SPCLI0T RI
  (P NIL V I SPCLI -1 SPCLI +I 0 =I .R- U))      ; SPXTT RTN FOR T/L
CSP/M:  (.Q T/M CSPT)       ; SPXTT RTN FOR T/M
CSP/I:  (T/I CSPT)          ;SPXTT RTN FOR T/I
CSP/C:  (T/C CSPT)          ;SPXTT RTN FOR T/C

CSP/P T/P WSPXT S! RETT!          ; INSTALL RTNS IN CURRENT SPACE
CSP/L T/L WSPXT S! RETT!          ;    EXHAUSTED RTN TYPE TABLE
CSP/M T/M WSPXT S! RETT!
CSP/I T/I WSPXT S! RETT!
CSP/C T/C WSPXT S! RETT!


; DEFINE ASSEMBLY PROCESSES
```

```
DEF/L!
WARTT: (ARTT)    ; CURRENT ASSEMBLY READ TYPE TABLE
WARPT: (ARPTT)   ; CURRENT AR T/P TYPE TABLE
WAWTT: (AWTT)
WAWPT: (AWPTT)
ARCX: (WARTT WITT WARPT WIPTT)  ; CONTEXT LIST FOR AR
AWCX: (WAWTT WITT WAWPT WIPTT)  ; CONTEXT LIST FOR AW

DEF/P!
; AR - START AT W(0), EXECUTE LIST W(1) INTERPRETED WITH
;    ARTT AND ARPTT, THEN MAKE A LIST OF TYPE
;    WTCKL.S OF ALL THE EXTRACTED SYMBOLS.
AR: (CPTR WPTR I         ; CREATE PTR TO LOC W(0)
  STPKL V          ; START A LIST OF TYPE WTCKL.S
  ARCX .XCX        ; EXECUTE LIST W(1) A/C AR CONTEXT LIST
  ENDKL            ; BUILD THE LIST
  WPTR S WPTR D E)        ; POP WPTR AND ERASE CREATED POINTER

; AW - START AT W(0), EXECUTE LIST W(1) INTERPRETED WITH
;    AWTT AND AWPTT.
AW: (CPTR WPTR I         ; CREATE PRT TO LOC W(0)
  AWCX .XCX        ; EXECUTE LIST W(1) A/C AW CONTEXT LIST
  WPTR S WPTR D E)        ; POP WPTR AND ERASE CREATED POINTER


; DEFINE FILE NAMING PROCESSES

; AW6BT - SETUP FOR SIXBIT AW
AW6BT: (6 .Q T/K WNBTT S IETT    ; PUSH 6 FOR SIZE
  B/K -40 0 +I .Q T/K WBTT S IETT)       ; PUSH NULL-40 FOR BASE

; AWRS - POP SIXBIT SETUP FOR AW
AWRS: (.Q T/K WNBTT S DETT .Q T/K WBTT S DETT)

; SETRD - W(0)=SYMBOL AND USES EXTERNAL NAME OF IT .LSF
SETRD: (DSK RSIF AW6BI  ; SETUP FOR SIXBIT AW AND RESET INTERFACE
  LNNT CVNKL DSK 4 +IS P 0 V RI AW  ; LAYDOWN SIXBIT FILE NAME
  .Q "LSF" DSK 5 +IS P 0 V RI AW  ; LAYDOWN SIXBIT EXTENSION
  AWRS) ; CLEAN-UP AND EXIT

; SETWR - W(0)=SYMBOL AND USES EXT NAME OF IT .LSF
SETWR: (DSK RSIF AW6BI  ; SETUP FOR SIXBIT AW AND RESET INTERFACE
  LNNT CVNKL DSK 10 +IS P 0 V RI AW  ; LAYDOWN SIXBIT FILE NAME
  .Q "LSF" DSK 11 +IS P 0 V RI AW  ; LAYDOWN SIXBIT EXTENSION
  AWRS) ; CLEAN-UP AND EXIT

; RDF - SIMPLE VERSION - READ FROM FILE W(0) INTERFACE DSK
RDF: (SETRD DSK WRD I)

; WRF - SIMPLE VERSION - WRITE FILE W(0) INTERFACE DSK
WRF: (SETWR DSK WWR I)
```

```
; DEFINE SAVE FOR RESTART ROUTINE

SAVE: (SV TTY RSIF DSK RSIF ((.- .O "V32 CONTINUING" ..)
        .O "V32 RESTARTED") WRWWR CR.LF)


; END OF INITIAL BOOTSTRAP - NOTIFY USER AND GO TO HIM

CR.LF! "INITIAL BOOTSTRAP LOADED" WRWWR! CR.LF! CR.LF!
TTY WRD R!
```

```
; SIMPLE ON-LINE EDITING SYSTEM FOR L*(F)
; EDT CHANGES KCR TO GET NEXT AND PRINT SYMBOL IN NEXT,
;        LEAVING NEXT IN W TO BE EDITED AS DESIRED.  EDT STARTS
;        BY PRINTING FIRST SYMBOL.  EDT ALSO PUTS EDTND IN W
;        AS MARKER, BOTH FOR USER AND FOR EDT.  (SEE BELOW).
;        DO NOT REMOVE MARKER.
; EDT. REMOVES ALL SYMBOLS IN W DOWN TO (AND INCLUDING) EDTND.
;        IT ALSO RETURNS ACTION FOR KCR TO PREVIOUS VALUE.
; EDTCRA IS ACTION FOR CR.
; EDTD ALLOWS USER TO GO DOWN A LEVEL.
;
; NOTE: ONE IS NOT "IN A SYSTEM" WITH EDT, BUT SIMPLY CAN
;        STEP THROUGH PROGRAMS AT WILL, DOING WHATEVER
;        OTHERR PROCESSING SEEMS APPROPRIATE.  IT IS A GOOD
;        IDEA, HOWEVER, TO LET EDT. CLEAN UP FOR YOU.

EDT: (EDTSTØL RN EDTNDØL .Q EDTCR .Q KCR ICKA EDTST)

EDT.: ((P EDTND =S .+ U .R) PRS CR.LF .Q KCR DCKA)

EDTCR: ((P EDTND =S .- EDT. ..)    ; QUIT IF NO MORE
  (N P .- P S PRS SPACE ..)  ; PRINT SYMBOL IF FIND NEXT
  .Q '= WRWWR PRS SPACE)     ; PRINT =NIL AND REMOVE NIL IF END

EDTD: (P S EDTST RN EDTST)

CR.LF! "EDT LOADED" WRWWR! CR.LF!
TTY WRD R!
```

Appendix 15 - Listing of Stepping Monitor File   STPMF.LSF 1

```
; STEPPING MONITOR FOR L*(F) - &STP
; &STP!  CHANGES KCR TO STEP THROUGH A PROGRAM (W(0)).
;        &STP IS A CLOSED SUBROUTINE AND CAN BE EXECUTED FROM
;        WITHIN A PROGRAM.
;        AT EACH POINT IT EXECUTES AN ARBITRARY ROUTINE FROM W&STP.
;        THE DEFAULT PRINTS W (PRL), WITH THE SYMBOL TO BE
;        EXECUTED SITTING IN THE TOP OF W.
;        DOING KCR EXECUTES W(0) AND ADVANCES TO THE NEXT ONE.
;        THUS, W(0) CAN BE CHANGED BEFORE CR, CHANGING WHAT IS EXECUTED.
;        THE SAME PATH IS FOLLOWED AS WITH REGULAR INTERPRETATION.
;          THE USER MAY ALTER THE CONTROL FLOW BY USING ONE OF
;          THE FOLLOWING CONTROL PROCESSES: &. &.- &.+
;          &.. &..- &..+ &.R &.R- &.R+
;          TO EXIT A LEVEL WHEN CODE DOESN'T SHOW IT: U! &.
;
; &STP.!  TERMINATES THE STEPWISE EXECUTION AND RETURNS KCR
;        TO ITS PRIOR STATE. &STP. REMOVES THE ITEM FROM W, BUT
;        W WILL STILL HAVE ARGUMENTS IF A ROUTINE WAS
;        TERMINATED IN MID-STREAM.
;
; &AUTO!  CONTINUES THE STEPWISE EXECUTION IN AUTOMATIC MODE
;        UNTIL AN &MANU IS EXECUTED OR UNTIL NORMAL TERMINATION.
;
; &STPD!  ALLOWS THE USER TO DESCEND ONE LEVEL TO STEP
;        THROUGH A NAMED PROGRAM SUB-LIST.
;
; NOTE:  ACCESSES TO WHS OR WHN ARE CHANGED TO ACCESSES TO
;        &WHS OR &WHN.  YOU MAY RUN INTO PROBLEMS IF YOU TRY
;        TO REACH ACROSS THE BOUNDARY (I.E. THE LEVEL &STP
;        WAS ENTERED).

DEF/P!
&STP: (&WXS@L R NIL &WXN@L R .O &STPX .O KCR ICKA .O NOP &DSC &STPX &EXEC)

&STPD: (&WXS R .O NOP &DSC)

&STPX: (&.X (&ADV .+ &ASC .R+ &STP. ..) &WXS S W&STP@L S .X)

&AUTO: (NIL &AUSW R (&.X (&ADV .+ &ASC .R+ &STP. ..) &WXS S W&STP
  S .X &AUSW S .R-))

(W PRL) W&STP R!

&STP.: ((&ASC .R+) .O KCR DCKA .O "..&END " WRWWR W&STP S .X TRUE &EXSW R)

&ADV: (&WXN S P S &WXS R P N &WXN R)

&ASC: (&WHS@L N P .- &WHS S &WXS R &WHS D &WHN@L S &WXN R &WHN D)

&DSC: (&WXS S &WHS T &WXN S &WHN I &WXS S &WXN R)
```

```
&.X: (P T &ITL@L SBAL .X)

&ITL: (T/P &/P T/M &/M T/L &/L T/I .X T/K .X T/C .X)

&/P: (P LNNT (.- .X ..) &WXS R &DSC)

&/M: (&IL/M SBAL .X)

&IL/M@L: (. &. .+ &.+ .- &.- .. &.. ..+ &..+ ..- &..-
  .R &.R .R+ &.R+ .R- &.R- .Q &.Q .X &.X)

&/L: (&IL/L SBAL)

&IL/L@L: (WHN &WHN WHS &WHS)

&.: (NIL &WXN R)
&.+: (.- &.)
&.-: (.+ &.)
&..: (NIL &WXN R NIL &WHN R)
&..+: (.- &..)
&..-: (.+ &..)
&.R: (&WHS S &WXN R)
&.R+: (.- &.R)
&.R-: (.+ &.R)
&.Q: (&WXN S S &ADV U)

SBAL: ((V W? I (P S W? S =S .+ N N P .P+ U W? S ..) N S) W? D)

&MANU: (TRUE &AUSW@L R)

&EXEC: (NIL &EXSW@L R (&EXSW S .+ WRD S RD P &EXSV@L R
  RDCX .XCX &EXSV S EL .R))

; RETURN TO USER

CR.LF! "STPM LOADED" WRWWR! CR.LF!
TTY WRD R !
```

```
; UTILITIES FOR L*(?)
;
; CTYP - CREATE NEW TYPE W(0) SIMILIAR TO W(1)
;
; CBLK - CREATE BLOCK W(0) WORDS LONG OF TYPE W(1)
;
; CTT - CREATE NEW TYPE TABLE
;
; LODTT - LOAD TYPE TABLE W(0) WITH W(1) AS ENTRIES
;
; .XTT - EXECUTE W(1) BY TYPE TABLE W(0)
;
; RNNT - REPLACE NAME W(1) BY SYMBOL W(0) IN NAME TABLE


; CTYP - CREATE NEW TYPE MAKES (W(0)) THE CHARACTERISTIC SYMBOL
;        FOR A NEW TYPE SIMILAR TO (W(1)).  CTYP SETS
;        UP THE CURRENT TYPE TABLES WITH THE APPROPRIATE ENTRIES.
;        A BLOCK OF SPACE IS OBTAINED BOR THE NEW TYPE, BUT NO
;        ATTEMPT IS MADE TO BUILD A SPACE EXHAUSTED ROUTINE.
;        ERR10 IS USED FOR THE SPACE EXHAUSTED RTN.
; NOTE:  AFTER DOING A CTYP, YOU MAY WHICH TO DO ANY OR ALL OF
;        THE FOLLOWING:
;          T/- : (---)        ; DEFINE THE CHARACTERISTIC SYMBOL
;          (T/- CSPT) E/- WSPXT S! RETT!  ; DEFINE A SPACE-EX. RTN.
;          (P T/- =T .- PR-- ..) PRSTR I!  ; DEFINE A PRINT RTN.
;          T/- TYPL I! '- TYPL I!   ; SETUP @- FOR DEFINING T/-
;          -- &ITL I! T/- &ITL I!   ; SETUP &STP TO HANDLE T/-

6 T.HI@I RI!     ; CURRENT HIGHEST TYPE INDEX

DEF/P!
CTYP : ( W0 I W1 I        ; W0←NEW TYPE, W1←MODEL
  W1 S W0 S RC            ; SETUP CHARACTERISTIC SYMBOL
  T.HI 1 T.HI +I            ; FIND NEW TYPE INDEX AND BUMP INDEX CNT
  W0 S RT                 ; SET TYPE OF CHARACTERISTIC SYMBOL
  W1 S .ITT SETT W0 S .ITT RETT ; CARRY OVER ENTRIES FOR TYPE TABLES
  W1 S .IPTT SETT W0 S .IPTT RETT
  W1 S ARTT SETT W0 S ARTT RETT
  W1 S ARPTT SETT W0 S ARPTT RETT
  W1 S AWTT SETT W0 S AWTT RETT
  W1 S AWPTT SETT W0 S AWPTT RETT
  W1 S BTT SETT W0 S BTT RETT
  W1 S NBTT SETT W0 S NBTT RETT
  .O ERR10 W0 S SPXTT RETT
  W1 S RDTT SETT W0 S RDTT RETT
  W1 S RDPTT SETT W0 S RDPTT RETT
  W1 S WRTT SETT W0 S WRTT RETT
  W1 S WRPTT SETT W0 S WRPTT RETT
  W0 S P TTT RETT                   ; SET CHAR. SYMBOL INTO TYPE TYPE TABLE
  W0 S CSPT               ; GET A BLOCK OF SPACE FOR THE NEW TYPE
```

```
  W0 D W1 D )             ; CLEAN-UP AND EXIT

; CBLK - CREATE BLOCK W(0) WORDS LONG OF TYPE W(1)

CBLK: (P W0 I W1 I W2 I ; W0←W1←LENGTH, W2←TYPE
  (W0 S 2000 <I .- HALT)          ; ERROR IF BLOCK TOO LARGE
  W2 S C P W3 I SAVEW             ; W3←WSAVE←CURRENT LOC
  (W0 S -1 0 +I P 0 =I .+ W0 R   ; EXIT IF WE HAVE ENOUGH
  W2 S C W3 S 1 +IS P W3 R =S .R+        ; REPEAT IF SEQUENTIAL
  W1 S W0 R W2 S C P W3 R WSAVE R .R)    ; START OVER IF NOT
  U W0 D W1 D W2 D W3 D RSTRW)   ; CLEAN-UP AND EXIT.

; CTT - CREATE NEW TYPE TABLE

CTT: (T/C TTN CBLK)

; LODTT - LOAD TYPE TABLE W(0) WITH W(1) AS ENTRIES

LODTT: (P W0 I TTN +IS W1 I W2 I   ; W0←START, W1←END, W2←ENTRY
  (W0 S W1 S =S .+ W2 S W0 S R W0 S 1 +IS W0 R .R)
  W0 D W1 D W2 D)

; .XTT - EXECUTE W(1) BY TYPE TABLE W(0)

.XTT: (V IT0 PI +IS S .X)

; RNNT - REPLACE NAME W(1) BY SYMBOL W(0) IN NAME TABLE

RNNT: (V LNNT 1 +IS R)

; NOTIFY USER AND RETURN TO HTM
CR.LF! "UTILITIES LOADED" WRWWR! CR.LF!
TTY WRD R!
```

```
; DICTIONARY TREE FOR L*(F) WITH SYNTAX ACTIONS AND CONTEXTS
;         REQUIREMENTS: UTTLF
; FORM OF NODE OF TREE:
;         NODE: (CHARACTER RECOG-LIST UP-LINK NODE ... NODE)
; FORM OF RECOG-LIST:
;         RECOG-LIST: (CONTEXT SYMBOL ... CONTEXT SYMBOL)
; NODE OF DEFINITION OF A SYMBOL IS PLACED IN A CONTEXT-NODE
;         LIST POINTED TO BY THE NEXT-PART OF SYMBOL-DESC. WORD.
; NNA IS THE NAME NODE ACTION USED FOR ALL CHARACTERS, EXCEPT
;         THOSE HAVING SPECIAL ACTIONS.
; ABND1 IS THE BOUNDARY ACTION FOR THE DICTIONARY.
;
; SYNTAX ACTIONS ARE OF THE FORM:
;         (PRECEDENCE-ORDER IMMEDIATE-ACTION DELAYED-ACTION)
; .ZATT POINTS TO THE CURRENT SYNTAX ACTION TYPE TABLE:
;         ZAW - INPUT SYMBOL TO W
;         ZA  - SYNTAX ACTION INTERPRETER (FOR T/ZA)

DTREE@L: (NIL NIL NIL)  ; INITIAL DICTIONARY TREE
DTREE WN@L R!    ; WN HOLDS POINTER TO CURRENT NODE IN DT
DEF/P!
NNA: (WN S N N (N P .- P S S WK S =S .R- S WN R TRUE) .+ T/L
  C P WN S V R P NIL V I P WK S V I P WN S N N IA WN R)

ABND1: (WN S DTREE =S .+ ((INUMF 1 =I .- T/I C P (ISGN 2 0 /RI
  0 =I .+ 0 INUM INUM -I U) INUM V RI ..) WN S N S P (.+ U T/L
  C P WN S N R P WTC S C P SAVEW V R ABND2 ..) LSCSL P (.- S ..)
  U WN S N S P WTC S C P SAVEW V I ABND2) DTREE WN R
  0 INUM RI 0 INUMF RI 0 ISGN RI P .ZATT S .XTT)

ABND2: ((WCTX S V I WSAVE S TD +IS P (N .+ WN S T/L C P SAVEW
  R WCTX S WSAVE S I RSTRW V RN ..) N P SAVEW WN S V I WCTX S
  RSTRW I) RSTRW)

ANK1: (-1 INUMF RI NNA)

ADK1: ((INUMF 0 =I .- 1 INUMF RI) NNA INUMF -1 =I .+ WK S ACCD)

A-K1: ((INUMF 1 =I .- -1 INUMF RI) ISGN 1 ISGN +I U NNA)

A+K1: ((INUMF 1 =I .- -1 INUMF RI) NNA)

; DEFINE ROUTINES FOR SYNTAX ACTIONS

T/L T/ZA CTYP!   ; CREATE TYPE SYNTAX ACTION (T/ZA)
T/ZA: (0)        ; MAKE T/ZA THE NULL ACTION
(T/ZA CSPT) T/ZA WSPXT S! RETT! ; DEFINE A SPACE-EX. RTN. FOR T/ZA
(P T/ZA =T .- PRLS ..) PRSTR I! ; DEFINE A PRINT RTN. FOR T/ZA
T/ZA TYPL I!  @Z TYPL I!        ; SETUP @Z FOR DEFINING T/ZA
.X &ITL I!  T/ZA &ITL I!        ; SETUP &STP TO DO T/ZA PROPERLY
```

```
CTT! .ZATT@C R! ; CREATE THE SYNTAX ACTION TYPE TABLE
ZAW .ZATT S! LODTT!      ; AND LOAD IT WITH ZAW (NOP)
ZA T/ZA .ZATT S! RETT!   ; SETUP ZA AS ACTION FOR T/ZA

(ABND .Q ZAW T/ZA .ZATT S IETT) '[ RCKA! ; SETUP [-] TO TURN OFF
(ABND T/ZA .ZATT S DETT) '] RCKA!        ; SYNTAX ACTION FOR -

; ZA - SYNTAX ACTION INTERPRETER
ZA: (W0 I ((W0 S S WZA@L S S >T .+
  WZA S N N S .X           ; EXECUTE DELAYED AC
  WZA S S WZA D            ; GET PREC. ORDER & POP
  W0 S S =I .R-)           ; REPEAT UNLESS P.O. SAME
  W0 S N P .- S .X         ; EXECUTE IM. AC
  W0 S N N P .-            ; EXIT IF NO DELAYED AC
  W0 S WZA I) U            ; STACK DELAYED AC
  W0 D)

BTAX: (377777777777) ; DEFINE BOTTOM ACTION WITH P.O.=LARGEST POS. NUM
BTAX WZA R!      ; WZA IS THE DELAYED ACTION STACK

; DEFINE ROUTINES FOR CONTEXT HANDLING

; RNDT - REPLACE NAME W(1) WITH STRUX W(0) IN CURRENT CONTEXT
RNDT: (W0 I P W1 I LNDT P SAVEW N S ((LSCS1 P .+ U WSAVE S N
  S P W0 S V I WCTX S V I ..) W0 S V R) W0 S TD +IS P W2 I N
  ((.+ T/L C P RSTRW V R P WCTX S V I W2 S RN ..) W2 S N
  ((LSCS1 P .+ U W2 S N P RSTRW V I WCTX S V I ..) RSTRW V R))
  W0 D W1 D W2 D)

; LSCSL - LOOKUP SYMBOL IN CONTEXT-SYMBOL LIST
LSCSL: ((W0 I WCTX  ; SEARCH CONTEXT STACK FOR EACH ENTRY IN LIST
  (P S W1 I W0 S (P S W1 S =S P .+ U N N P .R+) W1 D .+
  N P .R+ ..) V U N) W0 D)

; LSCS1 - LOOKUP SYMBOL IN CURRENT CONTEXT ON CONTEXT-SYMBOL LIST
LSCS1: ((P S WCTX S =S .+ N N P .R+ ..) N)

BTCTX@L WCTX@L R!  ; WCTX IS CONTEXT STACK (HAS BOTTOM CONTEXT INITIALLY)

; NOW, DEFINE ROUTINES TO EFFECT THE SWITCH FROM NT TO DT

; SWTCH PUTS ALL NAME TABLE ENTRIES INTO THE DICT. TREE AND
;  THEN REPLACES THE OLD NAME TABLE FUNCTIONS WITH DT FUNCTIONS.
SWTCH: (NT1T 2 0 *I NT1 V +IS W0 I        ; ALL NT ENTRIES TO DT
  NT1 (P CVNKL CVKDN WNSV@L R 1 +IS P S P TD +IS ((P N .+ T/L C
  P WNSV S V R P WCTX S V I V RN ..) N P WNSV S V I WCTX S V I)
  WNSV S N ((P S .+ V T/L C P SAVEW R WCTX S WSAVE S I RSTRW
  V R ..) S I WCTX S WNSV S N S I) 1 +IS P W0 S >S .R+ U)
  .Q ABND1 .Q ABND RRTN .Q ANK1 .Q ANK RRTN .Q A-K1 .Q A-K RRTN
  .Q A+K1 .Q A+K RRTN .Q ADK1 .Q ADK RRTN .Q LNDT .Q LNNT RRTN
  .Q CVDNK .Q CVNKL RRTN .Q RNDT .Q RNNT RRTN W0 D)
```

```
; RRTN - REPLACE RTN W(0) WITH RTN W(1)
RRTN: (P W0 I RC .Q T/P 0 TI W0 S RT W0 D)

; LNDT - LOOKUP NAME IN DT REPLACES LNNT
LNDT: (TD +IS N P .- LSCSL P .- S)

; CVKDN - CONVERT K-LIST TO DICT NODE
CVKDN: (P S WK R NNA N P .R+ U WN S DTREE WN R)

; CVDNK - CONVERT DICT NODE TO K-LIST REPLACES CVNKL
CVDNK: (P S .Q WKLSV@P R (N N S P .- P S .Q WKLSV I .R) U .Q
  WKLSV N NIL .Q WKLSV RC)

SWTCH!  ; SWITCH FROM NT TO DT!

; NOW, RETURN TO USER

CR.LF! "DICT LOADED" WRWWR! CR.LF!
TTY WRD R!
```

I. Changes to the Kernel

(1) Machine stack space was doubled from 128 to 256 words.

(2) The size of initial T/L reserved space was doubled from 32 to 64 cells.

(3) Sizes of both initial T/L and T/P available space were increased by 320 cells to 1280 cells for T/L and 1344 cells for T/P.

(4) Processes with no inputs and no outputs were given a null prefix instruction (a JFCL) so that the address of the process + 1 is the start of the process stem. This makes these processes consistent in this respect with processes of other input-output characteristics.

(5) A "START 141" (Debug entrance from monitor) now reads out the contents of R1 - R5 into new cells R1SV - R5SV , and MSTKP into new cell MSPSV before calling DEBUG.

(6) The internal save areas and machine stack have been moved away from the operating system processes to immediately before initial T/C available space. It would now be possible to expand the machine stack by pre-empting T/C available space (if not already used for other purposes).

(7) EXEC has been modified so that the current read interface is reset (RSIF) when an end-of-file is detected. It now operates analogously to the following T/P list:

```
((WRD S RD P .- P WEXEC I RDCX .XCX
   WEXEC S WEXEC D EL .R ) WRD S RSIF )
```

When the current read interface (WRD.S) is TTY , this addition to EXEC allows one to exit from a nested call on EXEC with a control-Z (end-of-file signal for TTY) and then continue reading successfully from the TTY at the outer level.

(8) ERROR has been modified so that the working register context (R1-R5) and machine stack pointer (MSTKP) are read into cells R1SV-R5SV and MSPSV before the swap into debug context and execution of WERR.S . After return from WERR.S and the swap back out of debug context, R1-R5 and MSTKP are restored from the cells R1SV-R5SV and MSPSV. This makes

error recovery much more feasible, but is just a stopgap solution.

As an error recovery example consider ERR15, the "out of space in name table" error. If we look at the point in CSNTW where the error occurs, we see that only the contents of R1 (addr of name table) and R5 (current name table index) are meaningful. Thus, we can write an error recovery routine to be placed into WERR which will set up a new name table complete with size and index words, insert it onto WNT, put its address into cell R1SV, put an initial table index of zero into cell R5SV, and exit. Execution will continue immediately after the error call location in CSNTW with the contents of R1 and R5 reflecting the new name table, and error recovery will be complete.

(9) CSP was modified to make it return space to the monitor if the value of the size W(1) is negative. If the value of W(1) is zero, core allocation is not changed. When no space is obtained from the monitor, CSP outputs NIL.

(10) <S , >S , =C , =I , <I , >I have been modified so that if the test succeeds and the W(1) input was NIL, then TRUE is left as output rather than the W(1) input. (This is how =S and =T already worked in V30).

(11) C (Copy W(0)) has been modified to work as the documentation says it should; namely, by copying the contents of input W(0) into the new cell which is output. In V30 C always created null structures rather than copying.

(12) C and C/L now swap into space-exhausted execution context (SPXCX) before executing space-exhausted routines (and swap back to the previous context upon return). This eliminates the possibility of space-exhausted routines failing if space is exhausted within some strange interpreter context (e.g. Write Context).

(13) LNNTW has been changed to search name tables backwards, so that most recently defined names will be found first when duplicate names exist for a symbol. LSNTW was also changed to search backwards for consistency.

(14) RD was changed to access the character base through the current base type table in WBTT rather than directly.

(15) In RD the SETSTS (set status) instruction to reset the end-of-file flag immediately after an end-of-file condition

was detected has been removed since it didn't really work. The problem of "permanent" end-of-file indications from the TTY has been solved by other means. (See (7)).

(16) .IWR and .IPWP were corrected to reference the current base type table through W cell WBTT, rather than directly as BTT.

(17) The space-exhausted context swap list (SPXCX) was added as an initial T/L structure. (See (12)).

(18) Additions were made to the write interpreter type tables:

In WRTT:	.I/M for T/M
	.I/S for T/I,T/C
In WRPTT:	.IP/M for T/M
	.IP/S for T/I,T/C

(19) Initial T/M available space was moved from between initial T/L & T/P available space to between initial T/C & T/I available space.

(20) The following names were added to NT1:

MSPSV
R1SV
R2SV
R3SV
R4SV
R5SV
SPXCX

II. Changes to the Bootstrap

(1) The character action for ! now goes into standard interpreter context for execution as in : (ABND .ICX .XCX) .

(2) The Debug swap list was expanded to include DWITT and DWIPT swapped with WITT and WIPTT respectively. DWITT initially contains .ITT , and DWIPT initially contains

.IPTT , so that standard interpretation will occur in Debug mode.

(3) A bug in PRSTR was fixed. (Named T/K symbols were being printed incorrectly.)

(4) SAVE was updated to reflect the current version number.

III. Changes to the Editor (EDT).

(1) The name EDTDN was changed to EDTD (for consistency with &STPD in the stepping monitor).

IV. Changes to the Stepping Monitor (&STP).

(1) &STP was made a closed subroutine so that it could be called from within a program.

(2) The name &STOP was changed to &STP. (for consistency with EDT. in the editor).

(3) A bug in &/P was fixed so that executing unnamed program lists with .X now works.

(4) A bug in &STPD was fixed so that you can alter W(0) and then step down the appropriate routine.

(5) &STP. was changed so that it prints "..&END" and then executes the routine in W&STP .

(6) SBAL was changed so that temporary work cell T1 is no longer clobbered.

(7) &ITL was changed so that T/I, T/K and T/C now have appropriate stepping monitor interpreters.

(8) A routine &MANU was added so that one can go into

automatic mode (&AUTO) and then leave it at a specified place (i.e. go into manual mode). &MANU must be placed in the routine that is being stepped through.