This document was produced by SDC in performance of contract F 19(628)-5166 with the Electronic Systems Division, Air Force Systems Command, in performance of ARPA Order 773 for the Advanced Research Projects Agency Information Processing Techniques Office.

TECH MEMO *a working paper*

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

TM- 3086/001/00

AUTHOR E. Book D. V. Schorre

TECHNICAL C. Weissman

RELEASE C. Weissman

for J. I. Schwartz

DATE 8/12/66 PAGE 1 OF 29 PAGES

(page 2 blank)

A Higher-Level Machine-Oriented Language as an Alternative to Assembly Language

ABSTRACT

This paper explains our concept of a higher-level machine-oriented language and illustrates it in detail with a description of MOL-32, which is such a language for the Q-32. A compiler for this language has been implemented and is being used in our research to write library routines for the META compiler; the MOL-32 compiler will not be released for general use.

1. INTRODUCTION

This document is not intended as a user's manual but rather as an explanation of a part of the work done in extending META compiler techniques. In a higher-level machine-oriented language, the operations and types of data are the same as those of the machine, but the format of the language is similar to that for a procedural compiler language, such as ALGOL or JOVIAL. Henceforth, machine-oriented language is referred to as MOL. Arithmetic calculations are written in the form of assignment statements. The flow of control is handled by Boolean expressions together with if statements, for statements, and loop statements. Direct code is allowed to give the user complete control over the machine.

The reason for using assembly language, as opposed to a machine-independent language, is that (1) the efficiency of the resultant program is of prime importance and (2) the program cannot be expressed naturally in a machine-independent language, to wit, recursive subroutines in JOVIAL or fixed-point arithmetic in FORTRAN.

Most of the programming which is now being done in assembly language could be done in a higher-level MOL. At present we are using MOL-32 to write library routines rather than entire programs. Since the purpose of these routines is to store and retrieve information in a manner that is efficient for the Q-32, they could not have been implemented in a machine-independent language. This means parts of the syntax of MOL-n would be changed if it were implemented for computer m.

2. DESCRIPTION OF MOL-32

A program written in MOL-32 consists of a declaration followed by a sequence of procedures and ended by the word .STOP. Blanks are ignored, except within strings. Let us get the flavor of the language by examining a sample procedure. The purpose of the procedure shown in Figure 1 is to read a line from teletype and to unpack it into an area specified as a parameter. A flow chart is given to assist in explanation. In actual practice, flow charts are unnecessary, because the flow of control is graphically expressed by conventions of indentation.

In the procedure shown in Figure 1, there are several reserved words. These words are listed below:

.LOCAL	.EXIT	.FOR	.THEN	.IF
.FROM	.END	.RETURN	.ELSE	

All reserved words of the language begin with a period so that the user does not have to worry if he is using a reserved word for one of his identifiers. This is especially important because we are continually adding new reserved words to the language.

2.1 SUBROUTINE LINKAGE

Parameters are passed to subroutines by means of a calling sequence. This means that parameters are supplied in consecutive words after the instruction that branches to the subroutine. Usually these parameters are addresses which are set up at compile time. The subroutine being called uses these addresses to obtain a value or as a location into which to store a value. Literal integers are also passed by putting them directly in the calling sequence. Another type of parameter which is often passed to a subroutine is a string. This consists of one word containing the number of characters in the string, followed by the characters of the string packed eight per word. Actually, any type of data can be put in a calling sequence so long as the routine being called has instructions to pick it up correctly.

Now look at Figure 1. The name of this procedure is TTYIN. It has one parameter, which is the address of the first word of a 72-word block into which the typed line is to be read, one character per word in the rightmost byte. A future version of MOL-32 will allow the names of formal parameters to be written within the parentheses which follow the name of the procedure being defined. The current compiler requires the user to write instructions to pick up these parameters. In Figure 1A these instructions appear in line 34.

```
31    TTYIN():  .LOCAL BUF(10), AREA, T,
32          ITLTY :=('META  16TNSTAT00MOVE  66TELTYP  INPUT 4 COREIX1',
33          #BUF);
34          AREA := [.EXIT];  .EXIT := .EXIT+1;
35          SPEAK(''); <LDA (ITLTY)R; BUC 202;>
36          $I := AREA;  $J := #BUF;
37          AR1 := AREA + 72;  EXPLODEX(); T := 0;
38          .FOR $I .TO 71:
39               .IF T = 0
40                    .THEN .IF [AREA+$I] = 63
41                         .THEN T := 1;  [AREA+$I] := ' '; .END
42                    .ELSE [AREA+$I] := ' '; .END .END
43          .RETURN
```

Figure 1A. A procedure in MOL-32 which reads a line from the teletype and unpacks it into an area specified as a parameter.

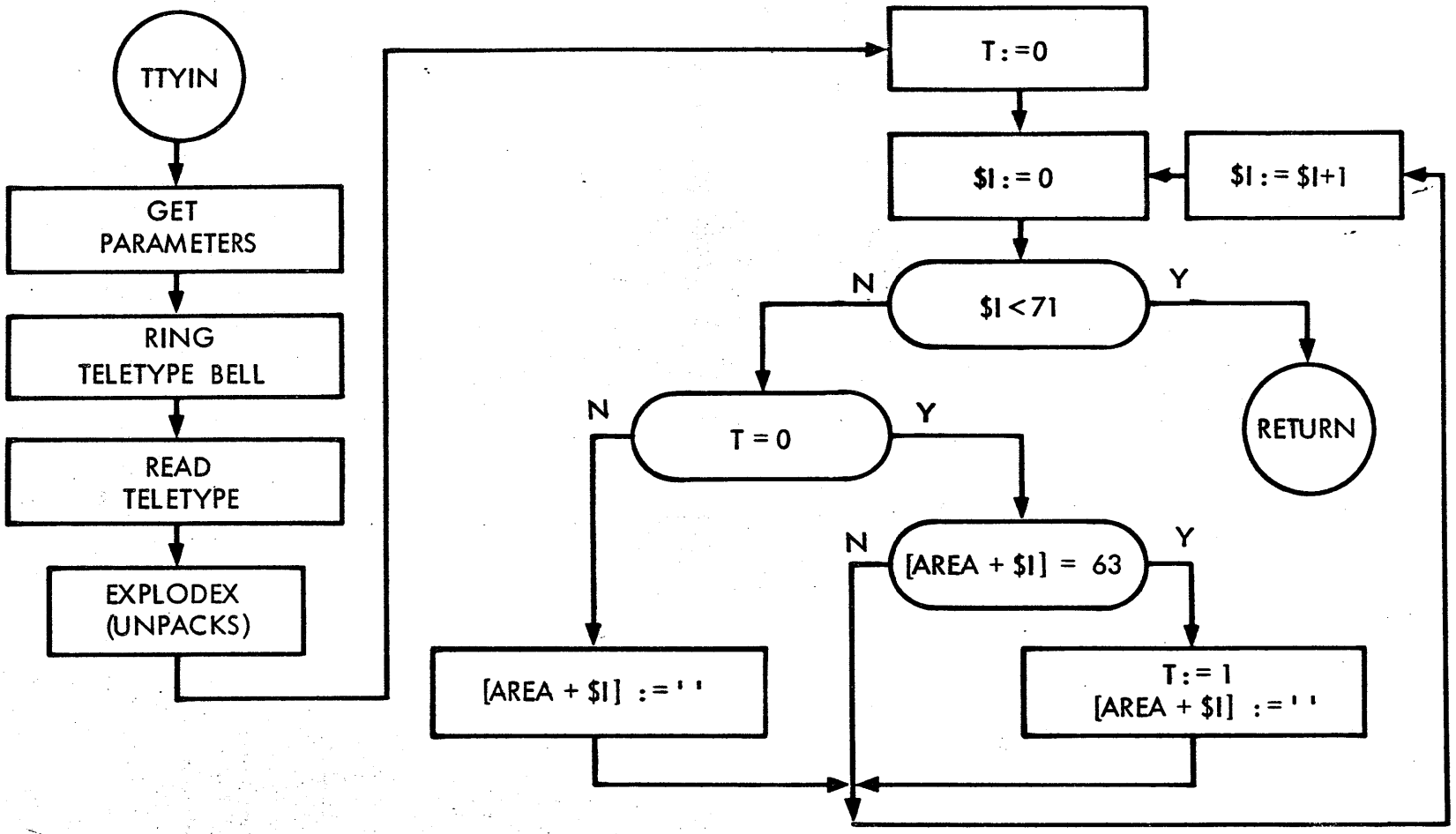

Figure 1B. A flow chart of the procedure to assist in explanation of the language. In actual practice, flow charts are unnecessary because the flow of control is graphically expressed by conventions of indentation.

2.2 THE ASSIGNMENT STATEMENT

The colon-equal (:=) is used in the assignment statement, just as in ALGOL. JOVIAL uses a single equal sign (=) for assignment, but we are using the equal sign for the relational operator which JOVIAL calls EQ. In line 34 of Figure 1A the word AREA is a local variable into which the parameter is stored. The square brackets around .EXIT indicate indirect addressing. The second assignment statement in line 34 adds one to the exit address so that control returns to the first word beyond the parameter.

In the current version of MOL-32 parentheses are not allowed within the expression on the right side of the assignment statement. All operations are performed from left to right without regard for precedence of operators. This simplification was made so that the compiler would not have to allocate temporary storage. A future version of MOL-32 will allow parentheses and follow the usual conventions for precedence of operators.

2.3 RELATIVE ADDRESSING AND INDIRECT ADDRESSING

An identifier followed by an expression enclosed in square brackets ([]) refers to a word whose address is obtained by adding the value of the expression to the address assigned to the identifier. For example, if AREA has been assigned the address 4050, then

```
AREA [25] := X;
```

means to store the contents of X into location 4075. Any legal arithmetic expression can occur between the square brackets.

An expression enclosed in square brackets but not preceded by an identifier indicates indirect addressing. For example,

```
X := [Y] ;
```

means to store the contents of the contents of location Y into location X. Any legal arithmetic expression can occur between the square brackets. Two levels of indirect addressing are shown in the example below:

```
X := [[Y]] ;
```

2.4 THE LOCAL DECLARATION

Now consider the local declaration which begins on line 31 of Figure 1A. The entries of the declaration are separated with commas (,); the declaration ends with a semicolon (;) on line 33. The first entry reserves a block of ten words, where BUF is the address of the first word. The second entry reserves one word to be called AREA, and the third reserves a word to be called T. Remember that we previously saw the identifier AREA in an assignment statement. The fourth and last entry says that ITLTY is the address of the first word of a block of

preassigned data. In a declaration, the colon-equal indicates preassigned data, whereas in the body of a procedure it indicates an assignment statement. The open parenthesis after the colon-equal indicates that more than one piece of preassigned data is to be given. The first piece of data is a long string of characters enclosed by single quotes. This is stored eight characters per word, and the last word is filled in with blanks on the right. The second piece of data is an address constant, #BUF, which is a word that contains the address of BUF in its address part. A programmer familiar with the Q-32 may recognize the preassigned data as a move call for the teletype.

Line 35 contains a call to the subroutine SPEAK. One argument is given to the subroutine, and this argument is a string consisting only of the single character bell. You cannot see this character in the listing, but you can hear it as the listing is being typed out. Usually one can see the characters inside a string, so it is especially unfortunate that this situation occurred in a procedure chosen for an example. The purpose of the procedure SPEAK is to print the given character string on the teletype or, as in this case, to ring the teletype bell.

2.5 DIRECT CODE

Direct code can be written between angle brackets (< and >). A programmer familiar with the Q-32 may recognize the code on line 35 of Figure 1A as a call to the system to read from the teletype.

2.6 INDEX REGISTERS

Index registers 1-6 are referred to directly as $I through $N. On lines 36 and 37 of Figure 1A, index registers 1 and 2, as well as the global variable AR1, are initialized before entry to the subroutine EXPLODEX. This routine unpacks the characters just read from the teletype and stores them in the block specified as parameter for TTYIN. Then the end of message character is removed, and the rest of the block filled with blanks.

2.7 THE FOR STATEMENT

The rest of the procedure can be understood if the for statement and the if statement are explained. The for statement is used to specify an indexed loop. All statements within the loop are indented. In Figure 1, the scope of the for statement continues to the end of the procedure, so that every statement is indented up to the reserved word .RETURN. Standard conventions for indenting should be followed by all programmers using MOL-32 so that the flow of control within a program can be recognized at a glance. The compiler ingores indentation, and considers the for statement to be terminated by the reserved word .END. The last .END on line 42 terminates the for statement. The other occurrence of .END on line 42 terminates an if statement, which is inside the for statement.

The general form of the for statement is given below:

```
                  ( $I )   ( .TO   )
.FOR              ( $J )   (       )
                  (  :  )  (       )                <expression>  :
                  (  :  )  (       )
                  ( $N )   ( .FROM )

        <sequence of statements>  .END
```

The index always ranges from 0 through the value of the expression; thus, the number of times the program goes through the loop is one greater than the value of the expression. When the value of the expression is 0, the loop is executed only once; when the value is negative, the instructions within the loop are not executed at all. Thus, index registers either start at 0 and are incremented by 1 up to the value of the expression, or start at the value of the expression and go down to 0. This is determined by the use of the words .TO and .FROM, respectively.

2.8 THE IF STATEMENT

The if statement has two basic forms, both of which are illustrated in Figure 1. The first form allows either of two sequences of statements to be executed, depending upon the value of a Boolean expression. Both sequences of statements are indented. The first sequence of statements begins with the reserved word .THEN; the second sequence begins with the reserved word .ELSE. The words .THEN and .ELSE are written at the same level of indentation, to indicate a parallel in the flow of control. The compiler ignores indentation, and considers the if statement to be terminated by the reserved word .END. The next to last .END on line 42 of Figure 1A terminates the if statement which begins on line 39. The .END on line 41 terminates another if statement which is inside this if statement.

The second form of the if statement allows for the optional execution of a sequence of statements. It is identical to the other form except that the else clause is omitted. The innermost if statement of the example, which begins on line 40, is of this form. The .END on line 41 terminates this statement. Notice that line 41 is indented five spaces from the beginning of line 40, and not five spaces from the occurrence of .IF on line 40.

The forms of the if statement may be summarized in meta-language as follows:

1. .IF <Boolean expression>
   .THEN <sequence of statements>
   .ELSE <sequence of statements> .END

2. .IF <Boolean expression>
   .THEN <sequence of statements> .END

2.9 BOOLEAN EXPRESSIONS

Now consider what is allowed in a Boolean expression. The relational operators are as follows:

Relation Operator	Meaning
==	equal, full word
=/=	unequal, full word
=	equal, numerical
/=	unequal, numerical
<	less than, numerical
>	greater than, numerical
<=	less than or equal, numerical
>=	greater than or equal, numerical

The reason for distinguishing between full word comparisons and numerical comparisions is that on the Q-32, +0 is different from -0.

There are two Boolean operators, .A. for and and .V. for or. Unlike arithmetic expressions, Boolean expressions may contain parentheses for grouping. The operator .A. takes precedence over .V.

Identifiers are not allowed to take on Boolean values; in other words, the operators .A. and .V. always connect relational expressions, never identifiers. Neither do we allow the Boolean operator not, but the effect of this operator can be obtained by using the appropriate relational operators.

2.10 THE LOOP STATEMENT

The procedure EXPLODEX, which is shown in Figure 2, contains an example of a loop statement, which was not illustrated in Figure 1. The general form of this statement is:

.LOOP WHILE <Boolean expression> :
    <sequence of statements> .END

```
176   EXPLODEX():
177         .LOOP WHILE $I < AR1:
178               [$I] := [$J].0;
179               [$I+1] := [$J].1;
180               [$I+2] := [$J].2;
181               [$I+3] := [$J].3;
182               [$I+4] := [$J].4;
183               [$I+5] := [$J].5;
184               [$I+6] := [$J].6;
185               [$I+7] := [$J].7;
186               $I := $I+8;  $J := $J+1; .END
187         .RETURN
```

Figure 2. A procedure in MOL-32 to unpack characters. Before entry to this procedure, the arguments are set up in index registers 1 and 2 ($I and $J) and in the global variable AR1.

The meaning of the "loop statement" is that the sequence of statements is to be executed as long as the Boolean expression is true. If the Boolean expression is false to begin with, the sequence of statements is never executed.

2.11 REFERENCING PART OF A WORD

In procedure EXPLODE, you can see the way of referring to parts of words. For example, [$J].0 on line 178 means the left-most byte of the word whose address is in index register 2. The computer word is divided into eight character-bytes, referred to as .0, .1, .2, ..., and .7, and into four other parts, referred to as prefix, decrement, tag, and address, referred to as .P, .D, .T, and .A, respectively. These may be used on either side of the colon-equal (:=) as illustrated in the following example:

```
A [B.D] .P:=C.5;
```

which stores the fifth character of the word whose address is C into the prefix of the word whose address is obtained by adding the address of A to the decrement of the word whose address is B.

2.12 DECLARATION BEGINNING A PROGRAM

Most of the features of the language that are used within procedures have been illustrated. The declaration that begins the source program has the same form as the local declaration within individual procedures, except that it begins with .DECLARE instead of .LOCAL.

2.13 THE GO STATEMENT

Statement labels and go statements are included in MOL-32, although they are seldom used. The statement label consists of an identifier followed by a colon (:); the go statement consists of the reserved word .GO followed by an expression (such as a statement label) which evaluates to the address of some instruction. A future version of MOL-32 will include a case statement which will handle situations presently requiring a computed go statement. In some cases a procedure can be made more efficient by using direct go statements, but this practice is not recommended because the flow of control will not be indicated by the indentation conventions.

2.14 THE ASSEMBLER

Figure 3 shows the SCAMP-like assembly language that is produced for the procedures shown in Figures 1 and 2. Most compilers generate statement labels which the assembler puts in a symbol table along with the identifiers in the source program. Instead of generating statement labels, the META SCAMP compiler turns out the labels *A and *B as well as the pseudo-instructions PSHA, POPA, PSHB, and POPB, which manage two stacks at assembly time. The pseudo-instructions PSHA and POPA serve as brackets so that *A is assigned the same address within their range. Similarly, the assignment of *B is done within the range of PSHB and POPB. We have drawn lines on the listing in Figure 3 in order to clarify this bracketing convention.

```
TTYIN
        SBR
        STP,567,7       EXIT
        PSHA
        BUC             *A
BUF
        BLK             10
AREA
        BLK   1
T
        BLK   1
ITLTY
        ('META  16TNSTAT00MOVE  66TELTYP  INPUT 4 COREIX1')R
        (BUF)R
*A
        POPA
        LDA,567,7       EXIT,I
        STA             AREA
        LDA,567,7       EXIT
        ADD,567,7,S     (1)R
        STA,567,7       EXIT
        BUC             SPEAK
        (1)
        ('')
        LDA (ITLTY)R
        BUC 202
        LDA             AREA
        STA,567,7,S     $X1
        LDA,567,7,S     (BUF)R
        STA,567,7,S     $X2
        LDA             AREA
        ADD,567,7,S     (72)R
        STA             AR1
        BUC             EXPLODEX
        LDA,567,7,S     (0)R
        STA             T
```

Figure 3. The procedures of Figures 1 and 2 are shown here in assembly language, Book's version of SCAMP.

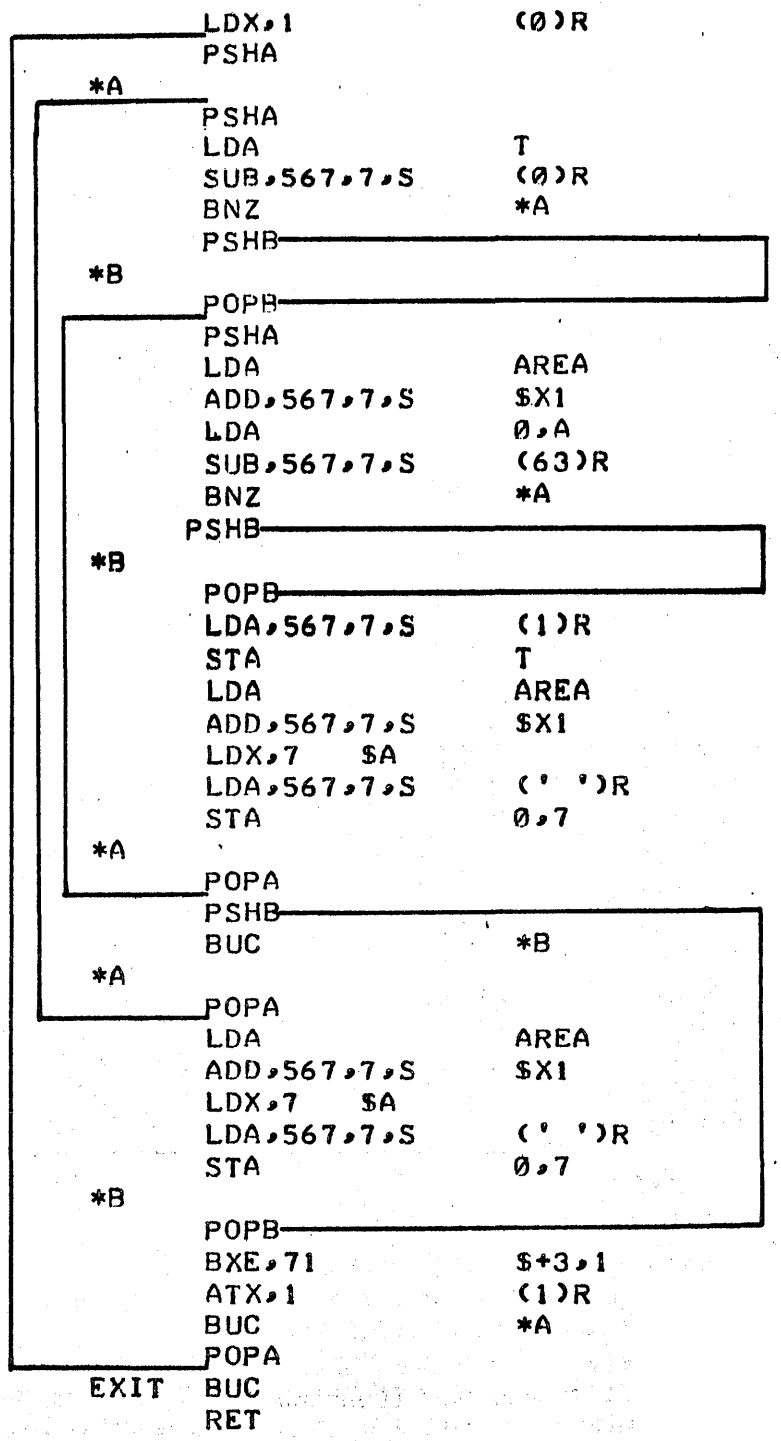

Figure 3. (Continued)

```
EXPLODEX
        SBR
        STP,567,7       EXIT
        PSHA
        PSHB
*B
        LDA,567,7,S     $X1
        SUB             AR1
        BOZ             *A
        BOP             *A
        PSHB
*B
        POPB
        LDA,7,0         ,2
        STA             ,1
        LDA,7,1         ,2
        STA             +1,1
        LDA,7,2         ,2
        STA             +2,1
        LDA,7,3         ,2
        STA             +3,1
        LDA,7,4         ,2
        STA             +4,1
        LDA,7,5         ,2
        STA             +5,1
        LDA,7,6         ,2
        STA             +6,1
        LDA,7,7         ,2
        STA             +7,1
        ATX,1           (+8)R
        ATX,2           (+1)R
        BUC             *B
*A
        POPA
        POPB
EXIT    BUC
        RET
        FIN
```

Figure 3. (Continued)

3. CONCLUSION

We have eliminated a major bottle-neck in our research by writing library routines of the META compiler in MOL-32 instead of in SCAMP assembly language. Not only are routines easier to write and check out, but easier to modify after they get cold.

In comparing the library written in MOL-32 to the library written in SCAMP, we noticed that the MOL-32 library took about 15% more space and executed about 15% slower than the SCAMP library. This seems reasonable considering the advantages gained.

The MOL-32 compiler was written in a version of META called SCAMP META. It took one month to program and only one and one-half days to check out.

The appendix contains (1) a specification of MOL-32 in SCAMP META and (2) the library of the META compiler, which serves as an example of a program written in the MOL-32.

APPENDIX A

Specification of MOL-32 in SCAMP META

```
00100-.SYNTAX  PROGRAM
00200-NUM .. DGT $ DGT ;
00210-NUM1 .. ,'+' NUM ;
00300-ID .. LET $(LET / DGT ) ;
00400-STRING = QUOTE .(,'(' .L ')' /,'(' *1 ')' /) ;
00500-DELETE = .Q(ISIT, CHAR, 0);
00600-ANY = .Q(ISIT, CHAR, 1) ;
00700-LET = .Q(ISIT, LETTER, 1) ;
00800-DGT = .Q(ISIT, DIGIT, 1) ;
00900-QUOTE .. +'''' $(-'''' ANY / +'''''') +'''' ;
01000-SCAMP1 .. ID $' ' (S1A / -'[') /
01100-      '[' $' ' ID $' '  ']' ,',I' / S1A ;
01200-INDX .. '$I' ,'$X1' / '$J' ,'$X2' / '$K' ,'$X3' /
01300-      '$L' ,'$X4' / '$M' ,'$X5' / '$N' ,'$X6' ;
01800-SCAMP2 = '.EXIT' ,'EXIT' ,',567,7' /
01900-      '[' '.EXIT' ']' ,'EXIT,I' ,',567,7' /
02000-      SCAMP1 PART / (CONST1/INDX) ,',567,7,S' ;
02100-S-NUM .. (+'+' / +'-') $' ' NUM ;
02200-CONSTANT .. S-NUM / NUM / QUOTE /
02300-      '#' ID $' ' (S-NUM / .EMPTY) ;
02400-CONST1 .. ,'(' CONSTANT ,')R' ;
02500-INDEX = '$I' ,',1' / '$J' ,',2' /
02600-      '$K' ,',3' / '$L' ,',4' /
02700-      '$M' ,',5' / '$N' ,',6' ;
02800-CONST2 .. S-NUM / '+#' ID $' '(S-NUM / .EMPTY) ;
02900-S1A = '[' $' ' (
03000-      '$I' $' ' (S-NUM / .EMPTY) ,',1' /
03100-      '$J' $' ' (S-NUM / .EMPTY) ,',2' /
03200-      '$K' $' ' (S-NUM / .EMPTY) ,',3' /
03300-      '$L' $' ' (S-NUM / .EMPTY) ,',4' /
03400-      '$M' $' ' (S-NUM / .EMPTY) ,',5' /
03500-      '$N' $' ' (S-NUM / .EMPTY) ,',6' /
03600-      S-NUM / NUM1) $' ' ']' ;
03700-PART = '.0' ,',7,0' / '.1' ,',7,1' /
03800-      '.2' ,',7,2' / '.3' ,',7,3' /
03900-     '.4' ,',7,4' / '.5' ,',7,5' /
04000-      '.6' ,',7,6' / '.7' ,',7,7' /
04100-      '.P' ,',7,0' / '.D' ,',567,3' /
04200-      -'.THEN' '.T' ,',7,4' / -'.A.' '.A' ,',567,7' /
04300-      .EMPTY ,' ' ;
```

APPENDIX A (Cont'd)

```
04400-EXP = (SCAMP2 (-'[') .(,'LDA' *1, *1/) /
04500-      (ID '['/'[' ,'0') EXP ']' PART .(,'LDA' *1, *1 ',A'/) ) $(
04600-      '+' SCAMP2 .(,'ADD' *1, *1/) /
04700-      '-' SCAMP2 .(,'SUB' *1, *1/) /
04800-      '*' SCAMP2 .(,'MUL' *1, *1/ ,'STB', '$A'/) /
04900-      '/' SCAMP2 .(,'LDB', '$A' / ,'SFC', '(47)R' /
05000-           ,'DVD' *1, *1/) /
05100-      '\' SCAMP2 .(,'LDB', '$A' / ,'SFC', '(47)R' /
05200-           ,'DVD' *1, *1 / ,'STB' ,'$A' /) /
05300-      '↑' CONST1 .(,'CYA', *1/) ) ;
05400-ASSIGNST = SCAMP2 ':=' EXP ';' .(,'STA' *1, *1/) /
05500-      (ID '[' EXP ']' / '[' EXP ']' ,'0') PART
05600-      ':=' .(,'LDX,7   $A' /)
05700-      EXP ';' .(,'STA' *1, *1 ',7' /) ;
05800-INDEXST = '$I' ':=' '$I' CONST1 ';' .(,'ATX,1', *1  /)/
05900-      '$J' ':=' '$J' CONST1 ';' .(,'ATX,2', *1  /)/
06000-      '$K' ':=' '$K' CONST1 ';' .(,'ATX,3', *1  /)/
06100-      '$L' ':=' '$L' CONST1 ';' .(,'ATX,4', *1  /) /
06200-      '$M' ':=' '$M' CONST1 ';' .(,'ATX,5', *1  /)/
06300-      '$N' ':=' '$N' CONST1 ';' .(,'ATX,6', *1  /);
06400-RELATION = EXP (
06500-      '==' CONSTANT .(,'BXE,' *1, '$+2,A' / ,'BUC', *A /) /
06600-      '==' SCAMP2 .(,'CML'*1, *1/ ,'BUC', *A/)/
06700-      '=/=' CONSTANT .(,'BXE,' *1, *A ',A' /) /
06800-      '=/=' SCAMP2 .(,'CML' *1, *1/ ,'BUC'  $+2' / ,'BUC', *A/) /
06900-      '=' SCAMP2 .(,'SUB' *1, *1/ ,'BNZ', *A/) /
07000-      '/=' SCAMP2 .(,'SUB' *1, *1 / ,'BOZ', *A/) /
07100-      '<' SCAMP2 .(, 'SUB' *1, *1/ ,'BOZ', *A / ,'BOP', *A/) /
07200-      '>' SCAMP2 .(,'SUB' *1, *1 / ,'BOZ', *A / ,'BNP' ,*A/ ) /
07300-      '<=' SCAMP2 .(,'SUB' *1, *1 / ,'BOP', *A/) /
07400-      '>=' SCAMP2 .(,'SUB' *1, *1 / ,'BOZ $+2'/ ,'BNP', *A/) );
07500-BASIC = RELATION / '(' BOOLEAN ')' ;
07600-FACTOR = BASIC $('.A.' BASIC) ;
07700-BOOLEAN = FACTOR .(+B)
07800-      $('.V.' .(,'BUC', *B/ *A/ -A+A) FACTOR)
07900-      .(*B / -B) ;
```

APPENDIX A (Cont'd)

```
08000-FORST = '.FOR' INDEX '.FROM' CONST1 ':'
08100-      .(,'LDX' +*2, *1/ +A*A/) $ ST '.END'
08200-      .(,'BPX,1' ,*A *1/-A)/
08300-      '.FOR' INDEX '.FROM' EXP ':'
08400-      .(+B, 'BXL,-0' ,*B ',A' / ,'SUB   (0)R'/ ,'LDX' +*1, '$A'/
08500-      +A*A /) $ ST '.END' .(,'BPX,1', *A *1/ -A*B/ -B)/
08600-      '.FOR' INDEX '.TO' CONSTANT  ':'
08700-      .(,'LDX' +*2, '(0)R' / +A*A/) $ ST '.END'
08800-      .(, 'BXE,' *1, '$+3' +*1 / , 'ATX' *1, '(1)R' /
08900-      ,'BUC', *A/ -A)/
09000-      '.FOR'INDEX '.TO' EXP ':'
09100-      .(+B, 'BXL,-0', *B ',A' / ,'SUB   (0)R' /
09200-      ,'STA,567,3', *B '-3' / ,'LDX' +*1, '(-0)R' / +A *A/)
09300-      $ ST '.END' .(,'BXE,0', *B +*1 / ,'ATX' *1, '(1)R' /
09400-      ,'BUC' , *A / -A *B / -B) ;
09500-LOOPST = '.LOOP' 'WHILE' .(+A +B *B/) BOOLEAN ':'
09600-      $ ST '.END' .(,'BUC', *B/ *A/ -A -B);
09700-IFST = '.IF' .(+A) BOOLEAN '.THEN' $ ST
09800-      ('.ELSE' .(+B,'BUC',*B / *A / -A)
09900-      $ST '.END' .(*B / -B) /
10000-      '.END' .(*A / -A) ) ;
10100-ERRORST = '.ERROR' 'UNLESS' .(+A) BOOLEAN ':'
10200-      .(+B, 'BUC', *B / *A, 'BUC SPEAK'/)
10300-      STRING ';' .(-A *B / -B) ;
10400-CALLST = ID '(' .(,'BUC', *1 /)
10500-      (ARG $(',' ARG) / .EMPTY) ')' ';' ;
10600-ARG = STRING / CONSTANT .(,'(' *1 ')'/) ;
10700-PUSHST = '.PUSH' INDEX ',' NUM ';'
10800-      .(,'ATX' +*2, '(-' *1 ')R' / ,'BMX,0   PUSHER' *1/) ;
10900-POPST = '.POP' INDEX ',' NUM ';'
11000-      .(,'ATX' +*2, '(' *1 ')R'/) .F(POP) ;
11100-GOST = '.GO' SCAMP2 .F(POP) .(,'BUC', *1/) ';' /
11200-      '.GO' EXP .(,'BUC   0,A'/) ';' ;
11300-LABEL = ID (-':=') ':' .(*1/) ;
11400-MACHINEST .. $(-';' ANY ) ';' ;
11500-MACHINE-CODE = '<' $(LABEL / -'>'
11600-      MACHINEST .(,*1/)) '>' ;
11700-ST = FORST / LOOPST / IFST / PUSHST / POPST /
11800-      GOST / MACHINE-CODE / ERRORST /
11900-      LABEL / INDEXST / CALLST / ASSIGNST ;
```

APPENDIX A (Cont'd)

```
12000-DATA = ID .(*1/) ('(' CONSTANT ')' .(,'BLK', *1/) /
12100-      ':=' (DATA1 /'(' DATA1 $(',' DATA1)')') /
12200-      .EMPTY .(,'BLK  1'/)) /
12300-      '#' ID '=' .(*1/) CONSTANT .(,'EQU ' *1/) ;
12400-DATA1 = CONST1 .(,*1/);
12500-DECLARATION = '.DECLARE' DATA $(',' DATA) ';' ;
12600-PROCEDURE = ID '():' .(*1/ , 'SBR' / ,'STP,567,7', 'EXIT'/)
12700-      ('.LOCAL' .(+A,'BUC',*A/) DATA $(',' DATA) ';'
12710-      .(*A /-A) / .EMPTY)
12800-      $(ST / -'.RETURN' ERROR
12900-      $(-'.RETURN' DELETE))
13000-      '.RETURN' ('[' EXP ']' / .EMPTY)
13010-      .('EXIT' ,'BUC' / ,'RET' /) ;
13100-PROGRAM = ('.PRIMITIVE' ID .(,'BUC', *1/,'BUC 195'/) /
13200-      .EMPTY) $ (PROCEDURE/DECLARATION) '.STOP' .(,'FIN'/) ;
13300-.END
```

APPENDIX B

Library of the META Compiler Written in MOL-32

```
00100-.DECLARE  #STARL = 1000,  STAR(#STARL),
00200-      #STACKL = 400,  STACK(#STACKL),
00300-      #INL1 = 73, #INL2 = 146, INBUF(#INL2),
00400-      #INPLACE = #INBUF + 73, INCOUNT, INX, MAXIMUM,
00500-      #OUTL1 = 73, #OUTL2 = 146, OUTBUF(#OUTL2),
00600-      #OUTPLACE = #OUTBUF + 73, OUTCOUNT, OUTX,
00700-      SIGNAL, TOKENDEPTH, TK(#INL1+1),
00800-      XPLBUF(#INL1), ERBUF(#INL1),
00900-      AR1, AR2, AR3, VL1, VL2, VL3, T1, T2, T3,
01000-      #ATOM = 1, P1INP(512), P1OUT(512),
01100-      BLANKS := '          ',
01200-      IUNIT, OUNIT, TTY := 'TTY      ',
01300-      FIELD, COUNT1, COUNT2,
01400-      SOURCE:=('META  18TNSTAT00MOVE  66SOURCE  INPUT 4 NUMWDS1',
01500-                512,'DISCIX1',0),
01600-      PRGRM:=('META  18TNSTAT00MOVE  66PRGRM   OUTPUT4 NUMWDS1'
01700-                ,512,'DISCIX1',0),
01800-      PERM  :=('META  17TNSTAT00INNAME66        INSERT66PRGRM  ',
01900-                'NUMWDS1', 0),
02000-      OLINE, ILINE, LETTER := 2, DIGIT := 12, CHAR := 1;
02100-SPEAK():  .LOCAL T, BUF(10), LFCR := 0327700000000000,
02200-      MESG :=('META  16TNSTAT00MOVE  66TELTYP  OUTPUT4 COREIX1',
02300-      #BUF);
02400-      T := [.EXIT]-1/8+1;
02500-      .FOR $I .FROM T-1:
02600-           BUF[$I] := [.EXIT+1+$I]; .END
02700-      BUF[T] := LFCR;
02800-      <LDA (MESG)R; BUC 202;>
02900-      .EXIT := .EXIT + T + 1;
03000-      .RETURN
03100-TTYIN():  .LOCAL BUF(10), AREA, T,
03200-      ITLTY :=('META  16TNSTAT00MOVE  66TELTYP  INPUT 4 COREIX1',
03300-      #BUF);
03400-      AREA := [.EXIT]; .EXIT := .EXIT+1;
03500-      SPEAK(''); <LDA (ITLTY)R; BUC 202;>
03600-      $I := AREA; $J := #BUF;
03700-      AR1 := AREA + 72; EXPLODEX(); T := 0;
03800-      .FOR $I .TO 71:
03900-           .IF T = 0
04000-                .THEN .IF [AREA+$I] = 63
04100-                     .THEN T := 1; [AREA+$I] := ' '; .END
04200-                .ELSE [AREA+$I] := ' '; .END .END
04300-      .RETURN
```

APPENDIX B (Cont'd)

```
04400-TTYOUT():  .LOCAL BUF(10), AREA, I,
04500-      OTLTY :=('META  16TNSTAT00MOVE  66TELTYP  OUTPUT4 COREIX1',
04600-      #BUF);
04700-      AREA := [.EXIT];  .EXIT := .EXIT+1;
04800-      $I := 71;
04900-      .LOOP WHILE [AREA + $I] = ' ' .A. $I >= 0:
05000-           $I := $I-1; .END
05100-      I := $I;
05200-      .IF I < 70
05300-           .THEN[I+AREA+1] := 032;
05400-           [I+AREA+2] := 077; .END
05500-      $I := AREA;  $J := #BUF; AR1 := AREA+72;  COMPRESSX();
05600-      .IF I < 70
05700-           .THEN[I+AREA+1] := ' ';
05800-           [I+AREA+2] := ' '; .END
05900-      <LDA (OTLTY)R; BUC 202;>
06000-      .RETURN
06100-READ():  .LOCAL AREA, FILE;
06200-      FILE := [.EXIT];  AREA := [.EXIT+1];  .EXIT := .EXIT+2;
06300-      .IF P1INP[ILINE-1].7 /= 26
06400-           .THEN .ERROR UNLESS P1INP[ILINE-1].7 /= 63:
06500-           'EOF READ';  ILINE := 0;
06600-           <LDA FILE;  BUC 202;>
06700-           .ERROR UNLESS SOURCE[1].7 =3: 'BAD READ';
06800-           SOURCE[8] := SOURCE[8]+1;
06900-           .LOOP WHILE P1INP[9].7 = 61:
07000-                <LDA FILE;  BUC 202;>
07100-                .ERROR UNLESS SOURCE[1].7 = 3:'BAD READ';
07200-                SOURCE[8] := SOURCE[8]+1; .END .END
07300-      $I := AREA;  $J := ILINE+#P1INP;
07400-      AR1 := AREA+#INL1-1;  EXPLODEX();
07500-      ILINE := ILINE+10;
07600-      .RETURN
```

APPENDIX B (Cont'd)

```
07700-WRITE():  .LOCAL AREA, FILE, T,
07800-      MORE :=('META  15TNSTAT00MODIFY66PRGRM   NUMWDS1',4096);
07900-      FILE := [.EXIT]; AREA := [.EXIT+1]; .EXIT := .EXIT+2;
08000-      .IF OLINE = 510
08100-           .THEN .IF PRGRM[8]+1\8 = 0
08200-                .THEN MORE[3] := [FILE+3];
08300-                MORE[5] := MORE[5]+4096;
08400-                <LDA (MORE)R; BUC 202;>
08500-                .ERROR UNLESS MORE[1].7 = 3:
08600-                'NO MORE DISC SPACE'; .END
08700-           T1 := COUNT2 * 100;
08800-           P1OUT[510] := COUNT1*100+24+T1;
08900-           P1OUT[511].3 := COUNT2-COUNT1+1;
09000-           <LDA FILE; BUC 202;> COUNT1 := COUNT2 + 1;
09100-           .ERROR UNLESS PRGRM[1].7 = 3: 'BAD WRITE';
09200-           OLINE := 0; PRGRM[8] := PRGRM[8]+1; .END
09300-      $I := AREA; $J := OLINE+#P1OUT;
09400-      AR1 := AREA+#OUTL1-1; COMPRESSX();
09500-      COUNT2 := COUNT2 + 1;
09600-      P1OUT[OLINE+9].4 := COUNT2\10; T := COUNT2/10;
09700-      P1OUT[OLINE+9].3 := T\10; T := T/10;
09800-      P1OUT[OLINE+9].2 := T\10; T := T/10;
09900-      P1OUT[OLINE+9].1 := T\10;
10000-      OLINE := OLINE + 10;
10100-      .RETURN
10200-IONAMES(): .LOCAL V(6);
10300-      .FOR $I .FROM 5:
10400-           V[$I] := ' '; .END
10500-      .ERROR UNLESS STAR[$M].A <= 6: 'LONG NAME';
10600-      .FOR $I .FROM STAR[$M].A-1:
10700-           V[$I] := STAR[$M+$I+1]; .END
10800-      POP(); VL1 := BLANKS;
10900-      VL1.0 := V[0]; VL1.1 := V[1]; VL1.2 := V[2];
11000-      VL1.3 := V[3]; VL1.4 := V[4]; VL1.5 := V[5];
11100-      .RETURN
```

APPENDIX B (Cont'd)

```
11200-INITIALIZE():  .LOCAL
11300-      ODISCF:=('META  19TNSTAT00FILE  66PRGRM    UNIT  0=',
11400-               'FORM  5CINLOC 1',#P1OUT,'NUMWDS1',4096),
11500-      IDISCF:=('META  17TNSTAT00REFILE66         RENAME66',
11600-               'SOURCE  INLOC 1',#P1INP);
11700-      .FOR $I .FROM 71:
11800-           ERBUF[$I] := ' '; .END
11900-      TOKENDEPTH := 0;  TK := 0;
12000-      $M := #STARL;  $N := #STACKL-1;  <LDI,070000 (6)R;>
12100-      INCOUNT := 0;  MAXIMUM := 0;
12200-      OUTCOUNT := 0;  OUTX := 1; SIGNAL := .EXIT;
12300-      $I := 0;
12400-      .FOR $I .FROM #OUTL2-1:
12500-           OUTBUF[$I] := ' '; .END
12600-      FIELD := 0;
12700-      INBUF[#INL1-1] := 077;  INBUF[#INL2-1] := 077;
12800-      IUNIT := TTY;  INBUF[#INL2-2] := ' ';
12900-      INX := #INL2-2;  SPEAK('INPUT OUTPUT');
13000-      ID();  ID();
13100-      IONAMES();  OUNIT := VL1;
13200-      .IF OUNIT =/= TTY
13300-           .THEN PERM[3] := OUNIT;  OLINE := 0;
13400-           <LDA (ODISCF)R;  BUC 202;>
13500-           PRGRM[8] := 0;  COUNT1 := 1;  COUNT2 := 0;
13600-           P1OUT[511] := 51;
13700-           .LOOP WHILE $I < 510:
13800-                P1OUT[$I+9] := 032;
13900-                $I := $I + 10;  .END .END
14000-           P1OUT[509] := 076;
14100-      IONAMES();  IUNIT := VL1;
14200-      .IF IUNIT =/= TTY
14300-           .THEN IDISCF[3] := IUNIT;
14400-           <LDA (IDISCF)R; BUC 202;>
14500-           SOURCE[8] := 0;
14600-           ILINE := 10;  P1INP[9] := 076; .END
14700-      .RETURN
```

APPENDIX B (Cont'd)

```
14800-COMPLETE():
14900-      .IF OUTX > 0
15000-           .THEN CARD();  .END
15100-      .IF OUTX > #OUTL1+1
15200-           .THEN CARD();  .END
15300-      .IF OUNIT =/= TTY
15400-           .THEN  P1OUT[OLINE-1].7 := 63;
15500-           T1 := COUNT2*100;
15600-           P1OUT[510] := COUNT1*100+24+T1;
15700-           P1OUT[511].3 := COUNT2-COUNT1+1;
15800-           <LDA (PRGRM)R;  BUC 202;>
15900-           .ERROR UNLESS PRGRM[1].7 = 3: 'BAD WRITE';
16000-           PERM[7] := PRGRM[8] + 1 * 512;
16100-           <LDA (PERM)R;  BUC 202;>
16200-           .ERROR UNLESS PRGRM[1].7 = 3:
16300-           'REDUNDANT OUTPUT NAME';  .END
16400-      .ERROR UNLESS SIGNAL /= 0: 'ILLEGAL PROGRAM';
16500-      .RETURN
16600-NXTCHR():
16700-      INX := INX+1;
16800-      .IF INX >= #INL2
16900-           .THEN .FOR $I .FROM #INL1-2:
17000-                INBUF[$I] := INBUF[$I+#INL1]; .END
17100-           INX := #INL1;  INCOUNT := INCOUNT+ #INL1;
17200-           .IF IUNIT =/= TTY
17300-                .THEN READ(#SOURCE, #INPLACE);
17400-                .ELSE TTYIN(#INPLACE);  .END .END
17500-      .RETURN
17600-EXPLODEX():
17700-      .LOOP WHILE $I < AR1:
17800-           [$I] := [$J].0;
17900-           [$I+1] := [$J].1;
18000-           [$I+2] := [$J].2;
18100-           [$I+3] := [$J].3;
18200-           [$I+4] := [$J].4;
18300-           [$I+5] := [$J].5;
18400-           [$I+6] := [$J].6;
18500-           [$I+7] := [$J].7;
18600-           $I := $I+8;  $J := $J+1; .END
18700-      .RETURN
```

APPENDIX B (Cont'd)

```
18800-COMPRESSX():
18900-      .LOOP WHILE $I < AR1:
19000-           [$J].0 := [$I];
19100-           [$J].1 := [$I+1];
19200-           [$J].2 := [$I+2];
19300-           [$J].3 := [$I+3];
19400-           [$J].4 := [$I+4];
19500-           [$J].5 := [$I+5];
19600-           [$J].6 := [$I+6];
19700-           [$J].7 := [$I+7];
19800-           $I := $I+8;  $J := $J+1;  .END
19900-      .RETURN
20000-ERROR():  .LOCAL A;
20100-      TTYOUT(#INPLACE);
20200-      .IF INCOUNT + INX > MAXIMUM
20300-           .THEN MAXIMUM := INCOUNT + INX; .END
20400-      A := MAXIMUM - INCOUNT - #INL1;
20500-      ERBUF[A] := '↑';
20600-      TTYOUT(#ERBUF);  ERBUF[A] := ' ';
20700-      .RETURN
20800-PUSHER():
20900-      .ERROR UNLESS 1 = 0 : 'OUT OF PUSHDOWN LIST';
21000-      .RETURN
21100-ISIT():  .LOCAL X, CONVERT :=(
21200-       9,  9,  9,  9,  9,  9,  9,  9,
21300-       5,  5, 33,  1,  1, 33, 33, 33,
21400-       1,  3,  3,  3,  3,  3,  3,  3,
21500-       3,  3, 33,  1,  1, 33, 33, 33,
21600-       1,  3,  3,  3,  3,  3,  3,  3,
21700-       3,  3, 33,  1,  1, 33, 33, 33,
21800-      17,  1,  3,  3,  3,  3,  3,  3,
21900-       3,  3, 33,  1,  1, 33, 33, 33 );
22000-      AR1 := [[.EXIT]];  AR2 := [.EXIT+1];  .EXIT := .EXIT +2;
22100-      X := CONVERT[INBUF[INX]];  <ANA AR1; STF SIGNAL;>
22200-      .IF SIGNAL /= 0
2300-           .THEN .IF AR2 /= 0
22400-                 .THEN TK := TK+1;
22500-                 .ERROR UNLESS TK < #INL1: 'LONG TOKEN';
22600-                TK[TK] := INBUF[INX];
22700-                 .IF TOKENDEPTH = 0
22800-                      .THEN MAKETOKEN();  .END .END
22900-           NXTCHR();  .END
23000-      .RETURN
```

APPENDIX B (Cont'd)

```
23100-EXPLODE():  .LOCAL T;
23200-      VL1 := [AR1]; VL2 := VL1 - 1 / 8 + 2;
23300-      $J := 0;
23400-     .FOR $I .TO VL2 - 2 :
23500-           T := [AR1 + $I + 1];
23600-           XPLBUF[$J] := T.0;
23700-           XPLBUF[$J+1] := T.1;
23800-           XPLBUF[$J+2] := T.2;
23900-           XPLBUF[$J+3] := T.3;
24000-           XPLBUF[$J+4] := T.4;
24100-           XPLBUF[$J+5] := T.5;
24200-           XPLBUF[$J+6] := T.6;
24300-           XPLBUF[$J+7] := T.7;
24400-           $J := $J + 8;  .END
24500-      .RETURN
24600-SKIPBLANKS():
24700-      .LOOP WHILE INBUF[INX] = ' ' .V. INBUF[INX] = 63:
24800-           NXTCHR();  .END
24900-      .RETURN
25000-INSERT():
25100-      AR1 := .EXIT;
25200-      EXPLODE();  .EXIT := .EXIT + VL2;  INSERTX();
25300-      .RETURN
25400-INSERTX():
25500-      .ERROR UNLESS TK+VL1 < #INL1: 'LONG TOKEN';
25600-      .FOR $I .FROM VL1-1:
25700-           TK[$I + TK + 1] := XPLBUF[$I]; .END
25800-      TK := TK + VL1;
25900-      .IF TOKENDEPTH = 0
26000-           .THEN MAKETOKEN();  .END
26100-      .RETURN
26200-MAKETOKEN():
26300-      $M := $M - 1 - TK;  .ERROR UNLESS $M > 0:'FULL STACK' ;
26400-      STAR[$M].D := TK + 1;
26500-      STAR[$M].A := TK;
26600-      STAR[$M].P := #ATOM;
26700-      .FOR $I .FROM TK - 1:
26800-           STAR[$M + $I + 1] := TK[$I + 1];  .END
26900-      TK := 0;
27000-      .RETURN
27100-COMP():
27200-      AR1 := .EXIT;  AR2 := 1;  COMPARE();  .EXIT := VL2;
27300-      .RETURN
27400-COMPS():
27500-      AR1 := .EXIT;  AR2 := 2;  COMPARE();  .EXIT := VL2;
27600-      .RETURN
27700-NCOMP():
27800-      AR1 := .EXIT;  AR2 := 3;  COMPARE();  .EXIT := VL2;
27900-      .RETURN
```

APPENDIX B (Cont'd)

```
28000-COMPARE():  •LOCAL TYPE, L;
28100-      TYPE := AR2;  L := [AR1];
28200-      EXPLODE(); VL2 := VL2 + AR1;
28300-      •IF TOKENDEPTH = 0
28400-           •THEN SKIPBLANKS();  •END
28500-      SIGNAL := •EXIT;
28600-      •FOR $I •FROM L-1:
28700-           •IF INBUF[INX + $I] /= XPLBUF[$I]
28800-                •THEN SIGNAL := 0;  •END •END
28900-      •IF TYPE = 3
29000-           •THEN •IF SIGNAL /= 0
29100-                •THEN SIGNAL := 0;
29200-                •ELSE SIGNAL := •EXIT;  •END
29300-           •ELSE •IF TYPE = 2 •A• SIGNAL /= 0
29400-                •THEN INSERTX();  •END
29500-           •IF SIGNAL /= 0
29600-                •THEN INX := INX + L;  •END •END
29700-      •RETURN
29800-MARK():
29900-      •IF TOKENDEPTH = 0
30000-           •THEN SKIPBLANKS();  •END
30100-      TOKENDEPTH := TOKENDEPTH + 1;
30200-      •RETURN
30300-TOKEN():
30400-      TOKENDEPTH := TOKENDEPTH - 1 ;
30500-      •IF TOKENDEPTH = 0 •A• SIGNAL /= 0
30600-           •THEN MAKETOKEN();  •END
30700-      •RETURN
30800-SAVE():
30900-      •PUSH $N, 3;
31000-      STACK[$N+1]•D := INCOUNT + INX;
31100-      STACK[$N+1]•A := OUTCOUNT + OUTX;
31200-      STACK[$N+2]•A := $M;
31300-      STACK[$N+2]•D := TK;
31400-      STACK[$N+3] := TOKENDEPTH;
31500-      •RETURN
31600-BACKUP():
31700-      •IF INCOUNT + INX > MAXIMUM
31800-           •THEN MAXIMUM := INCOUNT + INX;  •END
31900-      INX := STACK[$N+1]•D - INCOUNT;
32000-      OUTX := STACK[$N+1]•A - OUTCOUNT;
32100-      $M := STACK[$N+2]•A;
32200-      TK := STACK[$N+2]•D; TOKENDEPTH := STACK[$N+3];
32300-      •ERROR UNLESS 0 < INX •A• 0 < OUTX:  'EXCESSIVE BACKUP';
32400-      •RETURN
32500-RSTOR():
32600-      •POP $N,3;
32700-      •RETURN
```

APPENDIX B (Cont'd)

```
32800-CARD():
32900-      FIELD := 0;
33000-      .IF #OUTL1 < OUTX
33100-           .THEN .IF OUNIT =/= TTY
33200-                .THEN WRITE(#PRGRM, #OUTBUF);
33300-                .ELSE TTYOUT(#OUTBUF); .END
33400-           OUTCOUNT := OUTCOUNT + #OUTL1;
33500-           .FOR $I .FROM #OUTL1 - 1:
33600-                OUTBUF[$I] := OUTBUF[$I + #OUTL1];
33700-                OUTBUF[$I + #OUTL1] := ' ' ; .END .END
33800-      OUTX := #OUTL1 + 1;
33900-      .RETURN
34000-TAB(): .LOCAL   TABCOL := (0, 7, 23, 47), #N = 4;
34100-      FIELD := FIELD + 1;
34200-      .ERROR UNLESS FIELD < #N : 'TAB ERROR' ;
34300-      T1 := OUTX \ #OUTL1 ;
34400-      OUTX := TABCOL[FIELD] + OUTX - T1 ;
34500-      .RETURN
34600-OUTSTG():
34700-      AR1 := .EXIT;  EXPLODE();  .EXIT := .EXIT + VL2;
34800-      T1 := OUTX + VL1 - 1 / #OUTL1;
34900-      .IF OUTX - 1 / #OUTL1 /= T1
35000-           .THEN CARD();  .END
35100-      .FOR $I .FROM VL1 - 1:
35200-           OUTBUF[$I + OUTX] := XPLBUF[$I]; .END
35300-      OUTX := OUTX + VL1;
35400-      .RETURN
35500-OUTTOKEN(): .LOCAL L;
35600-      L := [AR1].A;
35700-      .ERROR UNLESS [AR1].P = #ATOM: 'NOT A TOKEN';
35800-      T1 := OUTX + L - 1 / #OUTL1;
35900-      .IF OUTX - 1 / #OUTL1 /= T1
36000-           .THEN CARD();  .END
36100-      .FOR $I .FROM L - 1:
36200-           OUTBUF[$I + OUTX] := [AR1 + 1 + $I]; .END
36300-      OUTX := OUTX + L;
36400-      .RETURN
```

APPENDIX B (Cont'd)

```
36500-LENGTH():  .LOCAL L;
36600-      L := STAR[$M].A-2;
36700-      .FOR $I .FROM L-1:
36800-           .IF STAR[$M+2+$I] = ''''
36900-                .THEN L := L-1;  $I := $I-1; .END .END
37000-      .IF L > 9
37100-           .THEN T1 := OUTX+1 / #OUTL1;
37200-           .IF OUTX-1 / #OUTL1 /= T1
37300-                .THEN CARD();  .END
37400-           OUTBUF[OUTX] := L / 10;
37500-           OUTBUF[OUTX+1] := L \ 10;
37600-           OUTX := OUTX + 2;
37700-           .ELSE T1 := OUTX / #OUTL1;
37800-           .IF OUTX-1 / #OUTL1 /= T1
37900-                .THEN CARD();  .END
38000-           OUTBUF[OUTX] := L \ 10;
38100-           OUTX := OUTX + 1; .END
38200-      .RETURN
38300-STAR1P():
38400-      AR1 := #STAR + $M;  OUTTOKEN();
38500-      .RETURN
38600-STAR2P():
38700-      AR1 := #STAR + $M + STAR[$M].D;  OUTTOKEN();
38800-      .RETURN
38900-STAR1():
39000-      STAR1P();  POP();
39100-      .RETURN
39200-POP():
39300-      $M := $M + STAR[$M].D;
39400-      .RETURN
39500-.STOP
```

Distribution List

Name	Room
S. Aranda	2214
J. Barnett	2025
P. Bartram	2336
F. Blair (IBM)	2306
M. Bleier	2324
R. Bleier	3673
E. Book	2332
D. Boreta	1218
R. Bosak	2013
S. Bowman	2322
H. Bratman	2340
R. Brewer	2320
E. Clark	2338
V. Cohen	2326
W. Cozier	2224
P. Cramer	1141B
R. Dinsmore	2220
G. Dobbs	2111B
L. Durham	2424
J. Farell	9731
D. Firth	2310
E. Foote	2415
D. Haggerty	9726
L. Hawkinson (III)	9717
J. Hopkins	2423
C. Irvine	1139
E. Jacobs	2344
S. Kameny	2009B
C. Kellogg	9514
H. Manelowitz	9915
B. Saunders (III)	9717
M. Schaefer	2424
V. Schorre (30)	2330
J. Schwartz	2123
C. Shaw	2428
H. Silberman	9518
R. Simmons	9439
T. Steel	9024
E. Stefferud	9734
P. Styar	9717
A. Vorhaus	2213
C. Weissman (10)	2214
K. Yarnold	9222

External Distribution List

Barry B. Smith
1608 Manzanita Lane
Manhatten Beach, California 90267

George Kreglow
1074 N/7312 S. Jefferson St.
Anaheim, California 92805

Edward Manderfield
North American Aviation
Space & Information Systems Division
D196/322 Bldg. 4
Downey, California

Lee O. Schmidt
Beckman Instruments Systems Division
2400 Harbor Blvd.
Fullerton, California 92634

Andy Chapman
1850 Colby Avenue
Los Angeles, California 90025

Fred Schneider
UCLA Computing Facility
3532 Engineering 3
405 Hilgard Avenue
Los Angeles, California 90024

Dr. P. Abrahams
Information International, Inc.
Room 400, 119 W. 23rd St.
New York, New York 10011

M. Levin
Information International, Inc.
545 Technology Square
Cambridge, Massachusetts 02139

Jorge Mezei 12-018
T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, New York 10598