SERIES BASE NO. / VOL. / REISSUE
TM 4520 /000/00

AUTHOR
J. Barnett
TECHNICAL
J. Barnett

RELEASE
for

DATE PAGE 1
March 2, 1970

TECH MEMO *a working paper*

System Development Corporation/2500 Colorado Ave./Santa Monica, California 90406

LISP 1.5 for 360 Computer
Language Maintenance Manual

ABSTRACT

This document describes the language processing facilities of SDC's LISP 1.5 system for the IBM 360. The system operates under ADEPT on the Model 50 and TS/EXEC on the Model 65. In most respects, this LISP is a direct superset of the system operating on the Q-32. Many implementation ideas were borrowed from the LISP 2 Project. Included are chapters on system special variables, declaration logic, compiler and assembler. The information herein is for use by persons maintaining or augmenting the system.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONT'D)

1. INTRODUCTION

This paper describes the language processing facilities of SDC's LISP 1.5 system for the IBM 360. The system operates under ADEPT-50 on the Model 50 and TS/EXEC on the Model 65. The following chapters describe special variables used by the system, the LISP 1.5 compiler, the LAP assembler and the supervisor.

The supervisor controls system inputs and the interaction process. The compiler translates LISP 1.5 programs into the equivalent LAP programs. LAP programs are translated to binary and placed in the computer's memory by the assembler. Provisions are made for using the compiler as a separate function from the assembler. This is useful if it is desired to translate a large set of functions to LAP and then preprocess the assembled code or save on disc in a quickly loadable form.

The documentation conventions used were the following:

The section describing system special variables names the variable and defines the format in which its data values are kept. These definitions are preceded by one or more of the words "COMP", "ASMB", or "SUPV" indicating the defined use is for the compiler assembler or supervisor respectively.

Entities are enclosed in quotes. This usage means the entity as written, in fact as if it had been a quoted item in a LISP program. Since neither T, F or NIL need be quoted in a LISP program they are not quoted in this document. Thus, "T" is equivalent to T.

In the documentation of system functions, the tailed name of a function is followed by a parameter list and a kind descriptor. The parameter list gives a one letter dummy name for each argument of that function. The kind descriptor is either "MACRO", "INSTRUCTION", "LAP" or blank. "LAP" implying a function written in LAP and blank implying a function written in LISP.

Other information describing the function is given in the following order: Args, Bindings, Side Effects, Value, Description. The information following Args describes each parameter's format, e.g., s-express, identifies, etc. The information following Bindings is a list of special or unspecial variables bound as lambda, or block variables in the described routine. The information following Side Effects is a list of special or unspecial variables whose value bound outside of the described routine may be affected directly by actions explicitly taken by this routine, i.e., SETQ, CSET, etc. The information following Value describes the value of the routine. The information following Description is a prose account of the action of this routine.

2. SYSTEM SPECIAL VARIABLES

The following is a list of special variables used by the compiler, assembler, and supervisor, and a brief description of their use.

2.1 (LBRL . 127)

COMP: A list of unique label names that have been referenced and not defined at the time of reference.

ASMB: A list of dotted pairs of labels that have been referenced and not defined at the time of reference and, the relative byte address of the last half-word of program to reference that label.

2.2 (LBDL . 127)

COMP: A list of label names visible to GO statements at this point in the compilation.

ASMB: A list of dotted pairs of all labels defined to this point in the assembly and, the byte location at which the label was defined.

2.3 (ELAB . 127)

COMP: The value of ELAB is either bound to an integer name of a compiler-generated label or NIL. The label value is the confluence point for terminal expressions.

2.4 (SLAB . 127)

COMP: The value of SLAB is either bound to an integer name of a compiler-generated label or NIL. The label value is the confluence point for terminal statements.

2.5 (TGO . 127)

COMP: The value of TGO is either bound to an integer name of a compiler-generated label or NIL. The label value is the transfer point for a predicate upon true evaluation. A NIL value indicates a fall-through upon true evaluation.

2.6 (FGO . 127)

COMP: The value of FGO is either bound to an integer name of a compiler-generated label or NIL. The label value is the transfer point for a predicate upon false evaluation. A NIL value indicates a fall-through upon false evaluation.

2.7 (BLAB . 127)

COMP: During the compilation of statement BLOCKs, BLAB is bound to the integer name of a compiler-generated label. This label is the transfer point for RETURN statements.

During the compilation of predicate BLOCKs, BLAB is bound to a dotted pair of compiler-generated label names. The first label is the transfer point for a RETURN statement whose expression body evaluates true. The second label is the transfer point for a RETURN statement whose expression body evaluates false.

2.8 (LBCK . 127)

COMP: The value of LBCK is bound to "EXP", "STAT", or "PRED" as the BLOCK presently being compiled is encountered in the expression mode, the statement mode, or the predicate mode respectively.

2.9 (SCLS . 127)

COMP: The value of SCLS is bound to either T or NIL. T indicates that the compilation is in the statement mode. NIL indicates that compilation is either in the expression mode or the predicate mode.

2.10 (PCLS . 127)

COMP: The value of PCLS is bound to either T or NIL. T indicates that the compilation is in the predicate mode. NIL indicates that compilation is in the expression mode. The value bound to PCLS is only meaningful when the value bound to SCLS is NIL.

2.11 (AC1 . 126)

COMP: The value of AC1 is bound to either NIL, "(NIL)", or a variable name. NIL indicates that the value of the accumulator is not known at this point of the compilation. "(NIL)" indicates that the value of the accumulator is known at this point in the compilation to be NIL. A variable name indicates that the accumulator is known to contain a copy of the value of that variable at this point in the compilation.

2.12 (AC2 . 126)

COMP: The value of AC2 is bound to either NIL, "(NIL)", or a variable name. NIL indicates that the value of the accumulator is not know at this point of the compilation. "(NIL)" indicates that the value of the accumulator is known at this point in the compilation to be NIL. A variable name indicates that the accumulator is known to contain a copy of the value of that variable at this point in the compilation.

2.13 (KIND . 122)

COMP, ASMB, SUPV: The value of KIND is bound to either "SPECIAL", "UNSPECIAL", "MACRO", "INSTRUCTION", or an integer. The value is used for communications by the declaration logic.

2.14 (DELTA . 122)

ASMB: The value of DELTA is bound to an integer displacement in bytes. This displacement is relative to a register specified by R1 or R2.

2.15 (R1 . 122)

ASMB: The value of R1 is bound to either a register number or name.

2.16 (R2 . 122)

ASMB: The value of R2 is bound to either a register number or name.

2.17 (V360 . 122)

COMP, ASMB, SUPV: The value of V360 is bound to either a copy of the value of a special variable or a pointer into PRS space. The value is used for communication by the declaration logic.

2.18 (LST . 127)

COMP: The value of LST is bound to LAP code output by the compiler to this point in the compilation. This list of instructions is built in reverse order.

2.19 (SLST . 127)

COMP, ASMB, SUPV: The value of SLST is bound to a list of section numbers used in the order of occurrence for the section default logic.

2.20 (SGUS . 127)

COMP, ASMB, The value of SGUS is bound to a section number used by the section default logic and the value is the guess section.

2.21 (GL . 127)

COMP: The value of GL is bound to an integer. The integer is the label name for the next compiler-generated label.

2.22 (EXP . 127)

COMP: The value of EXP is bound to the symbolic expression presently being compiled.

2.23 (SVAR . 127)

COMP: The value of SVAR is bound to either NIL or T to reflect the condition of either no special variables being bound in the BLOCK presently being compiled or, one or more special bindings in the BLOCK presently being compiled.

2.24 (ALIST . 127)

COMP: The value of ALIST is bound to a list of local variable names visible at this point in the compilation.

ASMB: The value of ALIST is bound to a list of pairs. The first element is the name of a local variable visible at this point in the assembly. The second element of the pairs is the relative pushdown stack location used for storage of the local variable value.

2.25 (FNAM . 127)

COMP, ASMB, SUPV: The value of FNAM is bound to the name of the function, instruction or macro being compiled or assembled.

2.26 (LOC . 126)

ASMB: The value of LOC is bound to an integer, which is the relative byte address of the half-word presently being assembled.

2.27 (HB . 126)

ASMB: The value of HB is bound to be either 0, 4, 8, or 12. The value is the number of bits from the left of the half-word being assembled that the next item is to be planted in the binary program image.

2.28 (DPDP . 126)

ASMB: The value of DPDP is bound to the integer logical pushdown pointer at this point in the assembly.

2.29 (MLST . 126)

ASMB: The value of MLST is bound to a list of pairs. The first element is an identifier name of a mask used in BC or BCR instructions. The second element in the pair is the four-bit integer equivalent for the mask identifier.

2.30 (RLST . 126)

ASMB: The value of RLST is bound to a list of pairs. The first element is the identifier name of a register used in 360 instructions. The second element in the pair is the four-bit integer equivalent for the register identifier.

2.31 (PLST . 126)

ASMB: The value of PLST is bound to a list of pairs. The first element is an integer, identifying a type of 360 instruction. The second element of the pair is a pattern describing the format of that type.

2.32 (CLST . 126)

ASMB: The value of CLST is bound to a list of integers. The integers are the values of DPDP in effect when an (ARGS) was assembled for which the corresponding CALL has not been assembled.

2.33 (ELST . 126)

ASMB, SUPV: The value of ELST is bound to a list of pairs that comprise entry definitions in the system. The first element is the identifier name of an entry. The second element of the pair is the relative to SORG, byte location of the entry.

2.34 (ENTL . 126)

SUPV: The value of ENTL is bound to an integer specifying the relative to SORG, byte location at which the next entry can be defined.

2.35 (LEAD . 126)

ASMB: The value of LEAD is bound to an integer number of bytes telling the distance from the location pointed at by register PDP to the logical beginning of the pushdown stock.

2.36 (XLST . 126)

ASMB, SUPV: The value of XLST is bound to a list of pairs. The first element is a pointer into PRS space. The second element of the pair is the number of references made by the program being assembled to this PRS item.

2.37 (REF . 122)

COMP, ASMB, SUPV: The value of REF is bound to a pointer into PRS space. REF is used for communications by the declaration logic.

2.38 (ERRFLG . 122)

COMP, ASMB, SUPV: The value of ERRFLG is bound to either NIL or T. The value T reflects the occurrence of an error in an assembly or compilation.

2.39 (FCNT . 127)

COMP: The value of FCNT is bound to an integer section number to be used in the name of the next explicitly defined, embedded functional argument.

3. SYSTEM DECLARATION MECHANISM

3.1 DECLARATIONS: PHILOSOPHY OF OPERATION

The declaration logic of the compiler and assembler deal with several types of entities: special variables, unspecial variables, lexical variables, macros, instructions, and functions. At several places, the type of entity being processed is specified as a value of the variable (KIND . 122) or as a parameter to declarations logic functions specifying kind information. The values of kind and the associated interpretations are listed below:

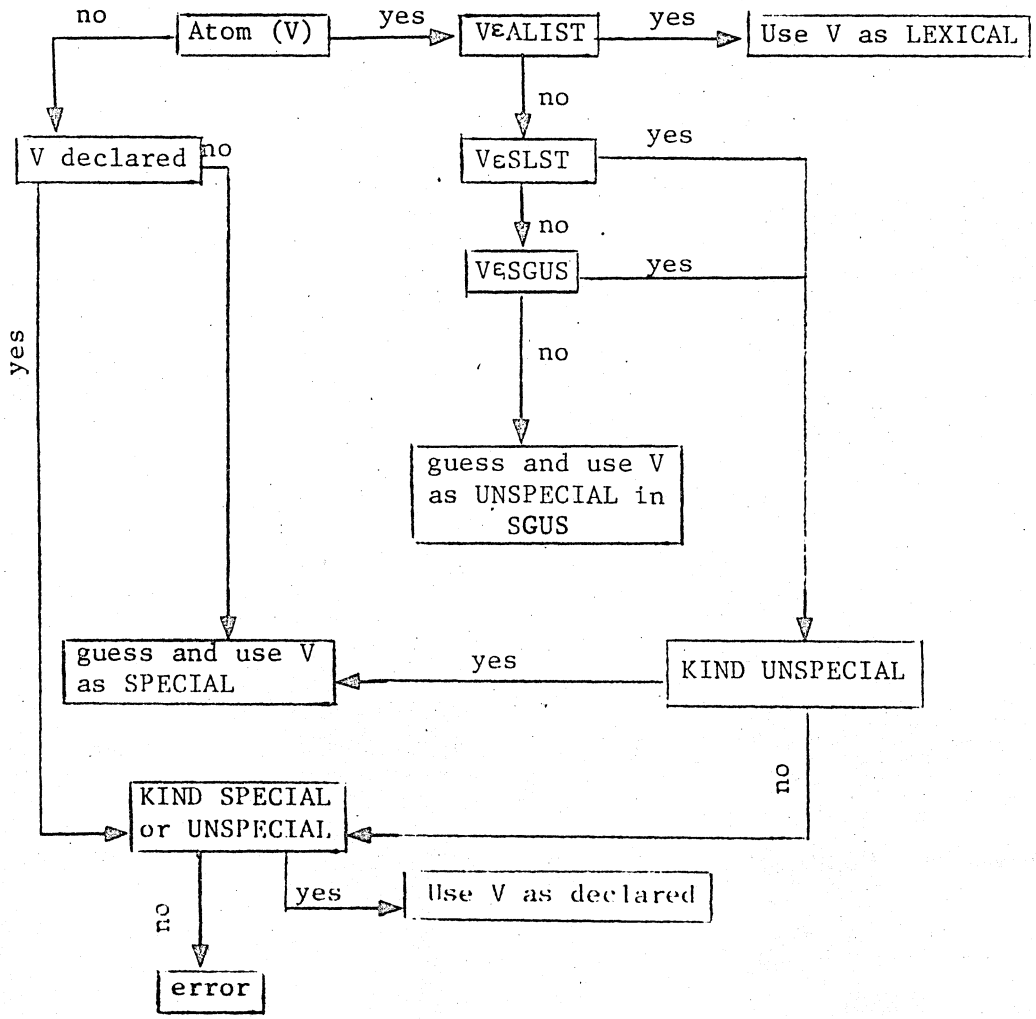
KIND	Meaning
SPECIAL	Special variable
UNSPECIAL	Unspecial variable
LEXICAL	Local variable
MACRO	Macro functional
INSTRUCTION	Instruction functional
0 - 15	Number of arguments for a function
16	Function with 16 or more arguments

KIND	Meaning
17	Function with 0 or 1 argument
18	Function with an unknown number of arguments
20	Function with an indefinite number of arguments

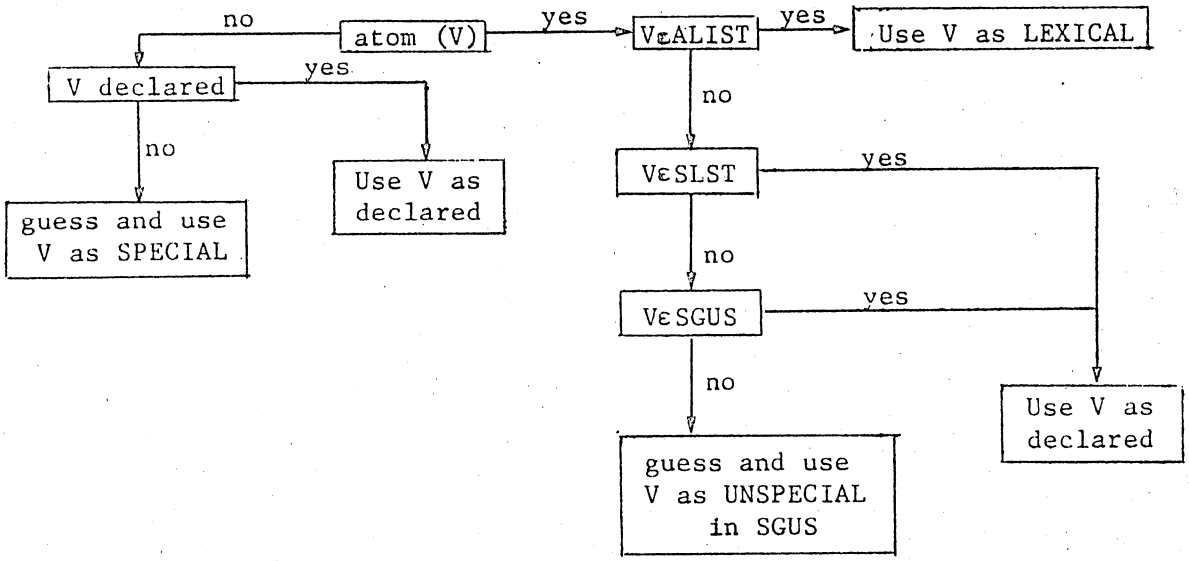
The compiler, assembler and supervisor recognize six distinct classes of variable usage. Examples of each with semantic descriptions of system actions is given in the following paragraphs. In these paragraphs, the phrase, "guess as ... " means to so declare the variable and issue a warning message. The variable KIND represents the old declaration and the variable K represents the new declaration to be made. The notation V∈SGUS means "Is V tailed in the section SGUS, declared?". The notation V∈SLST means "Is V tailed into any section in SLIST, declared?".

3.1.1 Reference Class

Variables in the reference class are used as address fields of LAP instructions and as expressions in LISP programs. The following flowchart shows the interpretation given to the variable V in a LISP program used in the reference class:

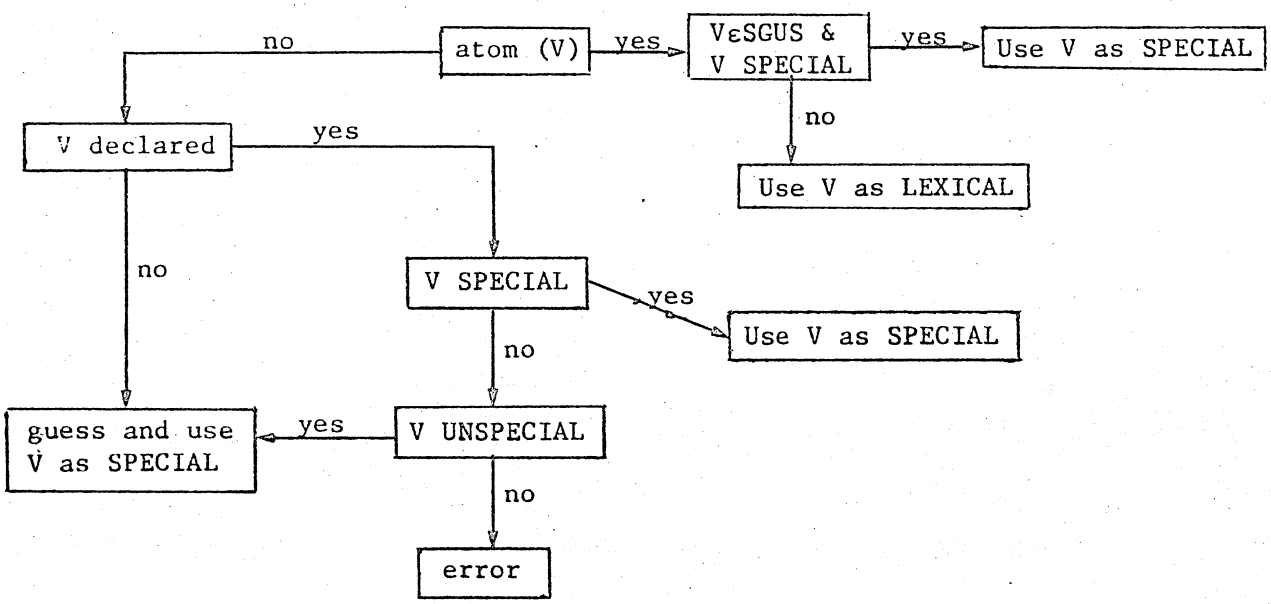


The following flowchart shows the interpretation given to variable V in a LAP program used in the reference class:



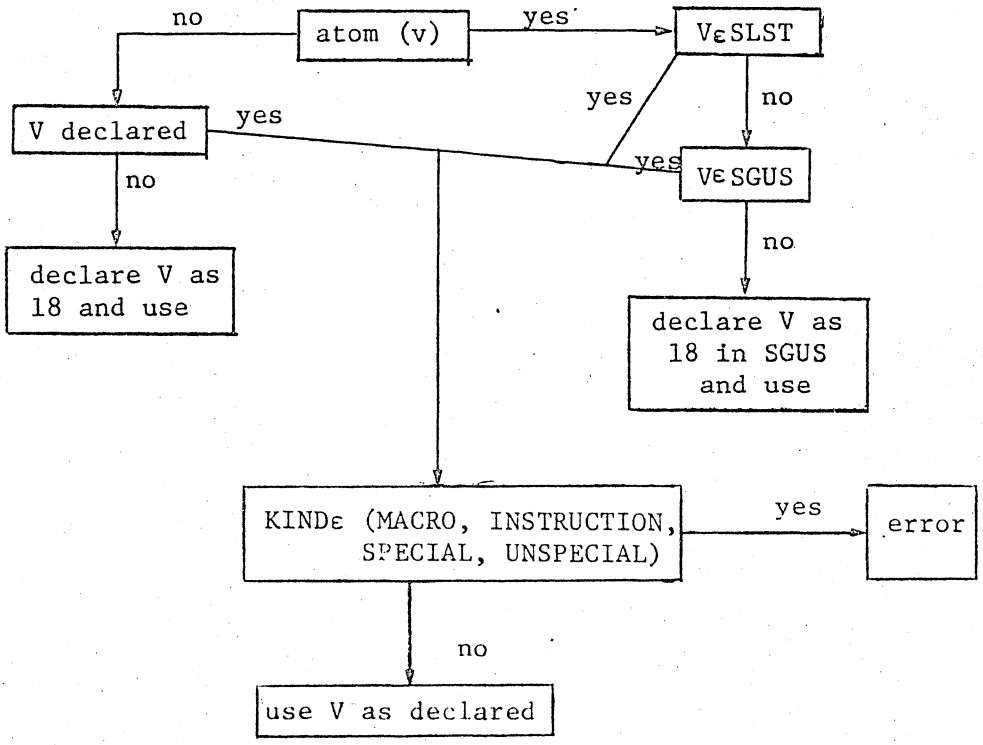
3.1.2 Bind Class

The bind class consists of variables bound as parameters of macros and functions or variables bound as block variables. The following flowchart shows the interpretation given to variable V used in either a LISP or LAP program in the bind class.

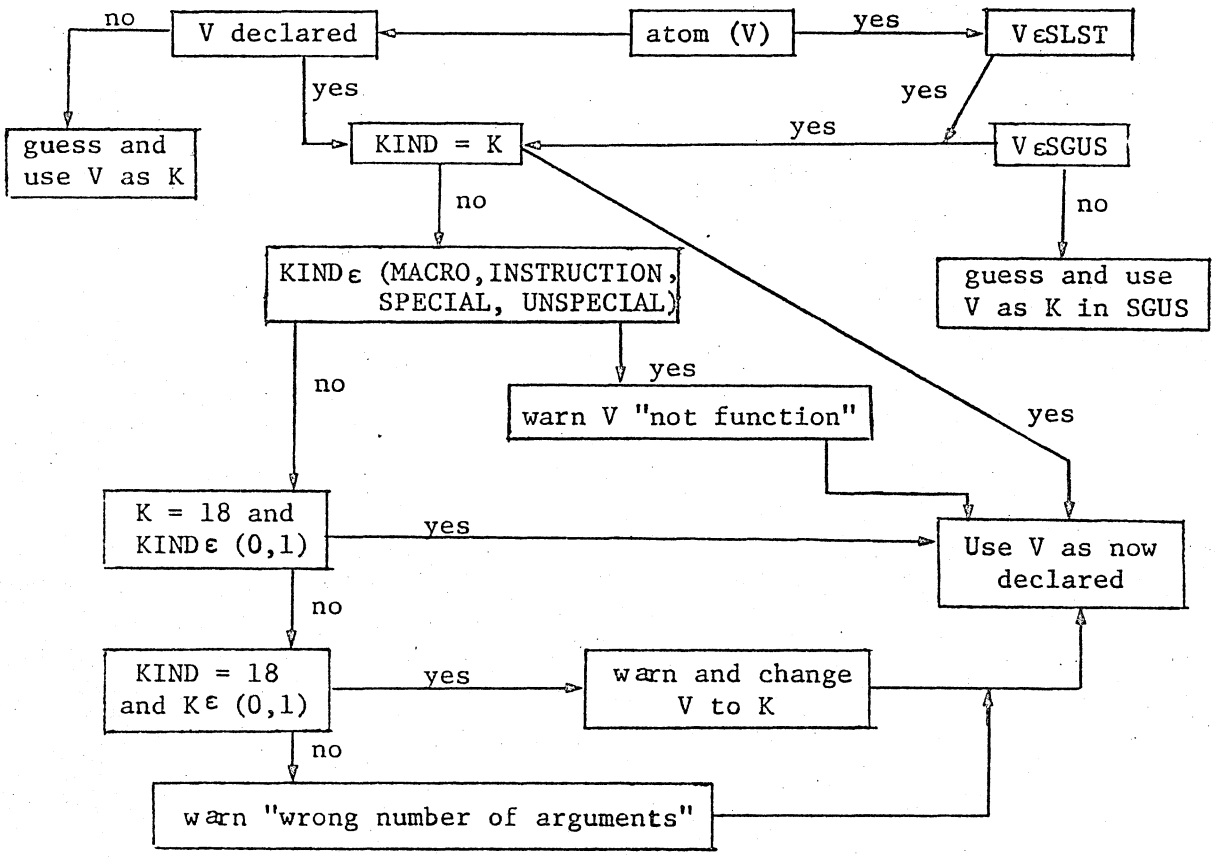


3.1.3 Explicit Function Class

The explicit function class consists of variables used in the form (FUNCTION V) in either LISP or LAP and as the body of either a CALL or CALI pseudo instruction in LAP. The following flowchart shows the interpretations given to the function name V used in LISP in the explicit function class.

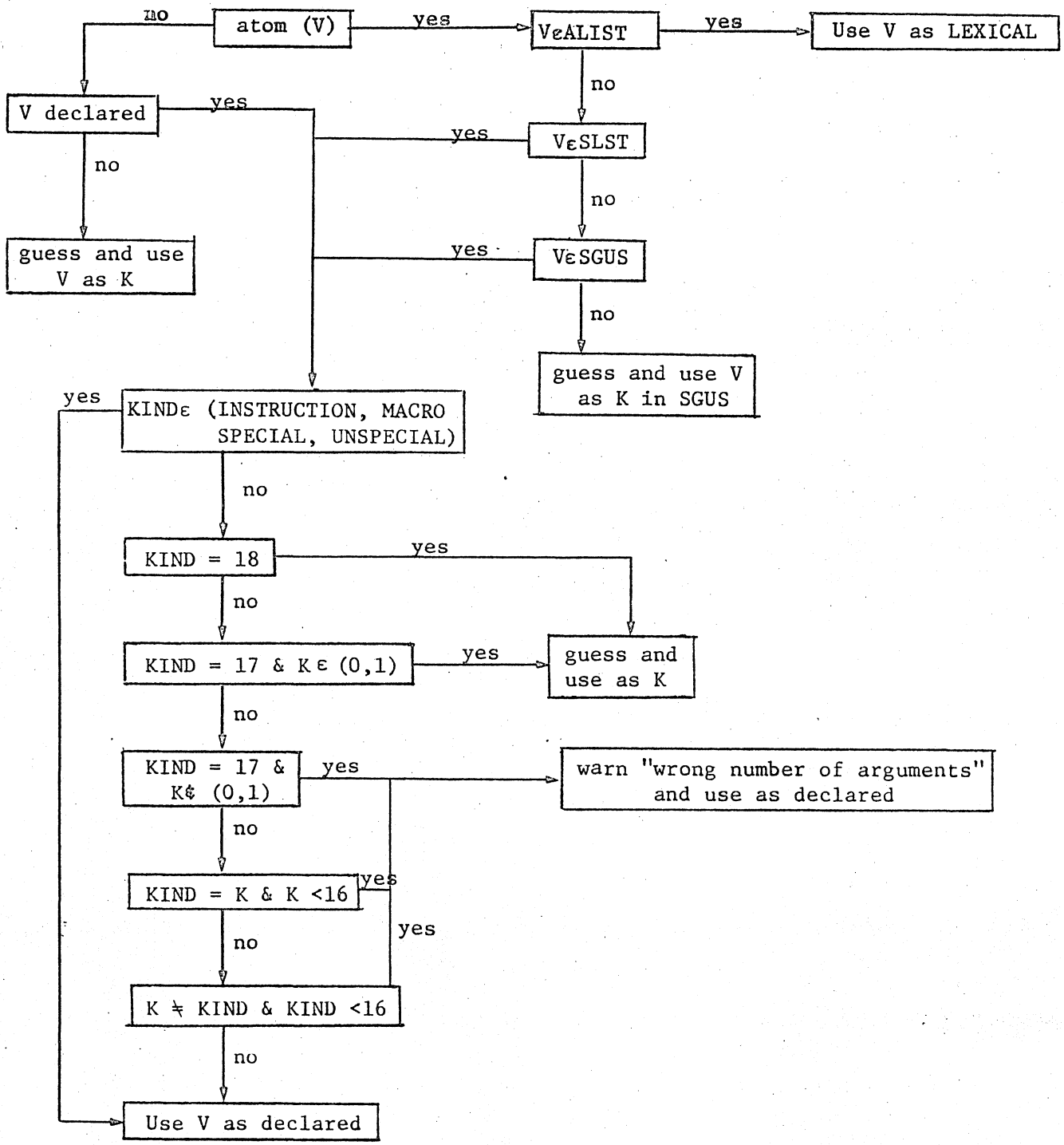


The following flowchart shows the interpretation given to the function name V used in the explicit function class in a LAP program.



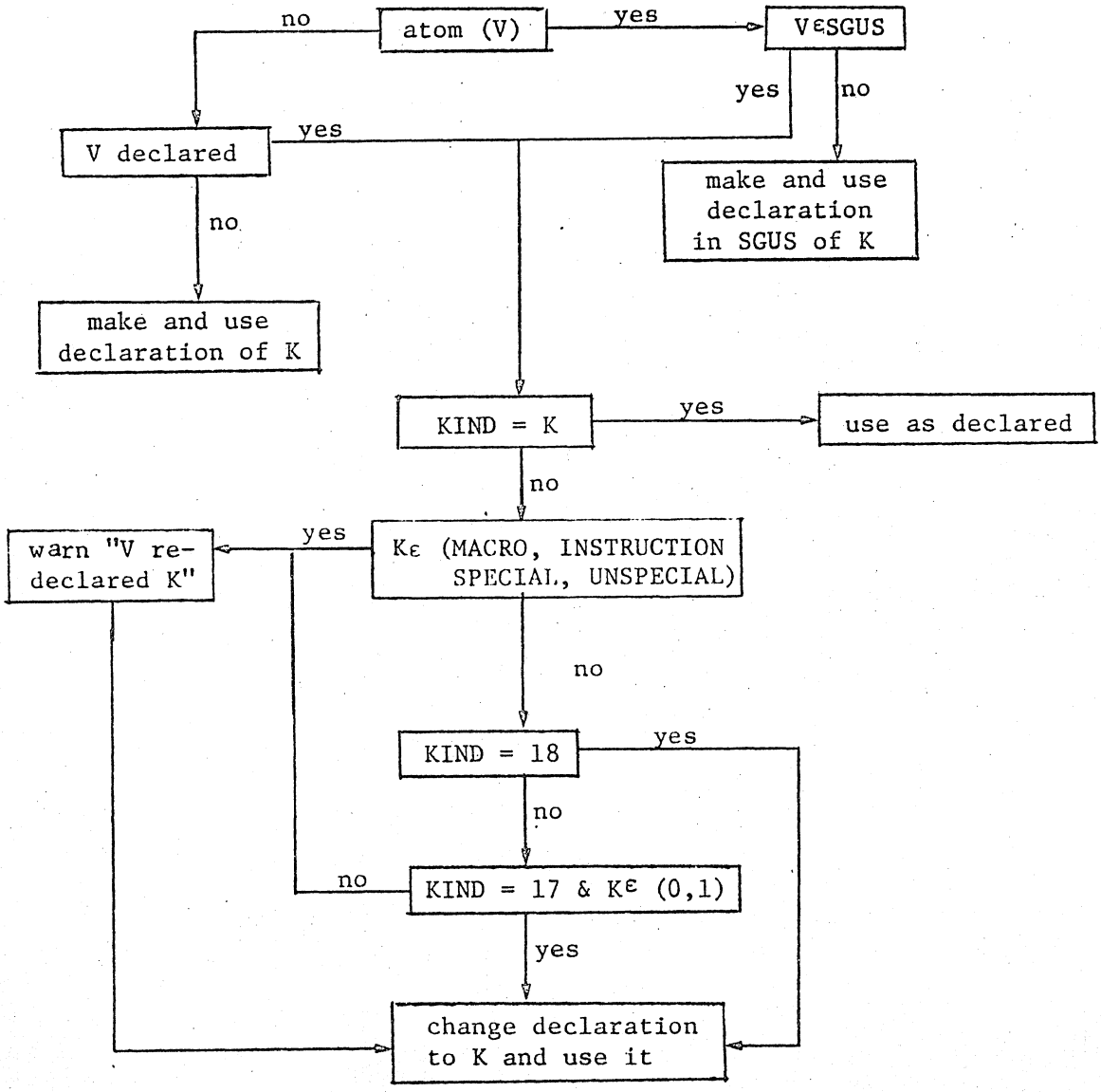
3.1.4 Operator Class

Variables appear in the operator class only in LISP programs. The following flowchart shows the interpretation given to a variable V used in a form operator in a LISP program.

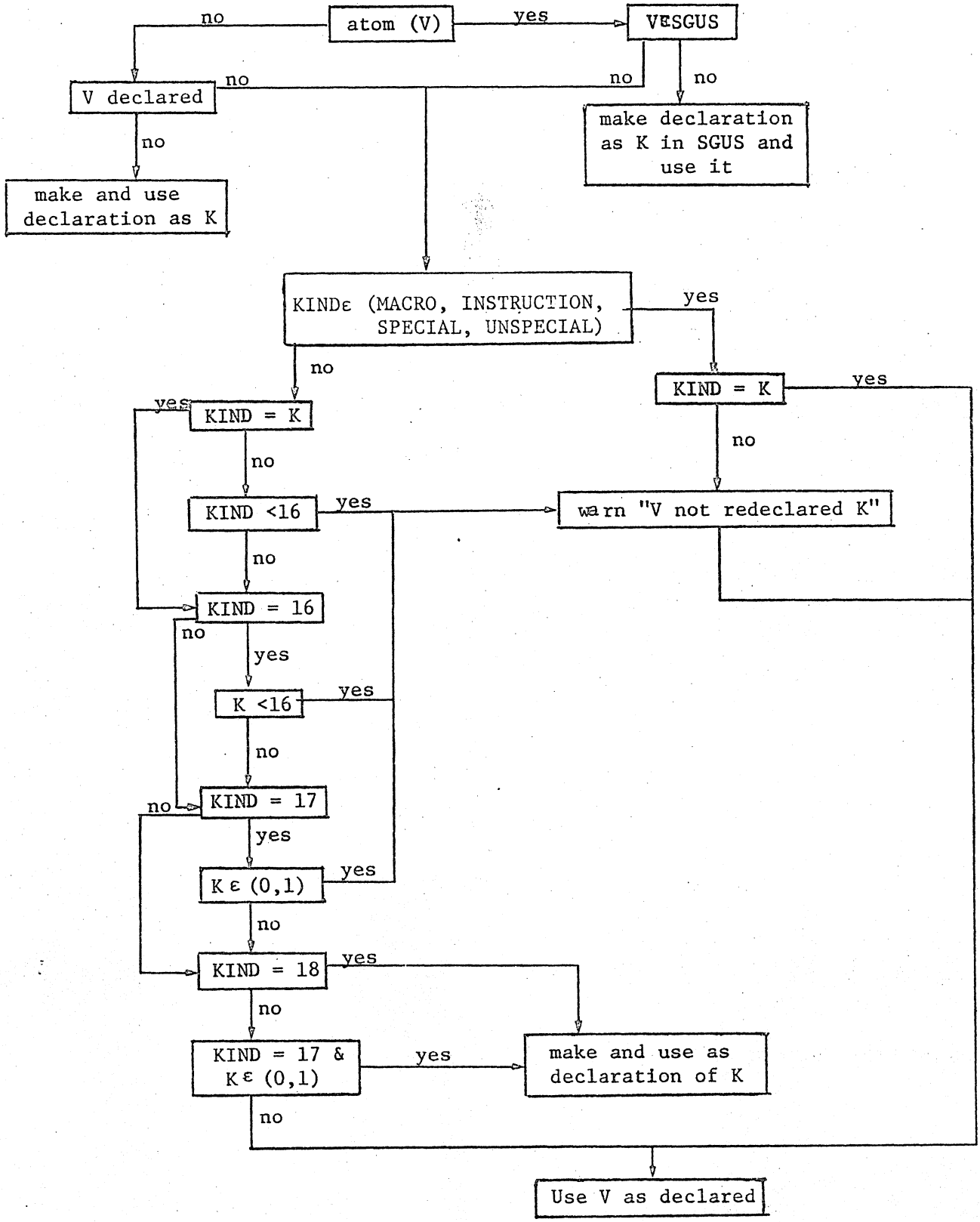


3.1.5 Entity Name Class

Variables in the entity name class are those variables being used to name the function, macro or instruction being compiled or assembled. The following flowchart shows the interpretation given to the available V in the entity name class in a LAP definition.

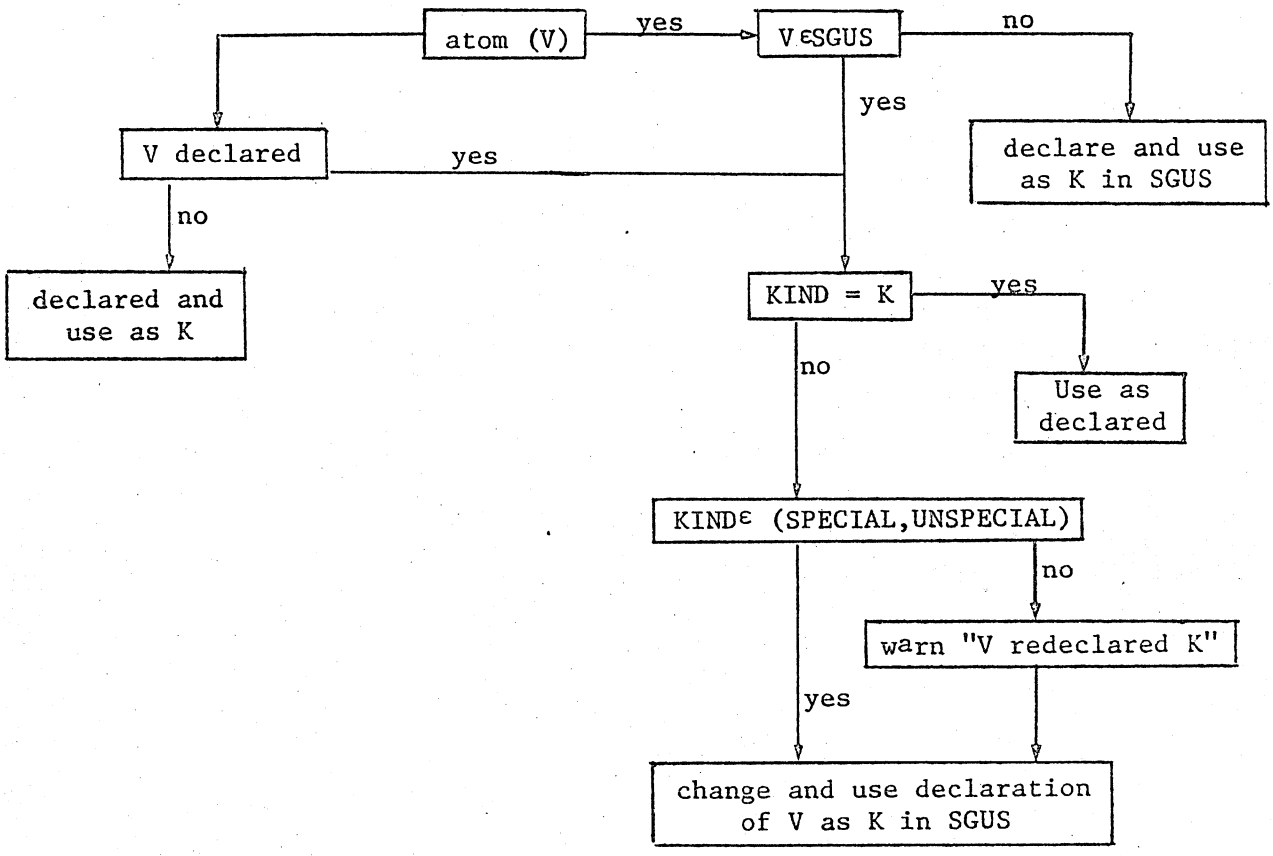


The following flowchart shows the interpretation given to the variable V in the entity name class in a LISP definition.



3.1.6 Explicit Declaration Class

Variables in the explicit declaration class are those appearing in SPECIAL or UNSPECIAL forms or as the first argument of CSET. The following flowchart describes the interpretation given to variables used in the explicit declaration class.



3.2 DECLARATIONS: IMPLEMENTATION

3.2.1 Declarations: Supervisor

The following group of functions are used by the supervisor for interfacing with the system declaration logic.

3.2.1.1 (SECTION . Ø) (S,L)

Args: S is a section number
L is a list of section numbers
Side effects: (SLST . 127)
(SGUS . 127)
Value: S
Description: SLST is set to L and SGUS is set to S.

3.2.1.2 (DVAR . 122) (L,K)

Args: L is a list of variables.
K is either "SPECIAL" or "UNSPECIAL".
Value: A copy of the variable list L, with the variables appropriately tailed.
Description: DVAR is used by SPECIAL, UNSPECIAL and CSET for appropriately defining variables.

3.2.1.3 (SPECIAL . Ø) (L)

Args: L is a list of variables.
Value: A copy of the variable list L with the variables correctly tailed.
Description: DVAR is used to declare the variables in L special.

3.2.1.4 (UNSPECIAL . Ø) (L)

Args: L is a list of variables.
Value: A copy of the variable list L with the variables correctly tailed.
Description: DVAR is used to declare the variables in L unspecial.

3.2.1.5 (CSET . Ø) (V,E)

Args: V is a variable.
E is any s-expression.
Side effects: The PRS word associated with V is altered.
Value: V appropriately tailed.
Description: The variable V is declared special using DVAR.
The value of V is set to E.

3.2.1.6 (CSETQ . Ø) (E) MACRO

Args: E is a CSETQ form.
Value: A CSET form equivalent to E.
Description: The CSETQ form, E is transformed to a CSET form with a quoted first argument.

3.2.2 Declarations: Compiler

The following group of functions are used by the compiler for interfacing with the system declaration logic.

3.2.2.1 (INSL . 127) (V)

Args: V is an identifier.
Value: NIL or a tailed name.
Description: Using GDEC, a declaration for V in a section from SLST is looked for. If none is found, the value is NIL. Otherwise the value is V dotted with the first section number in SLST for which a declaration exists.

3.2.2.2 (GSDC . 127) (I,S,K)

Args: I is an identifier.
S is a section number.
K is a kind of declaration.
Value: The dotted pair of I with S.
Description: The warning message, "((i . s) GUESSED k)" is issued and MDEC is then used to make the declaration of kind K.

3.2.2.3 (RDEF . 127) (V)

Args: V is a variable.
Value: Either V or V tailed into an appropriate section.
Description: RDEF obtains the declaration and full name for variables used in contexts other than operator,block or lambda variable, and function or embedded function name, i.e., reference class.

3.2.2.4 (BDEF . 127) (V)

Args: V is a variable.
Value: Either V or V tailed into an appropriate section.
Description: BDEF obtains the declaration and full name for variables bound in either a BLOCK or LAMBDA expression.

3.2.2.5 (BIND . 127) (L,P)

Args: L is a variable binding list from either a BLOCK or LAMBDA form.
P is T if L is from a BLOCK and NIL if L is from a LAMBDA.
Side effects: (SVAR . 127)
(ALIST . 127)
(AC1 . 126)
(AC2 . 126)
Value: The list of the variables in L, tailed when appropriate.
Description: BIND handles lists of variables to be bound. In the case of BLOCKS this list L, may contain explicit presets or implicit presets of NIL. BDEF is used to determine if the name is tailed. The contents of AC are reflected in AC1 and AC2. Lexical variables are added to ALIST. SVAR is given value T if any special variables are in L.

3.2.2.6 (ODEF . 127) (V,K)

Args: V is a variable.
K is the number of arguments V is called with.
Side effects: (KIND . 122).
Value: Either V or V tailed into an appropriate section.
Description: ODEF obtains the declaration and full name for a variable used as a form operator. KIND reflects the declaration and will be given the value "LEXICAL" if V is on ALIST.

3.2.2.7 (TDEF . 127) (V,K)

Args: V is a variable.
K is the number of arguments, "MACRO" or "INSTRUCTION".
Value: V properly tailed.
Description: TDEF makes an appropriate declaration for a macro, instruction, or function being compiled. The tailed name is returned.

3.2.2.8 (FDEF . 127) (V)

Args: V is a variable.
Value: V tailed into an appropriate section.
Description: FDEF obtains the declaration for V used in the context "(FUNCTION v)". The tailed name is returned.

3.2.2.9 (GVAR . 127) ()

Value: "*GUESS*"
Description: Supplies a guessed name for a variable during compilation.

3.2.3 Declarations: Assembler

The following group of functions are used by the assembler for interfacing with the system declaration logic.

3.2.3.1 (LGSD . 126) (N,S,K)

Args: N is an identifier.
S is a section number.
K is a kind of declaration.
Description: The warning message "((n . s) (GUESSED k)" is issued and LMPC is then used to make the declaration of kind K.

3.2.3.2 (LINS . 126) (I)

Args: I is an identifier.
Value: T or NIL.
Description: Using LGDC, a declaration for I in a section from SLST is looked for. If none if found, the value is NIL. Otherwise, the value is T.

3.2.3.3 (LCNT . 126) ()

Side effects: (XLST . 126)
Value: The displacement of the PRS word pointed to by REF from the beginning of PRS space.
Description: If the PRS entry referenced by REF is not already in XLST, it is added. If it is already there, the count of references is incremented.

3.2.3.4 (LRDF . 126) (V)

Args: V is a variable.
Side effects: (R1 . 122)
(DELTA . 122)
Description: LRDF is used for retrieving the definition of variables used as an address field of an instruction. DELTA and R1 are set to reflect appropriate values to be used as decrement and base register. If the reference is a non-local variable, LCNT is used to increment the reference count.

3.2.3.5 (LTDF . 126) (V,K)

Args: V is a variable.
K is the number of arguments, "MACRO", or "INSTRUCTION".
Value: The byte displacement of V's PRS word from the beginning of space.
Description: LTDF is used for retrieving and/or making a declaration for a function, macro or instruction being assembled. The count for this entry is incremented by one.

3.2.3.6 (LFDF . 126) (V,K)

Args: V is a variable.
K is an integer kind number.
Value: The byte displacement of V's, PRS word from the beginning of PRS space.
Description: The declaration for V is retrieved and/or made of kind K. The count for this function is incremented by one. LFDF is used for processing the named function in CALL or CALI pseudo instructions and in (FUNCTION f) types of address fields.

3.2.3.7 (LBDF . 126) (V)

Args: V is a variable.
Value: Either the variable name, V or the byte displacement of V's PRS word from the beginning of PRS space.
Description: LBDF is used for making and/or retrieving V's declaration. V is used, in a BIND pseudo instruction. If V is to be specially bound, its count is incremented by one.

4. THE LISP 1.5 360 COMPILER

4.1 COMPILER: PHILOSOPHY OF OPERATION

The LISP compiler is a fast, one-pass program written in LISP. The LISP input is translated to a computationally equivalent LAP program that may be then assembled.

The compiler acts as if the 360 were a one accumulator machine. All values are left in the accumulator, AC. The compiler keeps track of the value in AC. The variables, (AC1 . 126) and (AC2 . 126) have values which are either NIL, "(NIL)" or a variable name. The value NIL indicates the compiler doesn't know the contents of AC. The value "(NIL)" indicates that the value of AC is NIL. A variable name indicates that the value of AC is a copy of the value of the named variable.

Example:

If (SETQ X (SETQ Y (CAR Z)) is compiled, the value of AC1 would be "X" and the value of AC2 would be "Y". This scheme allows the compiler to remember the contents of AC "two deep".

Labels generated by the compiler are dotted pairs. The first element of the pair is a unique, integer label name. The second element of the pair is the number of references to the label, initially zero. Each use of the label causes the count to be incremented by one. The final count is used by the compiler optimization logic to determine whether the contents of AC may be remembered when the label is attached to the LAP listing, LST.

The compiler uses the macro/instruction mechanism for compiling special forms. When a form is compiled, the declaration for its operator is retrieved. If it is a macro, the functional associated with the operator is applied to the form being compiled. The value of this macro expansion is then compiled in place of the original form. If the declaration of the form is instruction, then the functional associated with this operator is executed and is expected to complete the entire job of compiling this form.

The compiler divides the contexts of compilation into five modes; statement, terminal statement, expression, terminal expression, and predicate. Statements are those forms used for side effect, not value. The top level of a PROG or BLOCK is a list of statements. Examples are (GO L), (PRINT X). A terminal statement is a form used as a statement where the semantics of LISP require an implicit transfer of program control after evaluations of the statement. In the conditional statement (COND (X(PRINT Y)) ....), (PRINT Y) is a terminal statement. A transfer to the end of the conditional statement must be added to the generated code after the call to PRINT.

Expressions are forms used for value. Examples of expressions are X in (CAR X) or (FN Z) in (BLOCK((Y (FN Z)))...). A terminal expression is a form used as an expression where the semantics of LISP require an implicit transfer of program control after evaluation of the expression. In the conditional expression (COND(X (CAR X))...), (CAR X) is a terminal expression. A transfer to the end of the conditional expression must be added to the generated code after the call to CAR. Another example of a terminal expression is the expression body of a RETURN statement. After evaluation of the expression, control must be transferred to the end of the BLOCK.

Predicates are a special kind of expression. They are expressions used not for their value, but for conditional code placement. In this conditional form, (COND(P X)...). P is a predicate. If the value of P is NIL, program control is transferred to the next clause, otherwise control falls through. This is an example of a one-way conditional branch. Consider the example; (COND((COND(X P)...) Y)...). The predicate P is evaluated. If the value is NIL, then a branch to the next phrase of the outer COND is executed; otherwise control branches to the evaluation of Y. This is an example of a two-way conditional branch.

Five functions; CSTA, CTST, CEXP, CTXP and CPRD comprise the contextual top drivers in the compiler. They are used for compiling forms in statement, terminal statement, expression, terminal expression, and predicate modes respectively. The five functions indicate the mode of compilation to the rest of the compiler by the values bound to several special variables. These variables, and arguments to these functions are summarized in the following table:

Function	Arguments	ELAB	SLAB	TGO	FGO	SCLS	PCLS
CSTA(X)	X is a form	-	NIL	-	-	T	-
CTST(X,SL)	X is a form SL is a label	-	SL	-	-	T	-
CEXP(X)	X is a form	NIL	-	-	-	NIL	NIL
CTXP(X,EL)	X is a form EL is a label	EL	-	-	-	NIL	NIL
CPRD(X,TL,FL)	X is a form TL is a label FL is a label	-	-	TL	FL	NIL	T

- indicates no binding or change of values of the variable.

The variables, ELAB, SLAB, TGO, SCLS, and PCLS are all in section 127. The functions bind the indicated variables to the value shown and pass the form to be compiled, X, to the function (COMP . 127). It should be noted that the mode of compilation may be unambiguously deduced from the values of the special variables.

COMP is the central compiler function that controls recursion. COMP has one argument, a form to be compiled. This form is bound to the special variable (EXP . 127). If the form to be compiled is a variable or datum, COMP finishes the compilation using the functions (CVAR . 127) or (CDAT . 127) respectively. If the form is a macro, then COMP applies itself to the value of the expansion of the form. If the form is an instruction, the instruction is invoked to continue the compilation. The only legitimate cases not yet covered are function and functional calls; these are handled by COMP using the functions (FNCL . 127) and (FTCL . 127) respectively.

When instructions are invoked, they may compile the arguments of the form in EXP using any of the five top driver functions (e.g., CEXP, ..).

A standard set of conditions in the compilation process is defined as follows: If the mode is statement or terminal statement, the code for the form being compiled has been generated. In other cases, code for the form has been generated that leave a value in AC. When these standard conditions exist, the function (RESP . 127) may be used to complete compilation by responding to mode information. RESP performs the following task: If the mode is terminal statement, attach an unconditional branch to the label in SLAB. If the mode is terminal expression, attach an unconditional branch to the label in ELAB. If the mode is predicate, attach a branch on NIL AC (BZM) to the label in FGO and attach an unconditional branch to the label in TGO.
If the mode is neither terminal statement, terminal expression or predicate, RESP acts as a NOP.

Any of the variables which have labels as values (SLAB, ELAB, FGO, TGO) may have value NIL. When NIL is the value, generation of branches for that label is suppressed. (This indicates the fall through for one-way predicates.)

The interrelationships of the compiler functions is illustrated by Figure X.

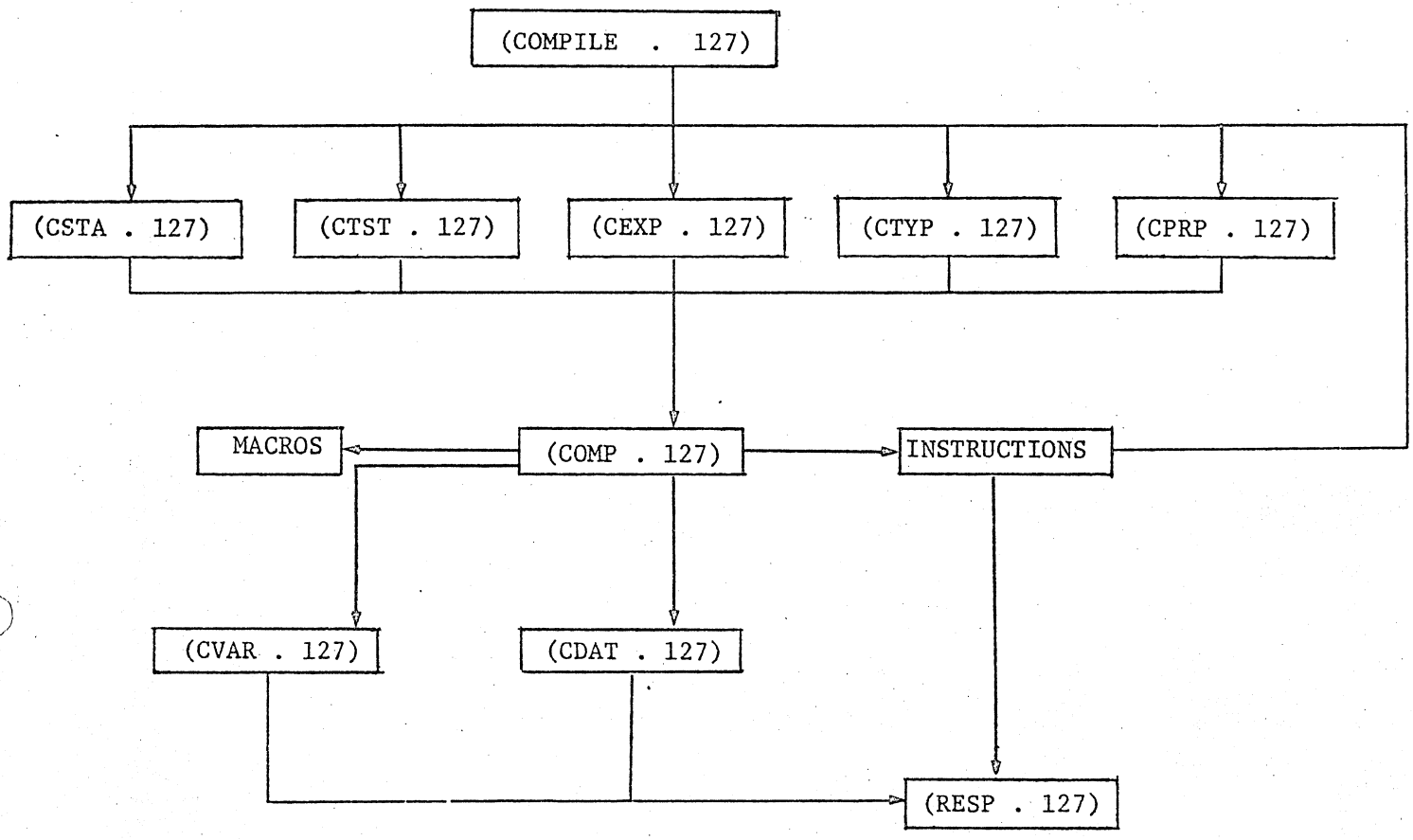

(COMPILE . 127)
(CSTA . 127)
(CTST . 127)
(CEXP . 127)
(CTYP . 127)
(CPRP . 127)
MACROS
(COMP . 127)
INSTRUCTIONS
(CVAR . 127)
(CDAT . 127)
(RESP . 127)

Figure X.

4.2 COMPILER: IMPLEMENTATION

4.2.1 Compilation Control and Central Functions

The following group of functions are the controllers of the compilation process.

4.2.1.1 (COMPILE . 127) (N,K,A,B)

Args: N is the name of the function, macro or instruction to be compiled.
K is the kind of entity to be compiled, i.e., MACRO, INSTRUCTION or integer detailing information about the number of arguments for a function.
A is the argument list.
B is the expression body of the entity to be compiled.

Bindings: (FNAM . 127) is bound to the name of the entity being compiled.
(AC1 . 126)
(AC2 . 129)
(SVAR . 127)
(GL . 127) is bound to Ø
(LST . 127)

Value: The LAP listing resulting from this compilation.

Description: Performs the compilation and declarations for this entity.

4.2.1.2 (CEXP . 127) (E)

Args: E is a form.

Bindings: (SCLS . 127)
(PCLS . 127)
(ELAB . 127)

Description: E is compiled in the expression mode using (COMP . 127).

4.2.1.3 (CTXP . 127) (E,L)

Args: E is a form
L is a compiler-generated label.

Bindings: (ELAB . 127) is bound to L.
(PCLS . 127)
(SCLS . 127)

Description: E is compiled in the terminal expression mode with L as the confluence point by using (COMP . 127).

4.2.1.4 (CSTA . 127) (S)

Args: S is a form.

Bindings: (SLAB . 127)
(SCLS . 127)

Description: S is compiled in the statement mode using (COMP . 127).

4.2.1.5 (CTST . 127) (S,L)

Args: S is a form.
L is a compiler-generated label.
Bindings: (SLAB . 127) is bound to L.
(SCLS . 127)
Description: S is compiled in the terminal statement mode with L as the confluence point by using (COMP . 127).

4.2.1.6 (CPRD . 127) (P,TL,FL)

Args: (TGO . 127) is bound to TL.
(FGO . 127) is bound to FL.
(SCLS . 127)
(PCLS . 127)
Description: P is compiled in the predicate mode using (COMP . 127). TL and FL give the confluence points for true and false evaluation of P respectively.

4.2.1.7 (COMP . 127) (E)

Args: E is a form.
Bindings: (EXP . 127) is bound to E.
Description: COMP is the master switch for recursion in the compilation process. Macros are expanded, LAMBDA forms are converted to BLOCK forms by MBLK, functional and functional calls are made by FTCL, FNCL, and FIDF, variables and constants are compiled using CVAR and CDAT, and instructions are invoked for code generation.

4.2.1.8 (MACX . 127) (E)

Args: E is any form.
Value: A form equivalent to E.
Description: The form E is macro-expanded if its operator is a macro name. Further, if the operator is not in the syntax of a variable, the operator is expanded using MACX.

4.2.1.9 (MBLK . 127) (L,P)

Args: L is a LAMBDA form.
P is a list of presets.
Value: A BLOCK form.
Description: The LAMBDA form L, with variable presets L, is transformed into an equivalent BLOCK form.

4.2.1.10 (RESP . 127) ()

Description: RESP assumes the compilation is in a standard form, i.e., if the mode is not statement or terminal statement, a value is in the accumulator. RESP attaches appropriate branch or conditional branch instruction to satisfy the terminal or predicate modes of compilation.

4.2.1.11 (CVAR . 127) (V)

Args: V is a variable name.
Side Effects: (AC2 . 126)
(AC2 . 126)
Description: When necessary, instructions are attached to copy the value of the variable named by V into the accumulator. AC1 and AC2 are updated to reflect this fact.

4.2.1.12 (CDAT . 127) (D)

Args: D is any atomic datum.
Side Effects: (AC1 . 126)
(AC2 . 126)
Description: Instructions to copy the datum named by D into the accumulator are generated. Appropriate tracks are left in AC1 and AC2.

4.2.2 Function for Building the Compilers Listing

The following group of functions are used by the compiler for attaching instructions to the listing.

4.2.2.1 (ATCH . 127) (I)

Args: I is a LAP instruction or label.
Side Effects: (LST . 127)
Description: I is attached to LST.

4.2.2.2 (ATIN . 127) (O,A)

Args: O is a "compiler" LAP macro name, i.e., P,C, etc.
A is an address field.
Side Effects: (LST . 127)
Description: The instruction (O . A) is attached to LST.

4.2.2.3 (ATBR . 127) (B,L)

Args: B is a "compiler" LAP macro name for a branch instruction, i.e., T, B, etc.
L is a compiler generate label.
Side Effects: (LST . 127)
Description: The instruction, (B . L) is attached to LST. The reference count of the label named by L is increased by one.

4.2.2.4 (ATUL . 127) (L)

Args: L is an identifier label name.
Side Effects: (LBDL . 127)
(LST . 127)
(AC1 . 127)
(AC2 . 127)
Description: The user-defined label named L is attached to LST. The label definition is added to LBDL.

4.2.2.5 (ATLB . 127) (L)

Args: L is a compiler-generated label.
Side Effects: (LST . 127)
(AC1 . 126)
(AC2 . 126)
Description: In principle, ATLB attaches the label named by L to LST. However, several kinds of optimizations are first attempted. For example, if the last instruction in LST references this label, the instructions may be discarded, and the reference count decremented by one. Also if the reference count is or becomes zero, the label is not attached, and therefore the contents of the accumulator are known to be preserved.

4.2.2.6 (ATHL . 127) (I,O,Q)

Description: ATHL is a long, messy, predicate expression used several times by ATLB. A further description would at best be confusing.

4.2.2.7 (GLAB . 127) ()

Side Effects: (GL . 127)
Value: A compiler-generated label.
Description: GLAB generates unique labels for the compiler.

4.2.2.8 (ULAB . 127) (L)

Args: L is a compiler-generated label.
Value: The integer name of L.
Description: The reference count for L is incremented by one.

4.2.2.9 (LBUC . 127) ()

Value: T or F.
Description: This predicate determines whether the last instruction attached to LST is an unconditional branch.

4.2.2.10 (BUCP . 127) (I)

Args: I is a LAP instruction.
Value: T or F.
Description: This predicate determines whether I is an unconditional branch or a BLOCK, whose last instruction is an unconditional branch.

4.2.3 Calling Sequence Generators

The following group of functions are used by the compiler to generate calling sequences.

4.2.3.1 (FCAL . 127) (E,A,I)

Args: E is a form.
A is a list of forms.
I is a LAP instruction, a fast or slow call.
Side Effects: (AC1 . 126)
(AC2 . 126)
Description: FCAL causes compilation of a call to the functional resulting from the evaluation of E with arguments specified by A.

4.2.3.2 (FTCL . 127) (E,A)

Args: E is a form.
A is a list of forms.
Description: FTCL compiles a SLOWCALL linkage, to the function resulting from the evaluation of E, with arguments specified by A, by using (FCAL . 127).

4.2.3.3 (FNCL . 127) (N,A)

Args: N is a function name.
Side Effects: (AC1 . 126)
(AC2 . 126)
Description: A function call to N with arguments specified by A is compiled.

4.2.3.4 (FIDF . 127) (D)

Args: D is a constant form.
Side Effects: (AC1 . 126)
(AC2 . 126)
Description: A function call to a function of an indefinite number of arguments is compiled. The form to be compiled is the value of (EXP . 127). If the call has zero arguments, D is used for the value, e.g., (LIST) = NIL. The call is forced to section 121.

4.2.3.5 (CARG . 127) (A,B)

Args: A is a list of forms.
B is a boolean flag.

Description: The forms in A are compiled. After each, a LAP instruction to move the value to the pushdown stack is attached to the listing. The value of the last argument (if there is one or more arguments) is left in the accumulator if the value of B is F.

4.2.3.6 (FASTCALL . Ø) () INSTRUCTION

Description: The FASTCALL form in (EXP . 127) is compiled.

4.2.3.7 (LIST . Ø) () INSTRUCTION

Description: An appropriate calling sequence is compiled.

4.2.3.8 (MAX . Ø) () INSTRUCTION

Description: An appropriate calling sequence is compiled.

4.2.3.9 (MIN . Ø) () INSTRUCTION

Description: An appropriate calling sequence is compiled.

4.2.3.10 (LOGOR . Ø) () INSTRUCTION

Description: An appropriate calling sequence is compiled.

4.2.3.11 (LOGAND . Ø) () INSTRUCTION

Description: An appropriate calling sequence is compiled.

4.2.3.12 (LOGXOR . Ø) () INSTRUCTION

Description: An appropriate calling sequence is compiled.

4.2.3.13 (PLUS . Ø) () INSTRUCTION

Description: An appropriate calling sequence is compiled.

4.2.3.14 (TIMES . Ø) () INSTRUCTION

Description: An appropriate calling sequence is compiled.

4.2.4 The FOR MACRO

The following description is of the FOR MACRO.

4.2.4.1 (FOR . Ø) (X) MACRO

Args: X is a FOR form.
Value: An equivalent of X expressed as a BLOCK.
Description: See language document for complete description of the expansion.

4.2.5 Compilation of Predicate Forms

The following group of functions are used for compiling predicate forms.

4.2.5.1 (NOT . Ø) () INSTRUCTION

Description: The true and false terminal labels, TGO and FGO, are reversed and compilation continues on the embedded form.

4.2.5.2 (NULL . Ø) () INSTRUCTION

Description: The true and false terminal labels, TGO and FGO, are reversed and compilation continues on the embedded form.

4.2.5.3 (AND . Ø) INSTRUCTION

Description: The AND form is compiled using (CBOL . 127).

4.2.5.4 (OR . Ø) INSTRUCTION

Description: The OR form is compiled using (CBOL . 127).

4.2.5.5 (EQ . Ø) INSTRUCTION

Description: LAP code for the EQ form is generated. Special, optimized code is generated when either embedded form is a constant or, when the last embedded form is a variable.

4.2.5.6 (PRDP . 127) ()

Value: T or F.
Description: PRDP is a semi-predicate. If the mode of compilation is predicate, the value of PRDP is F. Otherwise the value is T and, the form, EXP, is compiled in a modified mode. PRDP is used by NOT, NULL, OR, AND and EQ to assure proper compilation.

4.2.5.7 (CBOL . 127) (B)

Args: B is either T or F.
Description: CBOL compiles AND and OR forms with the value of B being T or F respectively. The main chore is optimizing the choice of arguments to CPRD for compilation of the embedded forms in the predicate mode.

4.2.6 Compilation of Forms Cognizant of Terminals

The following group of functions are used for compiling forms that cause transitions into the terminal statement or terminal expression modes.

4.2.6.1 (SELECTQ . Ø) () INSTRUCTION

Bindings: (AC1 . 126)
(AC2 . 126)
Side Effects: (EXP . 127)
Description: The SELECTQ form is compiled. Where necessary, appropriate terminal labels are generated and placed on the LAP listing. Care is taken to preserve the proper contents of the accumulator for optimization of generated code.

4.2.6.2 (SELECT . Ø) ()

Side Effects: (EXP . 127)
Description: The SELECT form is compiled. Where necessary, appropriate terminal labels are generated. Care is taken to generate optimized code wherever either the comparand or comparor is a constant. (SHLP . 127) is used as a large common subexpression by SELECT.

4.2.6.3 (COND . Ø) INSTRUCTION

Side Effects: (EXP . 127)
Description: COND handles the case of no embedded forms with an appropriate call to CONDERR. In other cases, (KSTA . 127), (KPRD . 127), or (KEXP . 127) are used to complete compilation of the COND form.

4.2.6.4 (KHLP . 127) ()

Value: A compiler-generated label.
Description: KHLP is used for compiling the predicates embedded in a COND form. The confluence point label used for false evaluation of the predicate is returned.

4.2.6.5 (KLOB . 127) (L,P)

Args: L is a partial LAP listing in reverse order.
P is a compiler generated label.
Side Effects: (AC1 . 126)
(AC2 . 126)
Description: If neither the reference count of P is zero, nor the reference count is one and the last instruction in L is a branch to P, the remembered contents of the accumulator is clobbered. In all other cases, KLOB is equivalent to a NOP.

4.2.6.6 (KSTA . 127) ()

Bindings: (AC1 . 126)
(AC2 . 126)
Side Effects: (EXP . 127)
Description: KSTA compiles COND forms encountered in the statement or terminal statement mode. Appropriate terminals and predicate labels are generated and placed on the LAP listing. Care is taken to properly map the contents of the accumulator along branch paths to the generated labels.

4.2.6.7 (KEXP . 127) ()

Bindings: (AC1 . 126)
(AC2 . 126)
Side Effects: (EXP . 127)
Description: KEXP compiles COND forms encountered in the expression or terminal expression mode. Appropriate terminal and predicate labels are generated. Care is taken to properly map the contents of the accumulator along branch paths to the generated labels.

4.2.6.8 (KPRD . 127) ()

Bindings: (AC1 . 126)
(AC2 . 126)
Side Effects: (EXP . 127)
Description: KPRD compiles COND forms encountered in the predicate mode. Appropriate predicate labels for both the COND form and embedded predicates are generated. Care is taken to properly map the contents of the accumulator along branch paths to the generated labels.

4.2.6.9 (SHLP . 127) (I,U,V)

Args: I is a LAP instruction.
U and V are compiler-generated labels.
Bindings: (AC1 . 126)
(AC2 . 126)
Description: SHLP is a large, common sub-expression used by the instruction, SELECT. The comparison and branch logic is compiled. Care is taken to properly map the contents of the accumulator along branch paths.

4.2.6.10 (BLOCK . Ø) () INSTRUCTION

Bindings: (LBRL . 127)
(LBDL . 127)
(SVAR . 127)
(ALIST . 127)
(LST . 127)
Side Effects: (LST . 127)
Description: BLOCK compiles the BLOCK form in EXP. Bindings are made using BIND. Either BSTA, BPRD or BEXP is used to compile the statements. Error diagnostics for undefined labels are issued.

4.2.6.11 (RETURN . Ø) () INSTRUCTION

Bindings: (LBRL . 127)
(LBDL . 127)
Description: Code for the RETURN form in EXP is compiled.

4.2.6.12 (PROG . Ø) (P) MACRO

Args: P is a PROG form.
Value: A BLOCK form equivalent to P.

4.2.6.13 (BEXP . 127) ()

Bindings: (ELAB . 127)
(SVAR . 127)
(LBCK . 127)
Value: A list of branch instructions or NIL.
Description: The BLOCK form in EXP is compiled as an expression. If the compilation is not in terminal expression mode, an appropriate label is generated and compilation is put in that mode. When necessary, a (RETURN NIL) statement is forced as the last statement of the BLOCK.

4.2.6.14 (BSTA . 127) ()

Bindings: (SLAB . 127)
(SVAR . 127)
(LBCK . 127)
(BLAB . 127)
Value: A list of branch instructions or NIL.
Description: The BLOCK form in EXP is compiled as a statement. If the compilation is not in terminal statement mode, an appropriate label is generated and compilation is put in that mode.

4.2.6.15 (BPRD . 127) ()

Bindings: (ELAB . 127)
(LBCK . 127)
(BLAB . 127)
Value: A list of branch instructions or NIL.
Description: The BLOCK form in EXP is compiled as a predicate. Appropriate labels are generated, and when necessary a (RETURN NIL) statement is forced as the last statement of the BLOCK.

4.2.6.16 (SMAP . 127) ()

Value: A statement or NIL.
Description: SMAP is used to compile the list of forms in the BLOCK form in EXP. The last form, if it is not a label, is returned.

4.2.7 The PROGN Logic

The following group of functions are used for generating code for PROGNs, both implicitly and explicitly written.

4.2.7.1 (PROGN . Ø) () INSTRUCTION

Description: The PROGN form in EXP is compiled using BRKT.

4.2.7.2 (MPGN . 127) (L)

Args: L is a list of forms.
Value: A PROGN form with the forms in L as the body.
Description: Converts a list of forms to a PROGN.

4.2.7.3 (BRKT . 127) (L)

Args: L is a list of forms.
Description: The list of forms in L are compiled in statement mode if the mode of compilation was in either the statement or terminal statement mode, and are compiled in expression mode in all other cases. The last form in L is compiled with no mode change whatsoever. If L is NIL, then NIL is compiled.

4.2.8 Miscellaneous Compiler Forms

The following is a group of miscellaneous forms used for compilation.

4.2.8.1 (GO . Ø) () INSTRUCTION

Side effects: (LBRL . 127)
Description: An appropriate branch instruction is generated. If the referenced label has not been defined, it is added to LBRL.

4.2.8.2 (LABEL . Ø) () INSTRUCTION

Description: The defined label is attached to the listing and the embedded statement is compiled.

4.2.8.3 (QUOTE . Ø) () INSTRUCTION

Description: The QUOTE form is compiled using CDAT.

4.2.8.4 (SETQ . Ø) () INSTRUCTION

Side Effects (AC1 . 126)
(AC2 . 126)
Description: Optimized code for the SETQ form in EXP is compiled. Care is taken to use the remembered contents of the accumulator, and when possible, update the contents to reflect new, known information.

4.2.8.5 (FUNCTION . Ø) () INSTRUCTION

Side Effects: (FCNT . 127)
(AC1 . 126)
(AC2 . 126)

Description: If the embedded form is a LAMBDA form, it is compiled under a compiler generated name. If the embedded form is a function name, the appropriate code to convert a function to a functional is generated.

4.2.9 Machine Dependent Forms Compilation

The following group of functions are used to generate code for machine dependent, non-LISP use.

4.2.9.1 (LEFT . Ø) () INSTRUCTION

Description: Code to access the left 16 bits of the argument is compiled using WPRT.

4.2.9.2 (RIGHT . Ø) () INSTRUCTION

Description: Code to access the second from the left group of 16 bits of the argument is compiled using WPRT.

4.2.9.3 (THIRD . Ø) () INSTRUCTION

Description: Code to access the third from the left group of 16 bits of the argument is compiled by WPRT.

4.2.9.4 (FOURTH . Ø) () INSTRUCTION

Description: Code to access the fourth from the left group of 16 bits of the argument is compiled by WPRT.

4.2.9.5 (WPRT . 127) (I)

Args: I is a LAP instruction.
Side Effects: (AC1 . 126)
(AC2 . 126)

Description: WPRT generates code to access a group of 16 bits (a half-word) from memory and convert it to a 24 bit symbolic pointer.

4.2.9.6 (CODE . Ø) () INSTRUCTION

Side Effects: (AC1 . 126)
(AC2 . 126)

Description: CODE attaches the instructions comprising the body of the form in EXP to the LAP listing.

4.2.9.7 (ADDSMALL . Ø) () INSTRUCTION

Description: The ADDSMALL form in EXP is compiled using ADHL.

4.2.9.8 (SUBSMALL . Ø) () INSTRUCTION

Description: The SUBSMALL form in EXP is compiled using ADHL.

4.2.9.9 (ADHL . 127) (B)

Args: B is either T or F.
Side Effects: (AC1 . 126)
(AC2 . 126)
Description: Optimized code is generated for the small integer arithmetic form in EXP. B indicates whether addition or subtraction is to be performed.

5. THE LISP 1.5 360 ASSEMBLER, LAP

5.1 LAP: PHILOSOPHY OF OPERATION

LAP is an acronym for LISP Assembly Program and is also the name of the language assembled. LAP is a one-pass assembler that handles the entire set of the IBM 360's order codes and in addition provides for several pseudo-instructions and a generalized macro capability. With the exception of a few functions, the entire assembler is coded in the LISP language. The other functions are themselves coded in LAP.

LAP is a one-pass assembler allowing forward references to labels, symbolic naming of variables, both temporary and global, automatic handling and allocation of the pushdown stack, block structuring and an interface with the full declarative logic of the system. The function being assembled is assembled as if its name were (**** . 126). At the completion of assembly, an appropriate declaration is made and the binary image is transferred and linked with the proper PRS word. **** is initialized to 4096 bytes but is shrunk to the correct length before the PRS relinking. If any errors (not warnings) are detected during assembly, the binary image does not become usable and the reference counts to external items are not incremented. This allows for proper excising. When an identifier op code is used, the function INSP obtains a value for that identifier as if it were a variable tailed into section 125. This value may be either a number or a functional. If it is a functional, the functional is called with the instruction to be assembled as an argument. The value of this call is a list of LAP instructions (either 360, pseudo or macro-instructions). The instructions in this list are now compiled instead of the original. This is the macro capability in LAP. Pseudo-instructions are handled through this same mechanism. The following is a list of pseudo-ops and the function performing the expansion:

Pseudo-Op	Function
PUSH.	(PUSH. . 126)
POP.	(POP. . 126)
ARGS	(ARGS . 126
BLOCK	(RAP . 126)
CALL	(CALL . 126)
CALI	(CALL . 126)
FASTCALL	(CALL . 126)
SLOWCALL	(CALL . 126)

The following is a list of op codes output by the compiler that are assembled using the macro expansion facility. With each op code is listed the function used for expansion:

Op Code	Function
G	(LGPR . 126)
Z	(LGPR . 126)
P	(LGPR . 126)
R	(LGPR . 126)
U	(LAPB . 126)
E	(LAPB . 126)
I	(LAPB . 126)
F	(LAPB . 126)
B	(LAPB . 126)
GO	(LGO . 126)
BZM	(BGZM . 126)
BGZ	(BGZM . 126)

If the value of the declaration found in section 125 is a number, the value of the op code for this instruction may be computed as the remainder of the number divided by 256. The quotient of the number divided by 256 indicates the format of the instruction being assembled. A list of format patterns dotted with an identifying number is kept as the value of the variable (PLST . 216). The format pattern is a list of the simple patterns:

Simple Patterns	Explanation
REG	A four bit integer or register mnemonic.
MASK	A four bit integer or mask mnemonic.
ADR	An address that consists of two registers and a displacement.
HADR	An address that consists of one register and a displacement.
SK	Ignore rest of half-word being assembled.
HB	A four bit integer.
BY	An eight bit integer.

Each 360 instruction is described by one of eleven patterns in PLST. The pattern matching instruction assembling is handled by the function IMCH.

Forward references to labels are handled by having a linked chain of references kept in the displacement portion of the referencing instructions in ****. The displacement of the original instruction is set to zero to indicate the end of the chain. The displacement of the last instruction to reference the label is kept in LBRL, a list of unsatisfied labels and link locations.

The assembler uses several functions to plant in the binary image, ****:

Function	Action
(GETHW . 126)	Gets-half-word (16 bits) from ****
(SETHW . 126)	Sets half-word (16 bits) in ****
(DINK . 126)	Advances location counter to next half-word.
(PLHW . 126)	Plants half-word (16 bits) in ****
(PLBY . 126)	Plants a byte (8 bits) in ****
(PLTB . 126)	Plants twelve bits in ****
(PLHB . 126)	Plants a half-byte (4 bits) in ****

The value of the variable, DPDP, gives the byte-displacement of the last active word put on the pushdown stack. Various pseudo-ops (e.g., PUSH) and address fields (e.g., TOP.) either modify DPDP and/or use DPDP as the displacement field for instructions. RAP (which handles BLOCK expansions) matches the variables in the BIND list with locations at and behind DPDP and adds these definitions to ALIST or uses them for generating the special variable bind/unbind code.

5.2 ASSEMBLER: IMPLEMENTATION

5.2.1 Assembly Control Functions

The functions described in this section comprise the top end control logic of the LAP assembler.

5.2.1.1 (LAP360 . 126) (X)

Args: X is a LAP form to be assembled.

Bindings:
(FNAM . 127)
(HB . 126)
(LOC . 126
(CLST . 126)
(XLST . 126)
(LBRL . 127)
(LBDL . 127)
(ALIST . 127)
(DELTA . 122)
(R1 . 122)
(R2 . 122)
(REF . 122)
(V360 . 122)
(KIND . 122)
(DPDP . 126)

Value: A list of four items: The tailed name of the assembled function, the size in bytes of the assembled function, a pointer to the PRS word for the assembled function, and a list of dotted pairs of pointers to PRS words for external references and reference counts from the assembled function.

Description: LAP360 assembles the form X. DPDP is set to a value dependent upon the number of arguments. The major portion of the assembly is accomplished using RAP by wrapping the code in a dummy BLOCK. Forward references to labels are satisfied and the return code is placed in the assembled binary program image. The size of the image is adjusted to be a multiple of four bytes. An appropriate declaration is made for the the image just assembled. LAP 360 may not be entered recursively.

5.2.1.2 (RAP . 126) (L)

Args: L is a BLOCK LAP form.
Bindings: (ALIST . 127)
(DPDP . 126)
Value: NIL
Description: RAP assembles LAP BLOCK forms. Necessary code for binding and unbinding Special variables is generated. ALIST is agumented with the name of lexical variables bound in this BLOCK. The instructions in the BLOCK are assembled using INSP. RAP may be entered recursively.

5.2.1.3 (INSP . 126) (I)

Args: I is a LAP instruction or label.
Side Effects: (LBDL . 127)
Description: If I is an Atom, a label definition is added to LBDL. If I is a Macro instruction, then it is expanded and the result assembled. Otherwise I is an ordinary 360 instruction and is assembled by IMCH.

5.2.1.4 (DECL . 126) (X)

Args: X is a list of variables
Value: A list of variables.
Description: If X is NIL then the value is "(RET.)". Otherwise the value is a copy of X with "RET," inserted as the next-to-last variable in the list. DECL is used by LAP 360 in dummying up a BLOCK for RAP.

5.2.2 Assembler Pattern Matching Functions

The following group of functions comprise the pattern-matching, instruction assembler portion of LAP.

5.2.2.1 (LADR . 126) (X)

Args: X is an ADR or HADR.
Side Effects: (R1 . 122)
(R2 . 122)
(DELTA . 122)
(DPDP . 126)

Description: LADR converts address fields of instructions being assembled into displacement (DELTA), base and index register (R1, R2) fields. The kinds of addresses handled are: variable, PUSH., POP., LABEL, QUOTE, FUNCTION, ENTRY, NUMBER, TOP., (displacement), (displacement, register), and (displacement, register, register). When LADR has an option, R1 is given a non-zero value and R2 is given a zero value.

5.2.2.2 (LAPR . 126) (X)

Args: X is a register mnemonic or number.
Description: The four-bit integer equivalent of X is placed in **** by PLHB.

5.2.2.3 (LAPM . 216) (X)

Args: X is a mask mnemonic or number. The four bit integer equivalent of X is placed in **** by PLHB.

5.2.2.4 (LAPH . 126) (X)

Args: X is a HADR
Description: The HADR, X, is broken apart by LADR. The four-bit integer value of R1 and the twelve-bit integer value of DELTA are placed in **** by PLHB and PLTB respectively.

5.2.2.5 (LAPA . 126) (X)

Args: X is an ADR.
Description: The ADR, X, is broken apart by LADR. The four-bit integer values of R1 and R2, and the twelve-bit integer value of DELTA are placed in **** by PLHB and PLTB respectively.

5.2.2.6 (REGD . 126) (R)

Args: R is a register mnemonic or number.
Value: The numeric equivalent of R.

5.2.2.7 (IMCH . 126) (S,P)

Args: S is the CDR of a 360 instruction to be assembled.
P is a pattern describing the legal format of S.
Description: P is a list of the following pattern parts; REG, ADR, MASK, HADR, HB, BY, and SK which match, in a one for one manner items in the list S. The matched items are planted in core by LAPR, LAPA, LAPM, LAPH, PLHB, PLBY and DINK respectively.

5.2.3 Assembler Macros

The following group of functions are used for macro expansions of system-provided LAP language forms.

5.2.3.1 (PUSH. . 126) (X)

Args: X is a PUSH. pseudo-instruction.
Side Effects: (DPDP . 126)
Value: NIL
Description: DPDP is incremented to allocate the appropriate number of bytes on the pushdown stack.

5.2.3.2 (POP. . 126) (X)

Args: X is a POP. pseudo-instruction.
Side Effects: (DPDP . 126)
Value: NIL
Description: PDPD is decremented to return the appropriate number of bytes to the pushdown stack.

5.2.3.3 (ARGS . 126) (X)

Args: X is an ARGS pseudo-instruction.
Side Effects: (CLST . 126)
Value: NIL
Description: The present value of DPDP is added to CLST.

5.2.3.4 (BGZM . 126) (I)

Args: I is either a BZM or BGZ pseudo-form.
Value: An equivalent 360 instruction.
Description: (BZM X(LABEL L)) is (BXLE X 1 (LABEL L))
(BGZ X(LABEL L)) is (BXH X 1 (LABEL L))

5.2.3.5 (CALL . 126) (C)

Args: C is either a CALL, CALI, FASTCALL or SLOWCALL pseudo-instruction.
Side Effects: (DPDP . 126)
(CLST . 126)
Value: NIL
Description: CALL plants in **** the appropriate code to make subroutine linkage. The stack increment is computed. A declaration, consistent with this use is computed. The pushdown pointer, DPDP, is adjusted to its value before this calling sequence and CLST has its top member popped.

5.2.3.6 (LGPR . 126) (I)

Args: I is either a G, P, R or Z pseudo-instruction.
Value: NIL
Description: The G, P, R, and Z pseudo-forms are used by the compiler for generating L, ST, C to register AC, and C to register ZERO instructions respectively.

5.2.3.7 (LAPB . 126) (I)

Args: I is either a U, E, B, T or F pseudo-instruction.
Value: NIL
Description: The U, E, B, T, and F pseudo-forms are used by the compiler for generative (BC U L), (BC E L), (BC A L), (BGZ AC L) and (BZM AC L) instructions respectively. "L" is a label.

5.2.3.8 (LGO . 126) (I)

Args: I is a GO pseudo-form.
Value: (GO L) is equivalent to (BC A (LABEL L)).

5.2.4 Assembler Planter Functions

The following group of functions are used to interface the system with non-LISP data structures.

5.2.4.1 (GENT . 126) (E)

Args: E is an entry name.
Side Effects: (DELTA . 122)
(R1 . 122)
Description: GENT finds the location of the entry E.
DELTA is set to the byte displacement for E, relative to SORG. R1 is set to SORG.

5.2.4.2 (ENTRY . 126) (L)

Args: L is a list of entry definition information.
Value: A list of pairs of entry names and locations relative to SORG.
Description: Each entry definition in L is processed using ENTRY1.

5.2.4.3 (ENTRY1 .126) (E)

Args: E is an entry definition.
Side Effects: (ENTL . 126)
(ELST . 126)
Value: A pair of the entry name and relative locations to SORG.
Description: The entry definition, E, is either an identifier entry name, implying a length of one word, or a list of entry name and length of the entry in words. The definition is put on ELST and ENTL is appropriately updated.

5.2.4.4 (LGLA . 126) (L)

Args: L is a label
Side Effects: (DELTA . 122)
(LBRL . 127)
(R1 . 122)
Description: LGLA is used by the assembler to process label references. R1 is made CB and an appropriate DELTA is computed. When necessary, this reference is either added to, or a reference chain is started in LBRL.

5.2.4.5 (GETHW . 126) (L) LAP

Args: L is an even, non-negative LISP integer smaller than 4096.
Value: A sixteen bit LISP integer.
Description: The half-word in relative location L of the function is converted to a LISP integer. L is the byte address.

5.2.4.6 (SETHW . 126) (L V) LAP

Args: L is an even, non-negative LISP integer smaller than 4096.
V is a LISP number.
Side Effects: (**** . 126)
Description: The number V, is truncated to sixteen bits and placed in the half-word of function **** specified by L. L is the relative byte address.

5.2.4.7 (DINK . 126) () LAP

Side Effects: (LOC . 126)
(**** . 126)
Description: LOC is pointed at the next half-word of ****.
The half-word in **** specified by the updated value of LOC is cleared to zero.

5.2.4.8 (PLHW . 126) (V) LAP

Args: V is a LISP integer.
Side Effects: (**** . 126)
Description: PLHW truncates V to sixteen bits and plants them at the half-word of **** pointed to by LOC. LOC is updated by DINK.

5.2.4.9 (PLBY . 126) (V) LAP

Args: V is a LISP integer.
Side Effects: (**** . 126)
(HB . 126)
Description: PLBY truncates V to eight bits and plants them at the byte of **** pointed to by LOC and HB. HB is updated and, if necessary, LOC is updated by DINK.

5.2.4.10 (PLTB . 126) (V) LAP

Args: V is a LISP integer.
Side Effects: (**** . 126)
(HB . 126)
Description: PLTB truncates V to twelve bits and plants them at the location pointed to by LOC and HB. HB is updated and, if necessary, LOC is updated by DINK.

5.2.4.11 (PLHB . 126) (V) LAP

Args: V is a LISP integer.
Side Effects: (**** . 126)
(HB . 126)
Description: PLHB truncates V to four bits and plants them at the location pointed to by LOC and HB. HB is updated and, if necessary, LOC is updated by DINK.

5.2.4.12 (GNUM . 126) (N)

Args: N is a LISP number.
Side Effects: entry NARY
entry NUML
(DELTA . 122)
(R1 . 122)
Description: GNUM either finds an occurrence of N in NARY or places N at the end of NARY. DELTA is set to the relative byte displacement of the copy of N from SORG. R1 is set to SORG. NARY contains the non LISP numbers in the system created for NUMBER fields of LAP instructions.

6. THE LISP 1.5 360 SUPERVISOR

6.1 SUPERVISOR: PHILOSOPHY OF OPERATION

The LISP supervisor controls the interaction process of the LISP system. Inputs are accepted from a terminal device and appropriately processed by the function, (SUPV . 122). Upon unwraps caused by various errors, SUPV is reentered. For operating files in EDLISP format, the function, (LOADEXP . Ø) is used. Figure Y shows the relationships among the various supervisor functions and control flow to the compiler and assembler. Supervisor functions and variables are customarily in Section Ø or Section 122, the system section. Compiler and Assembler functions and variables are customarily in Sections 127 and 126 respectively. Common functions and variables may be in any of the Sections; Ø, 122, 126 or 127

The supervisors, compiler and assembler's error reporting mechanism is implemented by the use of the special variable (ERRFLG . 122). ERRFLG normally has the value NIL, however, at determination of any error the value is changed to T.

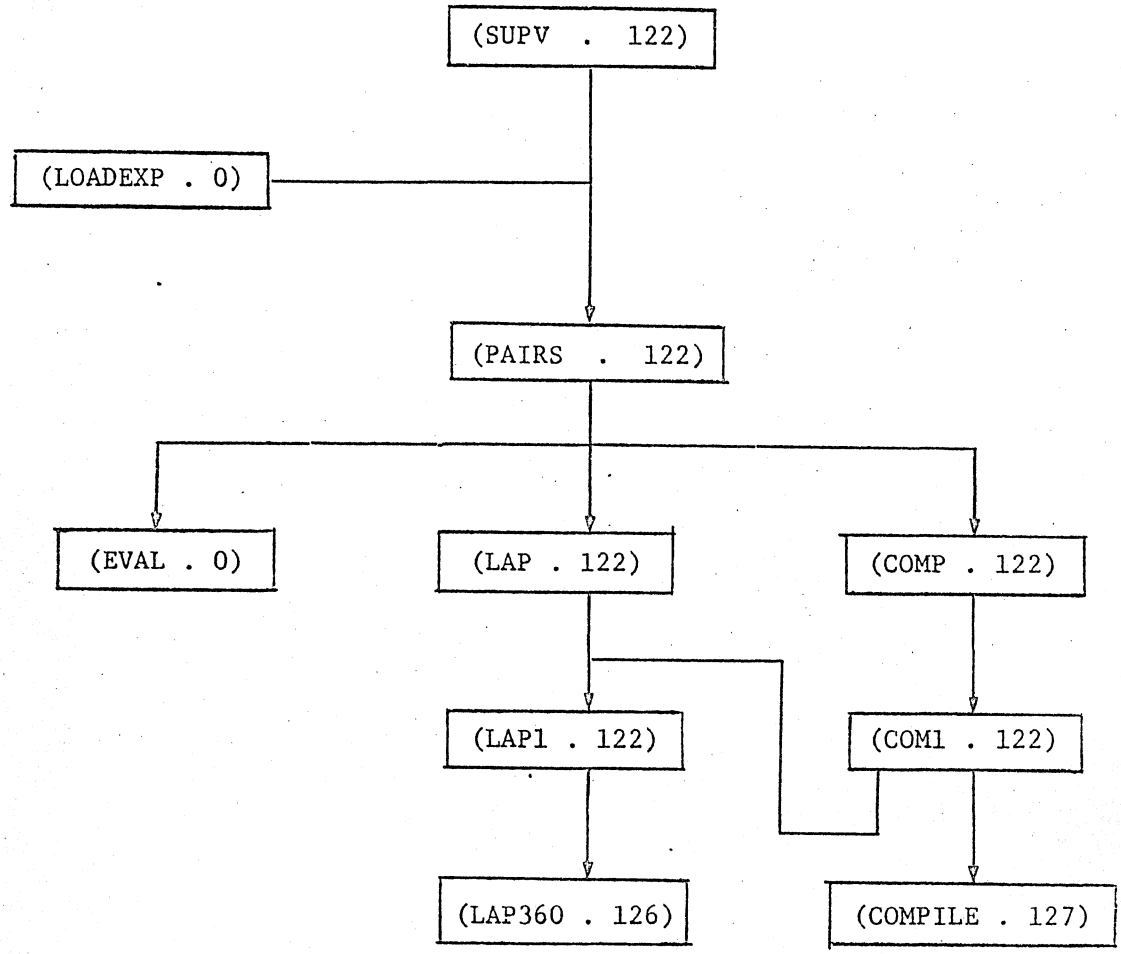

(SUPV . 122)
(LOADEXP . 0)
(PAIRS . 122)
(EVAL . 0)
(LAP . 122)
(COMP . 122)
(LAP1 . 122)
(COM1 . 122)
(LAP360 . 126)
(COMPILE . 127)

Figure Y.

6.2 SUPERVISOR: IMPLEMENTATION

6.2.1 Supervisor Control Functions

The following group of functions comprise the supervisor.

6.2.1.1 (SUPV . 122) ()

Description: Pairs of expressions are read from the file "ITTY" and processed by the function PAIRS. The file "ITTY" is selected before reading but is not automatically used during the evaluation by pairs.

6.2.1.2 (LOADEXP . Ø) (I)

Args: I is the identifier name of an opened input file.
Value: An identifier and several s-expressions.
Description: An s-expression is read from the file I. This s-expression is assumed to be in EDLISP file format, that is as list of an identifier and zero or more evalquote pairs. The value of LOADEXP is the identifier.

6.2.1.3 (PAIRS . 122) (A,B)

Args: A is an operator expression.
B is a list of s-expressions.
Value: The value of the operator A applied to the list of argument B.
Description: PAIRS applies the operator A to the arguments B and outputs this value using MESSAGE. If A is "DEFINE", "MACRO", or "INSTRUCTION", (COMP . 122) is used; if A is LAP, (LAP . 122) is used; otherwise EVAL is used with "QUOTE" wrapped around each argument in B.

6.2.1.4 (EVAL . Ø) (E)

Args: E is a LISP expression.
Bindings: (RRFLG . 122)
Value: The value obtained from E by evaluation.
Description: The form E is compiled under a gensym name, operated and excised.

6.2.1.5 (LAP . 122) (L)

Args: L is a list of LAP definitions
Bindings: (ERRFLG . 122)
Value: A list of entity names for the definitions in L.
Description: The list of definitions in L are assembled by (LAP1 . 122).

6.2.1.6 (COMP . 122) (L,K)

Args: L is a list of LISP definitions.
K is either "MACRO", "INSTRUCTION", or "DEFINE".

Bindings: (ERRFLG . 122)
Value: A list of dotted entity names for the definitions in L.
Description: The list of definitions in L is compiled by (COM1 . 122).

6.2.1.7 (LAP1 . 122) (L)

Args: L is a LAP definition.
Value: The dotted name of the assembled entity.
Description: (GBP1 . 122) is used to initialize ****. (LAP360 . 126) then assembles the entity. If any errors were detected by the assembler, (EXCISEØ . 122) is used on **** and assembly is terminated. Otherwise, the reference count of PRS words referenced by the assembled entity are appropriately incremented and (GBP2 . 122) is used to shorten **** and perform the PRS swap.

6.2.1.8 (COM1 . 122) (C,K)

Args: C is a named lambda form.
K is either "MACRO", "INSTRUCTION" or "DEFINE".
Value: The dotted name of the entity compiled or NIL.
Description: The form, C, is compiled using (COMPILE . 127). If any errors were detected by the compiler, the process is terminated at this point and NIL is returned. Otherwise the compilation is completed with (LAP1 . 122).

6.2.2 Supervisor, Compiler and Assembler Error Mechanism

The following group of functions comprise the error reporting mechanism for the compiler and assembler.

6.2.2.1 (LER2 . 122) (X,Y)

Args: X and Y are s-expressions.
Description: LER2 is used by LAP for output of error diagnostics. X and Y are CONSed and passed to (LERR . 122).

6.2.2.2 (CER2 . 122) (X,Y)

Args: X and Y are any s-expressions.
Description: CER2 is used by the compiler and supervisor for output of error diagnostics. X and Y are CONSed and passed to (CERR . 122).

6.2.2.3 (LERR . 122) (M)

Args: An error message issued by the assembler.
Description: LERR uses CERR.

6.2.2.4 (CERR . 122) (M)

Args: M is an error message issued by the compiler or supervisor.
Side Effect: (ERRFLG . 122)
Description: ERRFLG is set to T. MESAGE is used to output the error message M.

6.2.2.5 (WARN . 122) (M)

Args: M is a warning message.
Description: The warning message, M, is output by MESAGE.

6.2.3 Miscellaneous Service Functions

The following functions perform miscellaneous service functions.

6.2.3.1 (FINDN . 127) (I,L)

Args: I is any s-expression.
L is a list of dotted pairs.
Value: NIL or a pair.
Description: The list L is searched for the first pair whose first element EQs I. The pair so found is returned. If no such pair exists, the value NIL is returned.

6.2.3.2 (MEMB . 127) (I,L)

Args: I is any s-expression.
L is any list.
Value: T or F
Description: Identical to MEMBER, but EQ is used instead of EQUAL.

6.2.3.3 (MQUO . 127) (Q)

Args: Q is any s-expression.
Value: The list of "QUOTE" and Q.
Description: MQUO quotes its argument Q.

6.2.3.4 (CONP . 127) (I)

Args: I is any s-expression.
Value: T or F
Description: Determines if I is an atomic constant (including T or F).

6.2.3.5 (QUOP . 127) (Q)

Args: Q is any s-expression.
Value: T or F.
Description: Determines if Q is a QUOTEd expression.

6.2.3.6 (VARP . 127) (V)

Args: V is any s-expression.

Value: T or F.
Description: Determines if V is a legal variable name in LISP (excludes T and F).

6.2.3.7 (LVP . 127) (V)

Args: V is any s-expression.
Value: T or F.
Description: Determines if V is a legal variable name in LAP (includes T and F).