Burroughs

B 1700 SYSTEMS

SYSTEM SOFTWARE DEVELOPMENT LANGUAGE (SDL)

REFERENCE MANUAL (BNF VERSION)

$4.00

Burroughs

B 1700 SYSTEMS
SYSTEM SOFTWARE DEVELOPMENT LANGUAGE
(SDL)

REFERENCE MANUAL
(BNF VERSION)

Burroughs Corporation
Detroit, Michigan 48232

$4.00

Printed in U. S. America 12-74 1081346

AA500233

Correspondence regarding this document should be forwarded using the Remarks Form at the back of the manual, or may be addressed directly to Systems Documentation, Technical Information Organization, TIC-Central, Burroughs Corporation, Burroughs Place, Detroit, Michigan 48232.

TABLE OF CONTENTS

GENERAL ORIENTATION: THE METALANGUAGE

A LANGUAGE USED TO TALK ABOUT A LANGUAGE IS A METALANGUAGE. THE NATURAL LANGUAGES ARE, IN FACT, METALANGUAGES; FOR EXAMPLE, THE METALANGUAGE ENGLISH IS USED TO TALK ABOUT THE RELATIONSHIP E=IR, I.E., VOLTAGE EQUALS THE PRODUCT OF CURRENT AND RESISTANCE. BACKUS NAUR FORM (BNF), A METALANGUAGE POPULARIZED BY ITS USE TO DESCRIBE THE SYNTAX OF ALGOL 60 IS USED TO DESCRIBE THE SYNTAX OF SDL. TO AVOID THE CONFUSION BETWEEN THE SYMBOLS OF THE METALANGUAGE AND THOSE OF THE LANGUAGE BEING DESCRIBED, BNF USES ONLY 4 METALINGUISTIC SYMBOLS. LITERAL OCCURRENCES OF SYMBOLS, WITH NO BRACKETING CHARACTERS, REPRESENT THEMSELVES AS TERMINAL SYMBOLS OF THE LANGUAGE.

A GRAMMAR FOR SDL IS WRITTEN AS A SET OF BNF STATEMENTS, EACH OF WHICH HAS A LEFT PART, FOLLOWED BY THE METASYMBOL "::=" FOLLOWED BY A LIST OF RIGHT PARTS. THE LEFT PART IS A PHRASE NAME, AND THE RIGHT PARTS, SEPARATED BY THE METASYMBOL "/" ARE STRINGS CONTAINING TERMINAL SYMBOLS AND/OR PHRASE NAMES.

METASYMBOLS OF BNF

METASYMBOL	ENGLISH EQUIVALENT	USE
::=	IS DEFINED AS	SEPARATES A PHRASE NAME FROM ITS DEFINITION
/	OR	SEPARATES ALTERNATE DEFINITIONS OF A PHRASE
<IDENTIFIER>	"IDENTIFIER"	THE BRACKETING CHARACTERS INDICATE THAT THE INTERVENING CHARACTERS ARE TO BE TREATED AS A UNIT, I.E., AS A PHRASE NAME

NOTE: BNF ACTUALLY USES A VERTICAL BAR, RATHER THAN THE SLASH AS INDICATED ABOVE AS A SEPARATOR. HOWEVER, THE LINE PRINTER WHICH PRODUCES THIS DOCUMENT HAS NO VERTICAL BAR IN ITS CHARACTER SET. THEREFORE, A SLASH HAS BEEN USED THROUGHOUT TO SEPARATE ALTERNATE SYNTACTICAL DEFINITIONS. WHEN THE SLASH IS ACTUALLY PART OF THE SDL SYNTAX, IT WILL BE WRITTEN AS <SLASH>.

EACH BNF STATEMENT IS A REWRITING RULE, SUCH THAT WE MAY SUBSTITUTE ANY RIGHT PART FOR ANY OCCURRENCE OF ITS ASSOCIATED LEFT PART; AND WE HAVE A CHOICE OF RIGHT PARTS WHICH WE MAY SUBSTITUTE. THE FOLLOWING EXAMPLE SPECIFIES THE USE OF THESE RULES TO DETERMINE THOSE STRINGS WHICH ARE GRAMMATICALLY CORRECT IDENTIFIERS IN SDL.

```
<LETTER> ::=        A / B / C / D / E / F / G / H / I / J / K / L / M /
                    N / O / P / Q / R / S / T / U / V / W / X / Y / Z

<DIGIT> ::=         0 / 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9

<DOT> ::=

<IDENTIFIER> ::=    <LETTER>
                  / <IDENTIFIER> <LETTER>
                  / <IDENTIFIER> <DIGIT>
                  / <IDENTIFIER> <DOT>
```

XYZ12.B4 IS A PROPER SDL <IDENTIFIER> SINCE IT CAN BE GENERATED AS A TERMINATING SET OF SYMBOLS BY USING THE BNF RULES.

<IDENTIFIER>
<IDENTIFIER> <DIGIT>
<IDENTIFIER> <LETTER> 4
<IDENTIFIER> <DOT> B4
<IDENTIFIER> <DIGIT> .B4
<IDENTIFIER> <DIGIT> 2.B4
<IDENTIFIER> <LETTER> 12.B4
<IDENTIFIER> <LETTER> Z12.B4
<LETTER> YZ12.B4
XYZ12.B4

NOTICE THAT THE BNF RULES DO NOT, IN ANY WAY, LIMIT THE NUMBER OF LETTERS, DIGITS, AND DOTS WHICH COMPRISE THE <IDENTIFIER>. IN SUCH CASES, FURTHER SEMANTIC RULES WILL BE SPECIFIED; E.G., AN SDL <IDENTIFIER> IS LIMITED TO A MAXIMUM OF 63 CHARACTERS.

BASIC COMPONENTS OF THE SDL LANGUAGE

IN ORDER TO UNDERSTAND SDL GRAMMAR, THE USER SHOULD BE FAMILIAR WITH THE MOST BASIC ELEMENTS OF THE SDL LANGUAGE BELOW.

<DIGIT> ::= 0 / 1 / 2 / ... / 8 / 9

<LETTER> ::= A / B / C / D / ... / X / Y / Z

<SPECIAL CHARACTER> ::= <AMPERSAND> / . / < / ; / , / ← / ≤ / $ / : / > / ≥ / ≠ / = / + / (/) / * / - / <SLASH> / [/] / <BLANK>

<AMPERSAND> ::= &

<SLASH> ::= /

<BLANK> ::=

NOTE: <BLANK> IS THE OCCURENCE OF ONE NON-VISIBLE CHARACTER " ".

<IDENTIFIER> ::= <LETTER> / <IDENTIFIER> <LETTER> / <IDENTIFIER> <DIGIT> / <IDENTIFIER> <DOT>

RESTRICTIONS:

1. AN IDENTIFIER MUST BEGIN WITH A LETTER.
2. AN IDENTIFIER MAY NOT CONTAIN BLANKS.
3. AN IDENTIFIER MAY CONTAIN A MAXIMUM OF 63 CHARACTERS.
4. EXCEPT FOR SEGMENT AND GROUP IDENTIFIERS, RESERVED WORDS MAY NOT BE USED AS IDENTIFIERS.

5. "SPECIAL" WORDS MAY BE USED FOR SEGMENT AND DO-GROUP <IDENTIFIER>S WITHOUT LOSING THEIR SPECIAL SIGNIFICANCE IN SDL.

6. IN ALL OTHER CASES, "SPECIAL" WORDS MAY BE USED AS IDENTIFIERS, HOWEVER, THEY LOSE THEIR SPECIAL SIGNIFICANCE THROUGHOUT THE ENTIRE PROGRAM WHEN DECLARED AT LEXIC LEVEL 0. WHEN DECLARED AT ANY GREATER LEXIC LEVEL, THEY ONLY LOSE THEIR SPECIAL MEANING WITHIN THE PROCEDURE IN WHICH THEY ARE DECLARED.

   (ALSO SEE "BASIC STRUCTURE OF THE SDL PROGRAM" AND "APPENDIX III")

<DOT> ::=

<COMMENT STRING> ::= <SLASH>* <COMMENT TEXT> *<SLASH>

RESTRICTIONS:

1. THE PAIR /* PRECEDING THE <COMMENT TEXT> MUST APPEAR AS ADJACENT SYMBOLS. SIMILARILY, THE PAIR */ FOLLOWING THE <COMMENT TEXT> MUST ALSO APPEAR AS ADJACENT SYMBOLS.

```
<COMMENT TEXT> ::=          <EMPTY>
                          / <COMMENT TEXT CHARACTER>
                          / <COMMENT TEXT CHARACTER>
                            <COMMENT TEXT>
```

<EMPTY> ::=

NOTE: <EMPTY> IS THE NULL SET OR THE OCCURENCE OF NOTHING.

```
<COMMENT TEXT
CHARACTER> ::=              <DIGIT>
                          / <LETTER>
                          / <SPECIAL CHARACTER>
                          / " / @ / # / %
```

<CARD TERMINATOR> ::= %

RESTRICTIONS:

1. A % IS TREATED AS ANY OTHER STRING CHARACTER IF IT IS CONTAINED WITHIN A <CHARACTER STRING> OR IN <COMMENT TEXT>. HOWEVER, IN ALL OTHER CASES, A % WILL CAUSE THE SCANNING OF THE CURRENT SOURCE IMAGE TO TERMINATE, AND CAUSE SCANNING TO CONTINUE IN THE NEXT SOURCE IMAGE.

```
<NUMBER> ::=                    <DIGIT> / <NUMBER> <DIGIT>

                        NOTE: RANGE OF SIGNED NUMBERS -(2 EXP 23) TO
                              (2  EXP  23)-1.  RANGE  OF  UNSIGNED
                              NUMBERS 0 TO (2 EXP 24)-1.

<BINARY DIGIT> ::=              0 / 1 / <COMMENT STRING>

<BINARY DIGITS> ::=               <BINARY DIGIT>
                                / <BINARY DIGITS> <BINARY DIGIT>

<QUARTAL DIGIT> ::=             <BINARY DIGIT> / 2 / 3

<QUARTAL DIGITS> ::=              <QUARTAL DIGIT>
                                / <QUARTAL DIGIT> <QUARTAL DIGITS>

<OCTAL DIGIT> ::=               <QUARTAL DIGIT> / 4 / 5 / 6 / 7

<OCTAL DIGITS> ::=                <OCTAL DIGIT>
                                / <OCTAL DIGITS> <OCTAL DIGIT>

<HEX DIGIT> ::=                   <OCTAL DIGIT>
                                / 8 / 9 / A / B / C / D / E / F

<HEX DIGITS> ::=                  <HEX DIGIT>
                                / <HEX DIGITS> <HEX DIGIT>

<BIT GROUP>::=                    (4) <HEX DIGITS>
                                / (3) <OCTAL DIGITS>
                                / (2) <QUARTAL DIGITS>
                                / (1) <BINARY DIGITS>

<BIT STRING> ::=                  @<BITS>@

<BITS>::=                         <BIT GROUP> / <HEX DIGITS>
                                / <BITS> <BIT GROUP>
                                / <EMPTY>
```

RESTRICTIONS:

1. IF NO BIT MODE IS SPECIFIED (IE. THE INDICATOR DIGIT IN PARENTHESES IS OMITTED). "HEX" IS ASSUMED. THIS CAN ONLY BE ASSUMED IF THE BIT STRING DOES NOT START WITH A MODE INDICATOR; WHEN THE MODE IS SWITCHED TO "HEX", AN EXPLICIT "(4)" IS REQUIRED.

2. AS NOTED ABOVE, A <COMMENT STRING> MAY APPEAR ANYWHERE WITHIN A <BIT STRING>, BUT NOT WITHIN THE PARENTHESES BOUNDING THE INDICATOR DIGIT. THE PRESENCE OF A <COMMENT STRING> WILL, IN NO WAY, ALTER THE VALUE OF THE <BIT STRING>

CONTAINING IT.

EXAMPLE:
●(3)6330316260/* THIS */313230/* IS */63302560/* THE */
4321626360/* LAST */512523465124/* RECORD */●

```
<STRING> ::=                    <CHARACTER STRING>
                              / <BIT STRING>

<CHARACTER STRING> ::=          "<STRING CHARACTER LIST>"

<STRING CHARACTER LIST> ::=   <EMPTY>
                              / <STRING CHARACTER>
                                <STRING CHARACTER LIST>

<STRING CHARACTER> ::=          <DIGIT> / <LETTER> / <SPECIAL CHARACTER>
                              / "" / @ / # / %
```

RESTRICTIONS:

1. IF A QUOTE SIGN IS DESIRED IN A CHARACTER STRING, THEN TWO ADJACENT QUOTE SIGNS MUST APPEAR IN THE TEXT.

```
EXAMPLE:    DECLARE  STRING CHARACTER (6);
                     QUOTE CHARACTER (1)
            STRING←"AB""CDE";
            QUOTE ←"""";
```

AFTER EXECUTION, STRING WILL CONTAIN: AB"CDE, AND QUOTE WILL CONTAIN: ".

NOTE: A <CHARACTER STRING> MAY CONTAIN A MAXIMUM OF 256 CHARACTERS.

```
<CONSTANT> ::=                  <NUMBER> / <STRING> / TODAYS.DATE
                              / SEQUENCE.NUMBER
                              / HEX.SEQUENCE.NUMBER
```

NOTE: "TODAYS.DATE" REPRESENTS THE DATE AND TIME OF COMPILATION OF THE PROGRAM. IT IS THE SAME AS THE DATE AND TIME APPEARING AT THE TOP OF THE PROGRAM LISTING. IT IS A CHARACTER STRING WITH THE FOLLOWING FORMAT --
"MM/DD/YY HH:MM"

NOTE: "SEQUENCE.NUMBER" REPRESENTS A <CHARACTER STRING> OF 8 CHARACTERS WHICH IS THE SEQUENCE NUMBER OF THE CURRENT SOURCE IMAGE BEING CONPILED.

"HEX.SEQUENCE.NUMBER" REPRESENTS A BIT STRING

OF 8 (HEX) DIGITS WHICH IS THE SEQUENCE NUMBER OF THE CURRENT SOURCE IMAGE LINE BEING COMPILED. IF THIS SEQUENCE FIELD IS BLANK, THEN HEX.SEQUENCE.NUMBER = @00000000@.

IF THE CURRENT SOURCE IMAGE LINE SEQUENCE NUMBER IS 12753000, THEN ON THIS LINE:

SEQUENCE.NUMBER = "12753000"
HEX.SEQUENCE.NUMBER = @12753000@

BASIC STRUCTURE OF THE SDL PROGRAM

```
<PROGRAM> ::=                    <DECLARATION STATEMENT LIST>
                                 <PROCEDURE STATEMENT LIST>
                                 <EXECUTABLE STATEMENT LIST>
                                 FINI

<DECLARATION STATEMENT
LIST> ::=                        <EMPTY>
                               / <DECLARATION STATEMENT>
                                 <DECLARATION STATEMENT LIST>

<DECLARATION STATEMENT> ::=      <DECLARE STATEMENT>;
                               / <DEFINE STATEMENT>;
                               / <FILE DECLARATION STATEMENT>;
                               / <SWITCH FILE DECLARATION
                                 STATEMENT>;
                               / <FORWARD DECLARATION>
                               / <USE STATEMENT>;
                               / <SEGMENT STATEMENT>
                                 <DECLARATION STATEMENT>

<PROCEDURE STATEMENT
LIST> ::=                        <EMPTY>
                               / <PROCEDURE STATEMENT>;
                                 <PROCEDURE STATEMENT LIST>

<PROCEDURE STATEMENT> ::=        <PROCEDURE DEFINITION>
                               / <SEGMENT STATEMENT>
                                 <PROCEDURE STATEMENT>

<EXECUTABLE STATEMENT
LIST> ::=                        <EXECUTABLE STATEMENT>
                               / <EXECUTABLE STATEMENT>
                                 <EXECUTABLE STATEMENT LIST>

<EXECUTABLE STATEMENT> ::=       <DO GROUP>;
                               / <IF STATEMENT>
                               / <CASE STATEMENT>;
                               / <ASSIGNMENT STATEMENT>;
                               / <EXECUTE-PROCEDURE STATEMENT>;
                               / <EXECUTE-FUNCTION STATEMENT>;
                               / <GROUP TERMINATION STATEMENT>;
                               / <I/O CONTROL STATEMENT>
                               / <MODIFY INSTRUCTION>;
                               / <NULL STATEMENT>
                               / <STOP STATEMENT>;
                               / <FILE ATTRIBUTE STATEMENT>;
                               / <SEND STATEMENT>
                               / <RECEIVE STATEMENT>
                               / <ACCESS FILE HEADER STATEMENT>
```

/ <SEARCH STATEMENT>
/ <ZIP STATEMENT>
/ <ARRAY PAGE TYPE STATEMENT>
/ <COROUTINE STATEMENT>
/ <WAIT STATEMENT>;
/ <SEGMENT STATEMENT>
<EXECUTABLE STATEMENT>

A PROGRAM, WRITTEN IN SDL, MUST FOLLOW THE SEQUENTIAL STRUCTURE DESCRIBED IN THE ABOVE SYNTAX. THAT IS, THE EXECUTABLE SECTION OF THE PROGRAM MAY NOT APPEAR UNTIL ALL PROCEDURES HAVE BEEN DEFINED, AND PROCEDURES MAY NOT BE DEFINED BEFORE THE FORMATS OF DATA ITEMS (VARIABLES, ARRAYS, ETC.) HAVE BEEN DECLARED. "FINI" MUST PHYSICALLY OCCUR AS THE FINAL STATEMENT IN THE PROGRAM.

THE PROCEDURE STATEMENT, (INCLUDING DECLARATION, PROCEDURE, AND EXECUTABLE STATEMENTS), IS THE BASIC STRUCTURE IN SDL. AN SDL PROGRAM IS A COLLECTION OF PROCEDURES, EACH OF WHICH CAN BE DESCRIBED FOR CONCEPTUAL PURPOSES AS A MICROCOSM OF THE PROGRAM. ANY GIVEN PROCEDURE MAY CONTAIN A COLLECTION OF OTHER PROCEDURES WITHIN ITSELF. THIS PROCESS IS KNOWN AS "NESTING".

THE "LEXICOGRAPHIC LEVEL" OF ANY STATEMENT IN THE PROGRAM IS EQUAL TO THE NUMBER OF PROCEDURES IN WHICH IT IS NESTED. THE PROGRAM ITSELF WILL ALWAYS BE LEXIC LEVEL 0, AND NO PROCEDURE MAY HAVE A LEXIC LEVEL GREATER THAN 15. THE DIAGRAM IN FIGURE 1 ILLUSTRATES PROCEDURE NESTING AND LEXIC LEVELS.

IT IS IMPORTANT TO UNDERSTAND THE RELATIONSHIPS BETWEEN THESE NESTED PROCEDURES. AS FIGURE 1. INDICATES, THE NAME OF ANY GIVEN PROCEDURE IS CONTAINED IN THE PROCEDURE IN WHICH IT IS NESTED AT THE NEXT LOWER LEXIC LEVEL. FOR EXAMPLE, PROCEDURE D IS A LEXIC LEVEL 2 PROCEDURE, HOWEVER, ITS NAME, "D", IS PART OF LEXIC LEVEL 1.

THE "SCOPE" OF ANY GIVEN PROCEDURE IS RECURSIVELY DEFINED AS:

1) THE PROCEDURE ITSELF,

2) ANY PROCEDURE(S) NESTED WITHIN THE PROCEDURE,

3) ANY PROCEDURE (AND ITS NESTED PROCEDURES) WHOSE NAME APPEARS AT THE SAME LEXIC LEVEL AND WITHIN THE SAME PROCEDURE AS ITS OWN NAME, AND

4) THE PROCEDURE IN WHICH ITS OWN NAME IS DEFINED.

IN FIGURE 1., ONE CAN SEE THAT THE SCOPE OF PROCEDURE B INCLUDES:

1) ITSELF, I.E., PROCEDURE B

2) THE NESTED PROCEDURES WITHIN B (C AND D),

3) THE OTHER PROCEDURES DEFINED AT LL0: E (AND ITS NESTED PROCEDURES F AND G) AND PROCEDURE H (AND ITS NESTED PROCEDURES J, K, L, M, N, AND P.

4) THE PROCEDURE WHICH DEFINES B, IN THIS CASE, THE PROGRAM A.

NOTE: ALL THE LEXIC LEVEL 0 PROCEDURES HAVE SCOPE TO EACH OTHER. THIS OCCURS BECAUSE OF RULE 4 ABOVE, WHEREIN THE PROGRAM ITSELF IS THOUGHT TO BE A "PROCEDURE".

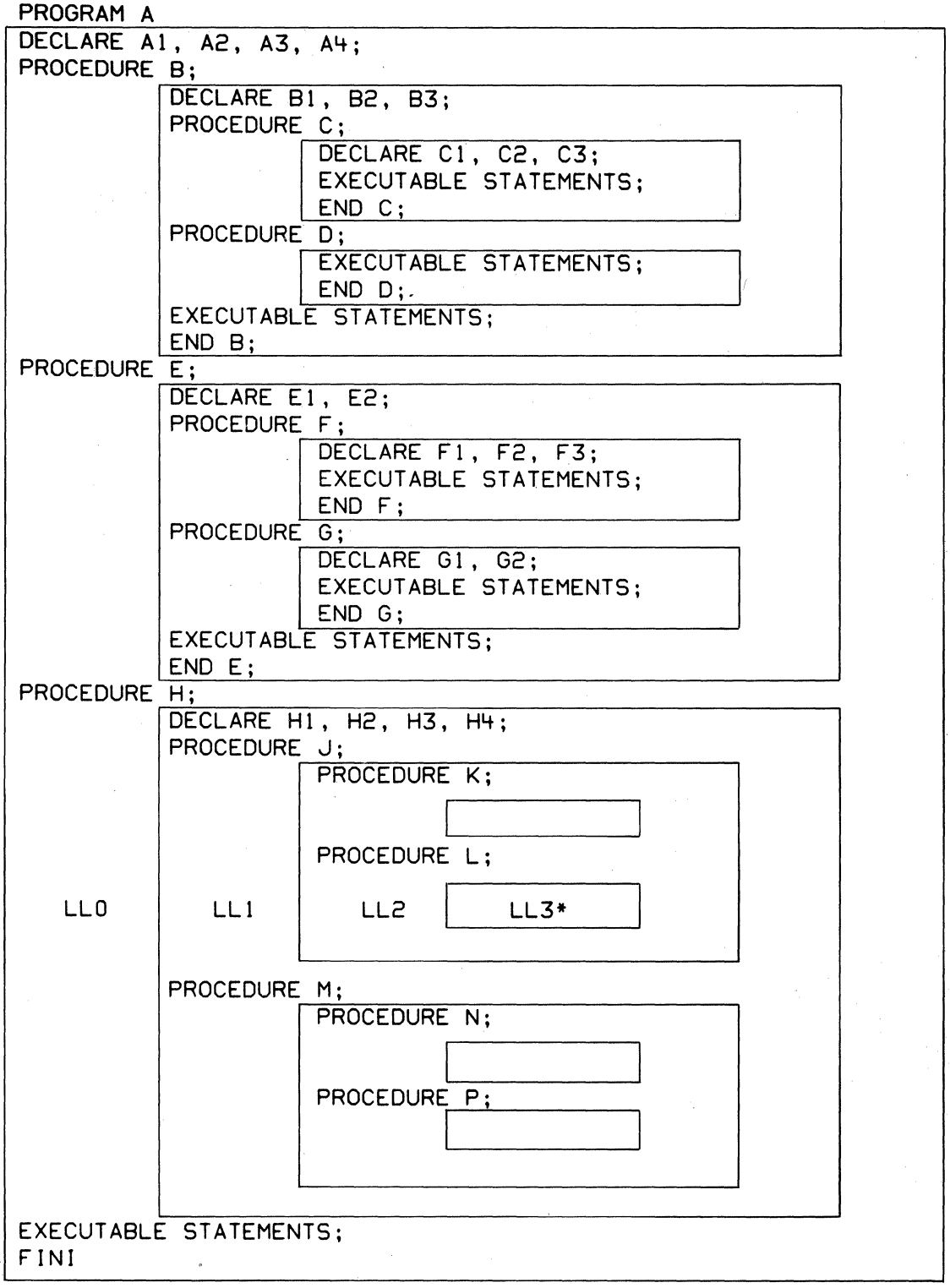
IN THE SAME MANNER, THE SCOPE OF PROCEDURE J INCLUDES J, K, L, M, N, P, AND H.

BY UNDERSTANDING THE RELATIONSHIPS BETWEEN THE VARIOUS PROCEDURES, IT IS POSSIBLE TO DETERMINE WHICH PROCEDURES MAY BE INVOKED BY ANY GIVEN PROCEDURE. SDL HAS BEEN DEFINED SO THAT ANY PROCEDURE X MAY CALL OR INVOKE ANY PROCEDURE Y, IF THE SCOPE OF Y ENCOMPASSES X.

IN FIGURE 1., PROCEDURE J MAY CALL PROCEDURES J,K,L,M,H,E, AND B BECAUSE EACH OF THESE CONTAINS J IN ITS SCOPE.

NOTE: J CANNOT CALL THE PROGRAM A SINCE THE NAME OF THE PROGRAM, IF THERE IS ONE, EXISTS OUTSIDE THE PROGRAM AND IS, THEREFORE, NOT COMPILED; HOWEVER, J MAY ACCESS THE DATA CONTAINED IN A (I.E., A1, A2, A3, AND A4).

FIGURE 2 SHOWS THE RELATIONSHIP BETWEEN SCOPE AND CALLING ABILITY FOR PROGRAM A.



* LL = LEXICOGRAPHIC LEVEL

FIG 1. PROCEDURE NESTING

CALLING PROCEDURES

PROCEDURE / SCOPE	A	B	C	D	E	F	G	H	J	K	L	M	N	P
B	*	*	*	*	*	*	*	*	*	*	*	*	*	*
C		*	*	*										
D		*	*	*										
E	*	*	*	*	*	*	*	*	*	*	*	*	*	*
F					*	*	*							
G					*	*	*							
H	*	*	*	*	*	*	*	*	*	*	*	*	*	*
J								*	*	*	*	*	*	*
K									*	*	*			
L									*	*	*			
M								*	*	*	*	*	*	*
N												*	*	*
P												*	*	*

NOTE: TO FIND THE SCOPE OF A PROCEDURE, FIND THE PROCEDURE IN THE COLUMN OF PROCEDURE NAMES. THE HORIZONTAL ROWS TO THE RIGHT INDICATE THE PROCEDURES IN ITS SCOPE. THE PROCEDURES WHICH MAY BE CALLED BY A GIVEN PROCEDURE ARE MARKED IN THE VERTICAL COLUMNS BELOW THAT CALLING PROCEDURE.

FIG 2. SCOPE AND CALLING ABILITY

SEGMENT STATEMENT

```
<SEGMENT STATEMENT> ::= SEGMENT (<SEGMENT IDENTIFIER>);
                      / SEGMENT.PAGE (<SEGMENT IDENTIFIER>
                        OF <PAGE IDENTIFIER>);

<SEGMENT IDENTIFIER> ::= <IDENTIFIER>

<PAGE IDENTIFIER> ::= <IDENTIFIER>
```

AS THE BNF INDICATES, THE <SEGMENT STATEMENT> MAY OCCUR ANYWHERE WITHIN AN SDL PROGRAM. ITS PURPOSE IS TO REDUCE THE CORE REQUIREMENT OF THE PROGRAM BY ALLOWING SEGMENTS TO OVERLAY EACH OTHER.

THERE IS A MAXIMUM OF 16 PAGES WITH 64 SEGMENTS PER PAGE. THE SEGMENT NAMES REPRESENT A PAGE NUMBER-SEGMENT NUMBER PAIR.

IT IS ONLY NECESSARY TO SPECIFY SEGMENT.PAGE ONCE FOR EACH PAGE. EVERY SUBSEQUENT SEGMENT WILL BE COMPILED TO THAT PAGE UNTIL ANOTHER SEGMENT.PAGE IS ENCOUNTERED.

IF THERE ARE NO SEGMENT.PAGE SPECIFICATIONS, ALL SEGMENTS WILL BE COMPILED TO PAGE ZERO, AND THERE MAY BE NO MORE THAN 64 SEGMENTS TOTAL. IF A PROGRAM IS TO BE SEGMENTED, THE FIRST STATEMENT MUST BE A <SEGMENT STATEMENT>. OTHERWISE A WARNING MESSAGE WILL APPEAR IN THE SOURCE LISTING.

THERE ARE TWO TYPES OF SEGMENTS: "PERMANENT" AND "TEMPORARY". EVERY STATEMENT FOLLOWING A PERMANENT <SEGMENT STATEMENT> WILL BE COMPILED TO THAT SEGMENT UNTIL ANOTHER <SEGMENT STATEMENT> IS READ. NON-CONSECUTIVE STATEMENTS MAY BE COMPILED TO THE SAME SEGMENT BY USING THE SAME <SEGMENT IDENTIFIER>. NOTE, HOWEVER, THAT <DO GROUP>S (SEE "DO GROUPS") AND PROCEDURES MUST END IN THE SAME SEGMENT IN WHICH THEY BEGIN.

THE FOLLOWING EXAMPLE ILLUSTRATES THE USE OF THE "PERMANENT" <SEGMENT STATEMENT>.

```
SEGMENT (XX);
DECLARE A1, A2, A3, A4;
PROCEDURE B;
     DECLARE B1, B2, B3;
     SEGMENT (YY);
     PROCEDURE C;
           .
           .
           .
     END C;
     PROCEDURE D;
           .
           .
           .
     END D;
SEGMENT (XX);
     .
     .
     .
END B;
     .
     .
     .
FINI
```

ONLY PROCEDURES C AND D HAVE BEEN COMPILED TO THE SEGMENT "YY". SEGMENT "XX" IS SEGMENT ZERO AND INCLUDES EVERYTHING ELSE.

A <SEGMENT STATEMENT> IS TREATED AS "TEMPORARY" ONLY WHEN IT PRECEDES A "SUBORDINATE EXECUTABLE STATEMENT" WITHIN ANY OF THE FOLLOWING STATEMENTS:

<ACCESS FILE HEADER STATEMENT>	<SEARCH DIRECTORY STATEMENT>
<CASE STATEMENT>	<SEND STATEMENT>
<IF STATEMENT>	<SPACE STATEMENT>
<READ STATEMENT>	<WRITE STATEMENT>
<RECEIVE STATEMENT>	<OPEN STATEMENT>

IN THESE SPECIFIC CASES, THE SEGMENT CHANGE APPLIES ONLY TO THE SUBORDINATE STATEMENT FOLLOWING IT. FOR EXAMPLE, THE SYNTAX FOR THE <IF STATEMENT> COULD BE WRITTEN AS FOLLOWS:

```
<IF STATEMENT> ::=    IF <EXPRESSION
                      THEN <SUBORDINATE EXECUTABLE STATEMENT>
                    / IF <EXPRESSION>
                      THEN <SUBORDINATE EXECUTABLE STATEMENT>
                      ELSE <SUBORDINATE EXECUTABLE STATEMENT>
```

THE SEGMENTATION OF A HYPOTHETICAL <IF STATEMENT> IS PRESENTED BELOW TO ILLUSTRATE THE USE OF A "TEMPORARY" <SEGMENT STATEMENT>.

```
SEGMENT (A);
PROCEDURE X;
      .
      .
      .
      IF Y>Z THEN Y:=Z; ELSE
      SEGMENT (B);
      DO SOME.FUNCTION;       *
            .                 *
            .                 *
            .                 *
      END SOME.FUNCTION;      *
            .
            .
            .
      Z := C+D
END X;
```

* COMPILED TO SEGMENT (B)

BECAUSE THE <DO GROUP>, "SOME.FUNCTION", IS A SUBORDINATE <EXECUTABLE STATEMENT> IN THE <IF STATEMENT>, SEGMENT (B) AUTOMATICALLY ENDS WHEN THE <DO GROUP> IS TERMINATED. ALL STATEMENTS FOLLOWING ARE COMPILED TO SEGMENT (A).

NOTICE THE DISTINCTION BETWEEN SEGMENT (A), A "PERMANENT" <SEGMENT STATEMENT>, AND SEGMENT (B), A "TEMPORARY" ONE.

DECLARATION STATEMENT

DATA TYPES

THREE MAIN TYPES OF DATA FIELDS MAY BE DECLARED IN SDL:

1) BIT
2) CHARACTER
3) FIXED

A BIT FIELD CONSISTS OF A NUMBER OF BITS SPECIFIED BY A NUMBER IN PARENTHESES FOLLOWING THE RESERVED WORD "BIT". THE FIELD MAY BE A MAXIMUM OF 65535 BITS.

A CHARACTER FIELD IS A NUMBER OF CHARACTERS, 8 BITS EACH, SPECIFIED BY A NUMBER IN PARENTHESES FOLLOWING THE RESERVED WORD "CHARACTER". THE FIELD MAY BE A MAXIMUM OF 8191 CHARACTERS.

A FIXED DATA FIELD IS A 24-BIT SIGNED NUMERIC FIELD WHERE THE HIGH ORDER BIT IS INTERPRETED AS THE SIGN. NEGATIVE NUMBERS ARE REPRESENTED IN 2-S COMPLEMENT FORM.

THE RANGE OF SIGNED NUMBERS (I.E., FIXED DATA FIELDS) IS -(2 EXP 23) TO (2 EXP 23)-1. THE RANGE OF UNSIGNED NUMBERS (BIT DATA FIELDS) IS 0 TO (2 EXP 24)-1. BIT FIELDS, AS NOTED ABOVE, ARE NOT RESTRICTED TO 24 BITS. HOWEVER, FOR ARITHMETIC PURPOSES, ONLY THE LOW-ORDER 24 BITS WILL BE CONSIDERED.

DECLARE STATEMENT

```
<DECLARE STATEMENT> ::=      DECLARE <DECLARE ELEMENT>
                           / <DECLARE STATEMENT>, <DECLARE ELEMENT>

<DECLARE ELEMENT> ::=        <DECLARED PART>
                             <TYPE PART>
                           / <STRUCTURE LEVEL NUMBER>
                             <STRUCTURE DECLARED PART>
                             <STRUCTURE TYPE PART>
                           / DYNAMIC <SIMPLE IDENTIFIER>
                             <DYNAMIC TYPE PART>
                           / PAGED <ELEMENTS-PER-PAGE PART>
                             <ARRAY IDENTIFIER> <ARRAY BOUND>
                             <TYPE PART>
```

THE <DECLARE STATEMENT> SPECIFIES THE ADDRESSES AND CHARACTERISTICS OF CONTENTS OF CORE STORAGE AREAS.

ANY NUMBER OF <DECLARE ELEMENT>S MAY BE DECLARED IN ONE <DECLARE STATEMENT>, AND MUST BE SEPARATED BY COMMAS. BEST CODE IS GENERATED IF ALL ELEMENTS ARE DECLARED WITHIN ONE <DECLARE STATEMENT>. (SEE APPENDIX VII).

THE MAXIMUM NUMBER OF DATA ELEMENTS (INCLUDING FILLERS, DUMMYS, AND IMPLICIT FILLERS) CONTAINED IN ONE STRUCTURE VARIES AS TO THE COMPILIER BEING USED, (CURRENTLY 50 - SMALL VERSION, 75 - LARGE VERSION). ANY ATTEMPT TO DECLARE MORE WILL CAUSE A TABLE OVERFLOW ERROR TO BE DETECTED AT COMPILE TIME.

AN ARRAY MAY HAVE A MAXIMUM OF 65535 ELEMENTS, EACH BEING A MAXIMUM OF 65535 BITS (8191 CHARACTERS).

THE FIVE TYPES OF <DECLARE ELEMENT>S ARE EACH DISCUSSED BELOW.

NON-STRUCTURED DECLARATIONS

```
<DECLARE ELEMENT> ::=          <DECLARED PART>
                               <TYPE PART>/...

<DECLARED PART> ::=              <COMPLEX IDENTIFIER>
                               / (<COMPLEX IDENTIFIER LIST>)
                               / <COMPLEX IDENTIFIER> REMAPS
                                 <REMAP IDENTIFIER>

<COMPLEX IDENTIFIER
LIST> ::=                        <COMPLEX IDENTIFIER>
                               / <COMPLEX IDENTIFIER>,
                                 <COMPLEX IDENTIFIER LIST>

<COMPLEX IDENTIFIER> ::=         <SIMPLE IDENTIFIER>
                               / <ARRAY IDENTIFIER> <ARRAY BOUND>

<SIMPLE IDENTIFIER> ::=          <IDENTIFIER>

<ARRAY IDENTIFIER> ::=           <IDENTIFIER>

<ARRAY BOUND>       ::=          (<NUMBER>)

<REMAP IDENTIFIER> ::=           BASE
                               / <SIMPLE IDENTIFIER>
                               / <ARRAY IDENTIFIER>

<TYPE PART> ::=                  FIXED
                               / CHARACTER <FIELD SIZE>
                               / BIT <FIELD SIZE>

<FIELD SIZE> ::=                 (<NUMBER>)
```

DATA MAY BE DECLARED AS SIMPLE, HAVING ONE OCCURRENCE, OR AS SUBSCRIPTED, HAVING AS MANY OCCURRENCES AS SPECIFIED BY THE <ARRAY BOUND>.

THE <TYPE PART> SPECIFIES THE TYPE OF DATA IN THE FIELD AND THE FIELD SIZE.

AS THE SYNTAX INDICATES, DIFFERENT DATA FIELDS HAVING THE SAME FORMAT MAY BE DECLARED COLLECTIVELY AS A <COMPLEX IDENTIFIER LIST>.

THE FOLLOWING EXAMPLES ILLUSTRATE THE VARIOUS OPTIONS AVAILABLE IN THIS TYPE OF <DECLARATION STATEMENT>.

```
DECLARE A FIXED,
        B CHARACTER (10),
        C BIT (40),
        (D, E, F (5)) BIT (10),
        G (20) FIXED,
        H (5) CHARACTER (6);
```

WHERE

1. A IS A 24 BIT SIGNED NUMERIC FIELD.

2. B IS A 10 BYTE CHARACTER FIELD.

3. C IS A 40 BIT FIELD.

4. D AND E ARE 10-BIT FIELDS EACH.

5. F IS ALSO A 10-BIT FIELD AND OCCURS 5 TIMES.

6. G OCCURS 20 TIMES AND IS A 24-BIT SIGNED NUMERIC FIELD.

7. H IS A 6-BYTE CHARACTER FIELD OCCURRING 5 TIMES.

DATA FIELDS MAY BE RE-FORMATTED BY THE USE OF THE REMAPPING DEVICE:

<COMPLEX IDENTIFIER> REMAPS <REMAP IDENTIFIER> <TYPE PART>

REMAPPING IS SUBJECT TO THE SAME GENERAL RULES DISCUSSED ABOVE. THE FOLLOWING EXAMPLE BEST ILLUSTRATES ITS USE.

```
        A FIXED, B BIT (50),
        AA REMAPS A CHARACTER (3),
        BB(2) REMAPS B FIXED;
```

NOTE THAT BB SPECIFIES 48-BITS (OR 2 ELEMENTS, 24-BITS EACH). THE LAST TWO BITS WILL BE CONSIDERED AS AN IMPLIED FILLER BY THE COMPILER. A FIELD MAY NOT BE REMAPPED LARGER THAN ITS ORIGINAL SIZE.

THERE IS NO LIMIT ON THE NUMBER OF TIMES A FIELD MAY BE REMAPPED. A FIELD WHICH HAS REMAPPED ANOTHER MAY ITSELF BE REMAPPED. THE REMAP OPTION SPECIFIES THAT THE IDENTIFIER ON THE LEFT SIDE OF THE RESERVED WORD "REMAPS" WILL HAVE THE SAME STARTING ADDRESS AS THE IDENTIFIER ON THE RIGHT SIDE.

FOR RULES CONCERNING THE REMAPPING OF DYNAMIC OR FORMAL DECLARATIONS, SEE THOSE SECTIONS.

A DATA FIELD MAY BE REMAPPED TO BASE WHICH WILL GIVE THE FIELD A RELATIVE ADDRESS OF ZERO. FOR EXAMPLE:

```
DECLARE X REMAPS BASE BIT(7);
```

THIS DEVICE IS USED AS A FREE STANDING DECLARATION SINCE IT DOES NOT REMAP A PREVIOUSLY DECLARED DATA ITEM AND IS USED PRIMARILY WITH DATA TO BE INDEXED (SEE "ADDRESS VARIABLES").

STRUCTURED DECLARATIONS

```
<DECLARE ELEMENT> ::=          .../<STRUCTURE LEVEL NUMBER>
                               <STRUCTURE DECLARED PART>
                               <STRUCTURE TYPE PART>/...

<STRUCTURE LEVEL
NUMBER> ::=                    <NUMBER>

<STRUCTURE DECLARED
PART> ::=                      <DECLARED PART>
                             / FILLER
                             / <DUMMY PART> REMAPS <REMAP IDENTIFIER>

<DECLARED PART> ::=            SEE "NON-STRUCTURED DECLARATIONS"

<DUMMY PART> ::=               DUMMY <ARRAY BOUND PART>

<ARRAY BOUND PART> ::=         <EMPTY>
                             / <ARRAY BOUND>

<ARRAY BOUND> ::=              (<NUMBER>)

<REMAP IDENTIFIER> ::=         BASE
                             / <SIMPLE IDENTIFIER>
                             / <ARRAY IDENTIFIER>

<STRUCTURE TYPE PART> ::=      <EMPTY>
                             / <TYPE PART>

<TYPE PART> ::=                SEE "NON-STRUCTURED DECLARATIONS"
```

SDL ALLOWS THE STRUCTURING OF DATA WHERE A FIELD MAY BE SUBDIVIDED INTO A NUMBER OF SUB-FIELDS, EACH OF WHICH HAS ITS OWN IDENTIFIER. THE WHOLE STRUCTURE IS ORGANIZED IN A HIERARCHICAL FORM, WHERE THE MOST GENERAL DECLARATION IS A LEVEL 01(OR 1). NO DECLARATION MAY BE ON A LEVEL GREATER THAN 99. A SUBDIVIDED FIELD IS CALLED A GROUP ITEM, AND A FIELD NOT SUBDIVIDED IS KNOWN AS AN ELEMENTARY ITEM.

THE TYPE AND LENGTH OF DATA NEED NOT BE SPECIFIED ON THE GROUP LEVEL. ALL ELEMENTARY ITEMS MUST INDICATE TYPE AND LENGTH, AND THE COMPILER WILL ASSUME TYPE BIT AND ADD THE LENGTHS OF THE COMPONENTS TO DETERMINE THE LENGTH OF THE GROUP ITEM. FOR EXAMPLE:

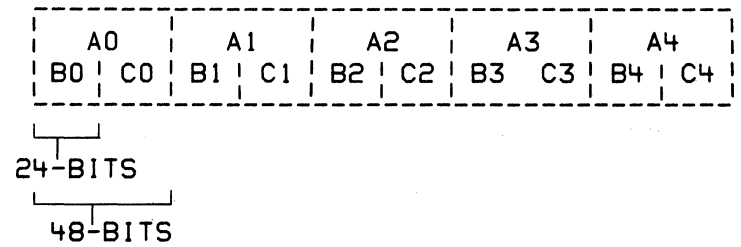
```
DECLARE 01 A,
          02 C,
             03 D BIT(20),
             03 E BIT(30),
          02 D CHARACTER(5);
```

IN THIS EXAMPLE, BOTH A AND C ARE CONSIDERED GROUP ITEMS, WITH A HAVING A TOTAL LENGTH OF 90 BITS AND C BEING 50 BITS LONG.

FILLERS MAY BE USED TO DESIGNATE CERTAIN ELEMENTARY ITEMS WHICH THE PROGRAM DOES NOT REFERENCE. IF THE FILLER IS THE LAST ITEM IN A STRUCTURE, IT MAY BE OMITTED, AND THE COMPILER WILL CONSIDER THE ITEM TO BE AN IMPLIED FILLER. A FILLER MAY NEVER BE USED AS A GROUP ITEM.

IF THE 01 LEVEL GROUP ITEM IS AN ARRAY, IT IS MAPPED AS A CONTIGUOUS AREA IN MEMORY. HOWEVER, SUBDIVISIONS OF THIS ARRAY ARE NOT CONTIGUOUS. IN THE EXAMPLE STRUCTURE BELOW:

```
01 A(5) BIT(48),          01 A(5),
   02 B FIXED,     OR        02 B FIXED,
   02 C FIXED;               02 C FIXED;
```



IF A GROUP ITEM IS AN ARRAY, AN ARRAY SPECIFICATION MAY NOT APPEAR IN ANY SUBORDINATE ITEM; THAT IS, ONLY ONE-DIMENSIONAL ARRAYS ARE ALLOWED. DOWN-LEVEL CARRY OF ARRAY SPECIFICATIONS IS IMPLIED.

STRUCTURED DATA MAY BE REMAPPED IN THE SAME MANNER AS NON-STRUCTURED DATA. IN ADDITION, STRUCTURED DATA MAY BE REMAPPED WITH A DUMMY GROUP IDENTIFIER. THE PURPOSE OF THIS CONSTRUCT IS TO ALLOW THE USER TO REMAP DATA ITEMS WITHOUT HAVING TO DECLARE ANOTHER GROUP ITEM WHICH DESCRIBES THE SAME AREA IN CORE. THUS IN THE FOLLOWING EXAMPLE:

```
01 A BIT(100),
   02 B BIT(20),
   02 C BIT(80);
```

"A" MIGHT BE REMAPPED AS

```
01 AA REMAPS A BIT(100),
   02 BB BIT(30),
   02 CC BIT(70);
```

OR

```
01 DUMMY REMAPS A BIT(100),
   02 BB BIT(30),
   02 CC BIT(70);
```

BOTH A AND AA IN THE ABOVE EXAMPLE REFER TO THE SAME AREA IN CORE. HENCE AA IS REDUNDANT. DURING RUNTIME, THE DESCRIPTOR FOR AA WILL ALSO BE ON THE STACK.

IF "DUMMY" IS SUBSTITUTED FOR THE IDENTIFIER AA, NO DESCRIPTOR WILL BE GENERATED, HOWEVER BB AND CC WILL BOTH POINT TO A IN THE CORRECT FASHION.

THE USER SHOULD NOTE THE DISTINCTION BETWEEN "DUMMY" AND "FILLER". "DUMMY" IS USED IN CONJUNCTION WITH "REMAPS" TO ELIMINATE THE NECESSITY OF DECLARING A REDUNDANT GROUP ITEM. "FILLER" IS USED IF ONE DESIRES TO SKIP OVER AN AREA OF CORE.

THE FOLLOWING RESTRICTIONS APPLY TO THE USE OF "DUMMY REMAPS":

1. "DUMMY" MAY ONLY BE USED WITH REMAP DECLARATIONS.

2. ALL THE RESTRICTIONS APPLYING TO "REMAPS" APPLY TO "DUMMY REMAPS".

3. "DUMMY" MUST NOT REMAP ANOTHER "DUMMY".

4. "DUMMY" GROUP ITEMS MUST HAVE AT LEAST ONE NON-FILLER COMPONENT.

DYNAMIC DECLARATIONS

```
<DECLARE ELEMENT> ::=          .../ DYNAMIC <SIMPLE IDENTIFIER>
                               <DYNAMIC TYPE PART>/...

<DYNAMIC TYPE PART> ::=          BIT <DYNAMIC FIELD SIZE>
                               / CHARACTER <DYNAMIC FIELD SIZE>

<DYNAMIC FIELD SIZE> ::=       (<EXPRESSION>)
```

THE DYNAMIC DECLARE STATEMENT ALLOWS THE USER TO DECLARE SIMPLE DATA WITH A NON-STATIC FIELD LENGTH. FOR EXAMPLE:

```
PROCEDURE ABX;
      DECLARE DYNAMIC X BIT(A);
```

WHERE "A" MAY BE OF VARIABLE LENGTH. THE VALUE OF THE <EXPRESSION> APPEARING IN THE <DYNAMIC FIELD SIZE> IS USED TO DETERMINE THE NUMBER OF BITS OR CHARACTERS IN THE DECLARED DATA ITEM.

RESTRICTIONS:

1. THE VARIABLES USED IN THE <DYNAMIC FIELD SIZE> MUST HAVE BEEN PREVIOUSLY INITIALIZED.

2. DYNAMICS MAY NOT APPEAR ON LEXIC LEVEL 0.

DYNAMIC VARIABLES MAY BE REMAPPED, HOWEVER A WARNING MESSAGE WILL APPEAR IN THE SOURCE LISTING. IT IS THE PROGRAMMER-S RESPONSIBILITY TO ENSURE THAT A DYNAMIC IS NOT REMAPPED LARGER THAN ALLOWED.

PAGED ARRAY DECLARATIONS

```
<DECLARE ELEMENT > ::=          .../ PAGED <ELEMENTS-PER-PAGE PART>
                                <ARRAY IDENTIFIER> <ARRAY BOUND>
                                <TYPE PART>

<ELEMENTS-PER-PAGE
PART> ::=                       (<NUMBER>)

<ARRAY IDENTIFIER> ::=          <IDENTIFIER>

<ARRAY BOUND> ::=               (<NUMBER>)

<TYPE PART> ::=                 FIXED
                              / CHARACTER <FIELD SIZE>
                              / BIT <FIELD SIZE>

<FIELD SIZE> ::=                (<NUMBER>)
```

THE PAGED ARRAY DECLARATION ALLOWS THE USER TO SEGMENT ARRAYS. THE <ELEMENTS-PER-PAGE PART> SPECIFIES THE NUMBER OF ARRAY ELEMENTS CONTAINED IN EACH SEGMENT. FOR EXAMPLE:

```
PAGED(64) A(4096) BIT(1);
```

IS AN ARRAY OF 4096, 1-BIT ELEMENTS, SEGMENTED INTO 64, 64-ELEMENT SEGMENTS.

RESTRICTIONS:

1. PAGED ARRAYS MAY NOT BE INDEXED.
2. PAGED ARRAYS MAY NOT BE PART OF A STRUCTURE.
3. PAGED ARRAYS MAY NOT BE REMAPPED.
4. THE NUMBER OF ELEMENTS PER PAGE MUST BE A POWER OF 2, AND MAY NOT EXCEED 65535.

FILE DECLARATIONS

```
<FILE DECLARATION
STATEMENT> ::=                  FILE <FILE DECLARE ELEMENT LIST>

<FILE DECLARE
ELEMENT LIST> ::=               <FILE DECLARE ELEMENT>
                              / <FILE DECLARE ELEMENT>,
                                <FILE DECLARE ELEMENT LIST>

<FILE DECLARE ELEMENT> ::=      <FILE IDENTIFIER><FILE ATTRIBUTE PART>

<FILE IDENTIFIER> ::=           <IDENTIFIER>

<FILE ATTRIBUTE PART> ::=       <EMPTY>
                              / (<FILE ATTRIBUTE LIST>)

<FILE ATTRIBUTE LIST> ::=       <FILE ATTRIBUTE>
                              / <FILE ATTRIBUTE>,
                                <FILE ATTRIBUTE LIST>

<FILE ATTRIBUTE> ::=            <LABEL PART>
                              / <DEVICE PART>
                              / <MODE PART>
                              / <BUFFERS PART>
                              / <VARIABLE RECORD PART>
                              / <LOCK PART>
                              / <SAVE FACTOR PART>
                              / <RECORD SPECIFICATION PART>
                              / <REEL NUMBER PART>
                              / <DISK FILE DESCRIPTION PART>
                              / <PACK-ID PART>
                              / <OPEN OPTION PART>
                              / <ALL.AREAS.AT.OPEN PART>
                              / <AREA.BY.CYLINDER PART>
                              / <EU.ASSIGNMENT PART>
                              / <MULTI.PACK PART>
                              / <USE.INPUT.BLOCKING PART>
                              / <SORTER STATION PART>
                              / <END.OF.PAGE PART>
                              / <REMOTE.KEY PART>
                              / <NUMBER.OF.STATIONS PART>
                              / <QUEUE.FAMILY.SIZE PART>
                              / <FILE TYPE PART>
                              / <WORK FILE PART>
                              / <LABEL TYPE PART>
```

ALL ATTRIBUTES ARE OPTIONAL, AS THE ABOVE SYNTAX INDICATES. DEFAULT STATUS WILL AUTOMATICALLY BE SET FOR OMITTED ATTRIBUTES AS FOLLOWS.

SYNTAX:

<LABEL PART> ::=	LABEL = <FILE IDENTIFICATION PART>
<FILE IDENTIFICATION PART> ::=	<MULTI-FILE IDENTIFICATION> / <MULTI-FILE IDENTIFICATION> <SLASH> <FILE IDENTIFICATION>
<MULTI-FILE IDENTIFICATION> ::=	<CHARACTER STRING>
<FILE IDENTIFICATION> ::=	<CHARACTER STRING>

WHERE:

<FILE IDENTIFIER> IS	A FILE OR PROGRAM IDENTIFIER BY WHICH THE PROGRAM IDENTIFIES THE FILE
<MULTI-FILE IDENTIFICATION> AND <FILE IDENTIFICATION> ARE	NAME OR CONTENTS OF IDENTIFICATION FIELD ON FILE LABEL OR DISK DIRECTORY BY WHICH THE SYSTEM IDENTIFIES THE FILE

FORMAT:

```
LABEL = "NAME.1" / "NAME.2"
OR
LABEL = "NAME.1"
```

EXAMPLE:

```
DECLARE INV.DATA.1 FILE
(LABEL = "RCD.TAPE" / "FILE.1");
```

NOTE: THE SYSTEM WILL USE ONLY THE FIRST TEN CHARACTERS OF THE "NAME".

DEFAULT: IF LABEL(S) IS (ARE) NOT SPECIFIED, THE INTERNAL FILE NAME, I.E., <FILE IDENTIFIER>, IS MOVED TO <MULTI-FILE IDENTIFICATION>, AND BLANKS ARE MOVED TO <FILE IDENTIFICATION> IN THE FPB (FILE PARAMETER BLOCK).

```
SYNTAX:     <DEVICE PART> ::=                DEVICE = <DEVICE SPECIFIER>

            <DEVICE SPECIFIER> ::=           CARD / TAPE
                                           / MULTI.FUNCTION.CARD
                                           / TAPE.7
                                           / TAPE.9
                                           / TAPE.PE
                                           / TAPE.NRZ
                                           / DISK <ACCESS MODE>
                                           / DISK.PACK <ACCESS MODE>
                                           / DISK.FILE <ACCESS MODE>
                                           / DISK.PACK.CENTURY <ACCESS MODE>
                                           / DISK.PACK.CAELUS <ACCESS MODE>
                                           / CARD.READER
                                           / CARD.PUNCH <DEVICE OPTION>
                                           / MFCU
                                           / PRINTER <DEVICE OPTION>
                                           / PUNCH <DEVICE OPTION>
                                           / PAPER.TAPE.PUNCH
                                             <DEVICE OPTION>
                                           / PUNCH.96 <DEVICE OPTION>
                                           / READER.PUNCH <DEVICE OPTION>
                                           / READER.PUNCH.PRINTER
                                             <DEVICE OPTION>
                                           / PUNCH.PRINTER <DEVICE OPTION>
                                           / READER.96
                                           / PAPER.TAPE.READER
                                           / SORTER.READER
                                           / READER.SORTER
                                           / SPO
                                           / CASSETTE
                                           / REMOTE
                                           / QUEUE

            <ACCESS MODE> ::=                <EMPTY> / SERIAL / RANDOM

            <DEVICE OPTION> ::=              <EMPTY>
                                           / <BACKUP OPTION>
                                           / <SPECIAL FORMS OPTION>
                                           / <SPECIAL FORMS OPTION>
                                             <BACKUP OPTION>

            <BACKUP OPTION> ::=              <BACKUP SPECIFIER>
                                           / OR <BACKUP SPECIFIER>

            <BACKUP SPECIFIER> ::=           BACKUP / BACKUP TAPE
                                           / BACKUP DISK

            <SPECIAL FORMS OPTION> ::=       FORMS

FORMAT:     DEVICE =    CARD
                        CARD.READER
```

```
      TAPE
      MULTI.FUNCTION.CARD
      TAPE.7
      TAPE.9
      TAPE.PE
      TAPE.NRZ
   ** DISK
   ** DISK.PACK
   ** DISK.FILE
   ** DISK.PACK.CENTURY
   ** DISK.PACK.CAELUS
   *  CARD.PUNCH                      %MARKED HARDWARE ONLY
   *  PRINTER                         %MARKED HARDWARE ONLY
   *  PRINTER FORMS                   %MARKED HARDWARE ONLY
   *  PUNCH                           %MARKED HARDWARE ONLY
   *  PUNCH FORMS                     %MARKED HARDWARE ONLY
   *  PAPER.TAPE.PUNCH                %MARKED HARDWARE ONLY
   *  PAPER.TAPE.PUNCH FORMS          %MARKED HARDWARE ONLY
   *  PUNCH.96                        %MARKED HARDWARE ONLY
   *  PUNCH.96 FORMS                  %MARKED HARDWARE ONLY
   *  READER.PUNCH                    %MARKED HARDWARE ONLY
   *  READER.PUNCH FORMS              %MARKED HARDWARE ONLY
   *  READER.PUNCH.PRINTER            %MARKED HARDWARE ONLY
   *  READER.PUNCH.PRINTER FORMS      %MARKED HARDWARE ONLY
   *  PUNCH.PRINTER                   %MARKED HARDWARE ONLY
   *  PUNCH.PRINTER FORMS             %MARKED HARDWARE ONLY
      READER.96
      PAPER.TAPE.READER
      SPO
      MFCU
      SORTER.READER
      READER.SORTER
      CASSETTE
      REMOTE
      QUEUE

*  MAY OR MAY NOT BE FOLLOWED BY ANY SINGLE OPTION BELOW:

      BACKUP
      BACKUP TAPE
      BACKUP DISK
      OR BACKUP
      OR BACKUP TAPE
      OR BACKUP DISK

** MAY OR MAY NOT BE FOLLOWED BY ANY SINGLE OPTION BELOW:

      SERIAL
      RANDOM

EXAMPLES: DEVICE = TAPE
          DEVICE = PRINTER BACKUP
          DEVICE = PRINTER FORMS BACKUP TAPE
```

DEFAULT: IN THE ABSENCE OF ANY SPECIFICATION, TAPE WILL BE ASSUMED BY THE FPB.

SYNTAX: <MODE PART> ::= MODE = <MODE SPECIFIER>

<MODE SPECIFIER> ::= <FILE PARITY PART>
/ <TRANSLATION PART>
/ <FILE PARITY PART>
<TRANSLATION PART>

<FILE PARITY PART> ::= ODD / EVEN

<TRANSLATION PART>::= EBCDIC / ASCII / BCL / BINARY

FORMAT:
```
MODE = BCL          OR
OR                  MODE = ASCII
MODE = EVEN         OR
OR                  MODE = EVEN ASCII
MODE = EVEN BCL
OR
```

DEFAULT: DEFAULT IS ODD OR EBCDIC, WHICHEVER IS APPLICABLE.

SYNTAX: <BUFFERS PART> ::= BUFFERS =
<NUMBER OF BUFFERS>

<NUMBER OF BUFFERS> ::= <NUMBER>

FORMAT: BUFFERS = NUMBER

DEFAULT: IF NOT SPECIFIED, BUFFERS WILL BE SET TO 1 IN THE FPB.

SYNTAX: <VARIABLE RECORD PART> ::= VARIABLE

FORMAT: VARIABLE

DEFAULT: NOT VARIABLE, I.E., FIXED-SIZE RECORDS.

SYNTAX: <LOCK PART> ::= LOCK

FORMAT: LOCK

DEFAULT: LOCK IS NOT SET.

SYNTAX: <SAVE FACTOR PART> ::= SAVE = <SAVE FACTOR>

<SAVE FACTOR> ::= <NUMBER>

FORMAT: SAVE = NUMBER (OF DAYS TO SAVE FILE)

DEFAULT: IF NOT SPECIFIED, THE SAVE SPECIFIER WILL BE SET TO 30 IN THE FPB.

SYNTAX: <RECORD SPECIFICATION PART> ::= RECORDS = <RECORD SIZE SPECIFIER>

<RECORD SIZE SPECIFIER> ::= <PHYSICAL RECORD SIZE>
/ <LOGICAL RECORD SIZE>
<SLASH>
<LOGICAL RECORDS PER PHYSICAL RECORD>

<PHYSICAL RECORD SIZE> ::= <NUMBER>

<LOGICAL RECORD SIZE> ::= <NUMBER>

<LOGICAL RECORDS PER PHYSICAL RECORD> ::= <NUMBER>

FORMAT:
```
RECORDS = NUMBER
OR
RECORDS = NUMBER / NUMBER
```

NOTE: <PHYSICAL RECORD SIZE> INDICATES THE NUMBER OF CHARACTERS PER BLOCK; <LOGICAL RECORD SIZE>, THE NUMBER OF CHARACTERS PER RECORD.

EXAMPLE:
```
RECORDS = 1200
OR
RECORDS = 120 / 10
```

DEFAULT: IN THE ABSENCE OF RECORD SPECIFICATIONS, UNBLOCKED RECORDS OF THE FOLLOWING LENGTHS WILL BE ASSUMED.

ANY CARD OR PUNCH CONFIGURATION	80 BYTES
ANY PRINTER CONFIGURATION	132 BYTES
DISK	180 BYTES
SPO	72 BYTES
ALL OTHERS	80 BYTES

SYNTAX: <REEL NUMBER PART> ::= REEL = <REEL NUMBER>

<REEL NUMBER> ::= <NUMBER>

FORMAT: REEL = NUMBER OF REEL

DEFAULT: THE FPB ASSUMES #1 IN THE ABSENCE OF ANY SPECIFICATION.

SYNTAX: <DISK FILE DESCRIPTION PART> ::= AREAS = <NUMBER OF AREAS> <SLASH> <PHYSICAL RECORDS PER AREA>

<NUMBER OF AREAS> ::= <NUMBER>

<PHYSICAL RECORDS PER AREA> ::= <NUMBER>

FORMAT: AREAS = # OF AREAS / #OF BLOCKS PER AREA

EXAMPLE: AREAS = 20 / 80

NOTE: <PHYSICAL RECORDS PER AREA> INDICATES THE NUMBER OF BLOCKS PER AREA. THIS ATTRIBUTE IS APPLICABLE FOR DISK FILES ONLY.

DEFAULT: IF AREAS ARE NOT SPECIFIED, THE FPB WILL ASSUME 25 AREAS WITH 100 BLOCKS PER AREA. IF THE RECORD SPECIFICATIONS HAVE BEEN GIVEN, THE COMPILER WILL COMPUTE THE NUMBER OF RECORDS PER AREA. HOWEVER, IF RECORD SPECIFICATIONS ARE OMITTED, THE FPB WILL ASSUME 100 RECORDS PER AREA. IN EITHER CASE THEN, WHETHER AREAS ARE SPECIFIED OR NOT, THE COMPILER WILL HAVE COMPUTED THE NUMBER OF RECORDS FOR INSERTION IN THE FPB.

SYNTAX: <PACK.ID PART> ::= PACK.ID = <PACK IDENTIFICATION>

<PACK IDENTIFICATION> ::= <CHARACTER STRING>

FORMAT: PACK.ID = "NAME"

EXAMPLE: PACK.ID = "TRANS.BAL"

NOTE: THE SYSTEM WILL USE ONLY THE FIRST TEN CHARACTERS OF THE "NAME".

DEFAULT: IF ABSENT, <PACK IDENTIFICATION> WILL BE SET TO BLANKS IN THE FPB.

SYNTAX: <OPEN OPTION>::= OPEN.OPTION= <OPEN OPTION ATTRIBUTE LIST>

<OPEN OPTION ATTRIBUTE LIST>::= <OPEN ATTRIBUTE>
/ <OPEN ATTRIBUTE> <SLASH> <OPEN OPTION ATTRIBUTE LIST>

<OPEN ATTRIBUTE> ::= SEE "OPEN STATEMENT"

FORMAT: OPEN.OPTION = ATTRIBUTE / ATTRIBUTE...

EXAMPLE: OPEN.OPTION = OUTPUT/NEW

NOTE: WHILE THE ATTRIBUTES ARE THE SAME, THE <OPEN ATTRIBUTE>S IN THE <OPEN STATEMENT> ARE SEPARATED BY COMMAS, AND THE <OPEN ATTRIBUTE>S IN THE <OPEN OPTION> ABOVE ARE SEPARATED BY SLASHES.

DEFAULT: IF ABSENT, THE <OPEN ATTRIBUTE>S WILL BE SET AS FOLLOWS:

IF <DEVICE> IS	<OPEN OPTION> IS
CARD	INPUT
PRINTER	OUTPUT
PUNCH	OUTPUT
DISK	INPUT

SYNTAX: <ALL.AREAS.AT.OPEN PART> ::= ALL.AREAS.AT.OPEN

FUNCTION: IF THIS OPTION IS SET, DISK SPACE FOR EACH AREA WILL BE ALLOCATED WHEN THE FILE IS OPENED. IF INSUFFICIENT SPACE IS AVAILABLE, A SPO MESSAGE WILL INDICATE THAT THERE IS NO USER DISK.

DEFAULT: AREAS ARE CREATED AS NEEDED.

SYNTAX: <AREA.BY.CYLINDER PART> ::= AREA.BY.CYLINDER

FUNCTION: IF THIS OPTION IS SPECIFIED, EACH AREA WILL BE PLACED AT THE BEGINNING OF A CYLINDER. IF THERE IS NO (MORE) SPACE AT THE BEGINNING OF ANY CYLINDER, A SPO MESSAGE WILL INDICATE THAT THERE IS NO USER DISK.

DEFAULT: AREAS ARE PLACED ANYWHERE ON DISK.

SYNTAX: <EU ASSIGNMENT PART> ::= EU.SPECIAL = <NUMBER>
/ EU.INCREMENTED = <NUMBER>

FUNCTION: THE <NUMBER> SPECIFIES ANY INTEGER 0 THROUGH 15. "EU.SPECIAL" IS APPLICABLE ONLY WITH HEAD PER TRACK DISKS AND SYSTEMS DISK PACKS, AND SPECIFIES THE DRIVE ON WHICH THE FILE IS TO GO. "EU.INCREMENTED" SPECIFIES THE DISK DRIVE ON WHICH THE FIRST AREA OF A FILE IS TO GO. EACH SUBSEQUENT AREA IS PLACED ON THE NEXT DRIVE. IF, WITH EITHER OPTION, THE NECESSARY E.U. IS NOT AVAILABLE, E.U. 0 WILL BE TAKEN.

DEFAULT: SPACE FOR FILES AND AREAS IS ALLOCATED ANYWHERE ON DISK.

SYNTAX: <MULTI PACK PART>::= MULTI.PACK

FUNCTION: IF THIS OPTION IS SPECIFIED, THE ENTIRE FILE MAY BE PUT ONTO SEVERAL DISK PACKS.

DEFAULT: THE FILE WILL BE PLACED ON ONE DISK PACK.

SYNTAX: <USE.INPUT.BLOCKING PART> ::= USE.INPUT.BLOCKING

FUNCTION: THIS OPTION IS ONLY APPLICABLE WITH INPUT DISK FILES. IF SPECIFIED, THE RECORD AND BLOCK SIZE SPECIFICATIONS WILL BE TAKEN FROM THE DISK FILE HEADER AND THE USER-S SPECIFICATIONS WILL BE IGNORED.

DEFAULT: THE FILE ATTRIBUTES ARE AS STATED IN THE FILE DECLARATION. THOSE OPTIONS OMITTED ARE SET TO THEIR DEFAULT STATUSES.

SYNTAX: <SORTER STATION PART> ::= SR.STATION = <NUMBER>

FUNCTION: THE NUMBER INDICATES WHICH READ STATION(S) IS(ARE) TO BE USED ON A SORTER-READER FILE. THE POSSIBLE STATIONS ARE THE MAGNETIC INK CHARACTER READER AND THE OPTICAL CHARACTER READER. THE READ STATIONS ARE INTERCHANGEABLE, THUS THE SYSTEM DOCUMENTATION SHOULD BE CONSULTED FOR SPECIFIC HARDWARE CONFIGURATIONS. THE VALUES ALLOWED ARE AS FOLLOWS:

1= FIRST STATION
2= SECOND STATION
3= BOTH STATIONS

DEFAULT: SR.STATION = 0

SYNTAX: <END.OF.PAGE PART> ::= END.OF.PAGE.ACTION

FUNCTION: THIS ATTRIBUTE WILL CAUSE THE <EOF PART> OF A <WRITE STATEMENT> TO BE EXECUTED AT THE END OF A PAGE ON A PRINTER FILE. REFER TO "WRITE STATEMENT" FOR DETAILS.

DEFAULT: NO AUTOMATIC PAGING ACTION

SYNTAX: <REMOTE.KEY PART>::= REMOTE.KEY

FUNCTION: THIS ATTRIBUTE IS USED ONLY WITH FILES OF TYPE "REMOTE". WHEN PRESENT, IT INDICATES THAT A KEY MAY BE PRESENT ON A READ OR WRITE TO THAT FILE. IF MISSING, THEN NO KEY CAN BE USED. THE FORMAT OF THE KEY IS GIVEN BELOW. EACH FIELD OF THE KEY IS IN DECIMAL CHARACTERS. THE KEY IS A TOTAL OF 10 CHARACTERS FORMATTED AS FOLLOWS:

STATION NUMBER	3 CHARACTERS
MESSAGE LENGTH (BYTE COUNT)	4 CHARACTERS
MESSAGE TYPE (MUST BE "000")	3 CHARACTERS

DEFAULT: NO REMOTE KEY

SYNTAX: <NUMBER.OF.STATIONS PART>:= NUMBER.OF.STATIONS = <NUMBER>

FUNCTION: THIS ATTRIBUTE IS USED ONLY WITH FILES OF TYPE "REMOTE". WHEN PRESENT, IT SPECIFIES THE MAXIMUM NUMBER OF STATIONS THAT CAN BE ATTACHED TO THIS FILE.

DEFAULT: NUMBER.OF.STATIONS=1

SYNTAX: <QUEUE.FAMILY.SIZE PART>::= QUEUE.FAMILY.SIZE=<NUMBER>

FUNCTION: THIS ATTRIBUTE IS USED ONLY WITH FILES OF TYPE "QUEUE". IT SPECIFIES THE NUMBER OF MEMBERS IN A QUEUE FAMILY.

DEFAULT: QUEUE.FAMILY.SIZE=1

SYNTAX: <FILE TYPE PART>::= FILE.TYPE=<FILE TYPE SPECIFIER>

<FILE TYPE SPECIFIER>::= DATA/INTERPRETER/CODE/INTRINSIC

FUNCTION: THIS ATTRIBUTE ALLOWS SDL PROGRAMS TO SPECIFY THE TYPE OF THE FILES THEY ARE CREATING. IN PARTICULAR, THE COMPILERS WILL USE THE TYPE "CODE" FOR THEIR CODEFILES.

DEFAULT: FILE TYPE PART=DATA

SYNTAX: <WORK FILE PART>::= WORK.FILE

FUNCTION: THIS ATTRIBUTE CAUSES THE JOB NUMBER TO BE INCLUDED AS PART OF THE FILE IDENTIFIER.

DEFAULT: NOT A WORK FILE

SYNTAX: <LABEL TYPE PART>::= LABEL.TYPE=<LABEL TYPE SPECIFIER>
<LABEL TYPE SPECIFIER>::= UNLABELED

FUNCTION: THIS ATTRIBUTE ALLOWS THE LABEL TYPE TO BE SPECIFIED. CURRENTLY, "UNLABELED" IS THE ONLY TYPE.

DEFAULT: BURROUGHS STANDARD LABEL

SWITCH FILE DECLARATIONS

```
<SWITCH FILE
DECLARATION STATEMENT>::=     SWITCH.FILE <SWITCH FILE
                              DECLARE ELEMENT LIST>

<SWITCH FILE
DECLARE ELEMENT LIST>::=        <SWITCH FILE DECLARE ELEMENT>
                              / <SWITCH FILE DECLARE ELEMENT>,
                                <SWITCH FILE DECLARE ELEMENT LIST>

<SWITCH FILE
DECLARE ELEMENT> ::=          <SWITCH FILE IDENTIFIER> (<FILE
                              IDENTIFIER LIST>)

<SWITCH FILE IDENTIFIER>::=   <IDENTIFIER>

<FILE IDENTIFIER LIST>::=     <FILE IDENTIFIER>
                            / <FILE IDENTIFIER>, <FILE IENTIFIER LIST>
```

A SWITCH FILE DECLARATION SPECIFIES THE ELEMENTS OF A "CASE", THESE ELEMENTS BEING FILES. A SUBSCRIPTED <SWITCH FILE IDENTIFIER> MAY BE USED ANYWHERE THAT A <FILE IDENTIFIER> MAY BE USED. IF THERE ARE N FILES IN THE <FILE IDENTIFIER LIST>, THEN THE SUBSCRIPT MUST RANGE FROM 0 TO N-1. THE VALUE OF THE SUBSCRIPT SELECTS ONE OF THE N FILES IN THE LIST, DEPENDING UPON ORDINAL POSITION (THE FILES IN THE <FILE IDENTIFIER LIST> ARE NUMBERED FROM LEFT TO RIGHT, BEGINING WITH 0). IF ALL FILES IN THE <FILE IDENTIFIER LIST> ARE OF TYPE "REMOTE", THEN THE SWITCH FILE IDENTIFIER IS OF TYPE "REMOTE".

THE FOLLOWING EXAMPLE COPIES CARD IMAGES FROM CARDS, TAPE, OR DISK TO CARDS, PRINTER, TAPE, OR DISK:

```
FILE
     CARDS(DEVICE=CARD)
    ,TAPEI(DEVICE=TAPE,USE.INPUT.BLOCKING)
    ,DISKI(DEVICE=DISK,USE.INPUT.BLOCKING)
    ;
FILE
     PUNCH(DEVICE=PUNCH)
    ,LINE(DEVICE=PRINTER)
    ,TAPEO(DEVICE=TAPE,RECORDS=80/4)
    ,DISKO(DEVICE=DISK,RECORDS=80/9)
    ;
SWITCH.FILE
     INPUT(CARDS,TAPEI,DISKI)
    ,OUTPUT(PUNCH,LINE,TAPEO,DISKO)
```

```
    ;
DECLARE
     INPUT.TYPE BIT(24)
    ,OUTPUT.TYPE BIT(24)
    ,BUFFER CHARACTER(80)
    ;
DISPLAY "***** INPUT TYPE";
ACCEPT INPUT.TYPE;
INPUT.TYPE←BINARY(SUBSTR(INPUT.TYPE,0,1)) MOD 3;
DISPLAY "***** OUTPUT TYPE";
ACCEPT OUTPUT.TYPE;
OUTPUT.TYPE←BINARY(SUBSTR(OUTPUT.TYPE,0,1)) MOD 4;
OPEN INPUT(INPUT.TYPE) INPUT;
OPEN OUTPUT(OUTPUT.TYPE) OUTPUT/NEW;
DO FOREVER;
    READ INPUT(INPUT.TYPE) (BUFFER);
         ON EOF UNDO;
    WRITE OUTPUT(OUTPUT.TYPE) (BUFFER);
END;
CLOSE OUTPUT(OUTPUT.TYPE) WITH LOCK;
STOP;
FINI
```

DEFINE STATEMENT

```
<DECLARATION STATEMENT> ::=  .../<DEFINE STATEMENT>;/...

<DEFINE STATEMENT> ::=         DEFINE <DEFINE ELEMENT>
                             / <DEFINE STATEMENT>,
                               <DEFINE ELEMENT>

<DEFINE ELEMENT> ::=           <DEFINE IDENTIFIER>
                               <FORMAL PARAMETER PART>
                               AS <DEFINE STRING>

<DEFINE IDENTIFIER> ::=        <IDENTIFIER>

<FORMAL PARAMETER PART> ::=  (<FORMAL PARAMETER LIST>)
                             / [<FORMAL PARAMETER LIST>]
                             / <EMPTY>

<FORMAL PARAMETER LIST> ::=  <FORMAL PARAMETER>
                             / <FORMAL PARAMETER>,
                               <FORMAL PARAMETER LIST>

<FORMAL PARAMETER> ::=         <IDENTIFIER>

<DEFINE STRING> ::=            #<WELL-FORMED CONSTRUCT>#

<WELL-FORMED CONSTRUCT> ::=  <EMPTY>
                             / <BASIC COMPONENT>
                               <WELL-FORMED CONSTRUCT>

<BASIC COMPONENT> ::=          <RESERVED WORD>          %SEE APPENDIX
                             / <IDENTIFIER>
                             / <SPECIAL CHARACTER>
                             / <COMMENT STRING>
                             / <CONSTANT>
```

THE <DEFINE STATEMENT> ASSIGNS THE TEXT ENCLOSED BETWEEN THE "#" SIGNS FOLLOWING THE RESERVED WORD "AS" TO THE <DEFINE IDENTIFIER>. INVOCATION OF THE <DEFINE IDENTIFIER> CAUSES THE TEXT TO REPLACE THE IDENTIFIER, THEREBY PROVIDING A FORM OF SHORTHAND CODE.

AT DECLARATION TIME, THE COMPILER IS UNCONCERNED WITH THE CONTENTS OF THE <DEFINE STRING>. HOWEVER, WHEN THE <DEFINE IDENTIFIER> IS INVOKED, THE <WELL-FORMED CONSTRUCT> MUST CONFORM TO THE SYNTACTICAL REQUIREMENTS OF THE STATEMENT CONTAINING THE IDENTIFIER.

THERE ARE TWO TYPES OF <DEFINE STATEMENT>S: SIMPLE AND PARAMETRIC, WHERE THE PARAMETERS ARE ENCLOSED IN PARENTHESES FOLLOWING THE <DEFINE IDENTIFIER>. BELOW ARE EXAMPLES OF BOTH TYPES:

```
DEFINE A AS #IF X>10 THEN PROCX#,
       CH AS #CHARACTER#,
       B(Y,Z) AS #IF Y<Z THEN Y:=Z #,
       C(M) AS # X:=M; A #;
```

NOTICE THAT <DEFINE STATEMENT>S MAY BE FACTORED, WITH COMMAS SEPARATING EACH ELEMENT.

THE <DEFINE IDENTIFIER> HAS SCOPE IN THE SAME MANNER AS ANY OTHER IDENTIFIER (EXCEPT FOR SEGMENT AND DO-GROUP IDENTIFIERS).

RESTRICTIONS ON THE USE OF DEFINES:

1. RESERVED WORDS MAY NOT BE USED AS <DEFINE IDENTIFIER>S, HOWEVER, AN IDENTIFIER MAY DEFINE A RESERVED WORD.

2. "SPECIAL" WORDS MAY BE USED AS <DEFINE IDENTIFIER>S, HOWEVER, THEIR SPECIAL SIGNIFICANCE IS LOST WITHIN THE THE SCOPE OF THAT <DEFINE STATEMENT>.

3. <DEFINE INVOCATION>S MAY APPEAR WITHIN A <WELL-FORMED CONSTRUCT>, I.E., A <DEFINE IDENTIFIER> MAY APPEAR WITHIN ANOTHER <DEFINE ELEMENT>. <DEFINE IDENTIFIER>S MAY BE NESTED NO MORE THAN 12 LEVELS DEEP.

4. THE IDENTIFIERS LISTED BELOW ARE NEVER LOOKED UP IN THE LIST OF DEFINE NAMES.

   DECLARE, DEFINE, PROCEDURE, AND FORMAL IDENTIFIERS,

   SEGMENT AND DO-GROUP IDENTIFIERS,

   FILE, OPEN, AND CLOSE ATTRIBUTES,

   <FILE ATTRIBUTE STATEMENT> ATTRIBUTE NAMES

   "ON" CONDITION NAMES (EOF, EXCEPTION, FILE.MISSING, Q.FULL, Q.EMPTY, NO.INPUT. FILE.LOCKED).

   "ACCEPT"/"DISPLAY" SPECIFIERS: END.OF.TEXT AND CRUNCHED.

IF ONE OF THESE IDENTIFIERS HAPPENS TO BE THE SAME AS A <DEFINE IDENTIFIER>, NO SUBSTITUTION OCCURS. THE <WELL-FORMED CONSTRUCT> OF THE DEFINE WILL NOT REPLACE THE IDENTIFIER. NOTE, HOWEVER, THAT DUPLICATE IDENTIFIERS MAY NOT APPEAR WITHIN THE SAME LEXIC LEVEL; AN ERROR MESSAGE RESULTS.

5. THERE MAY BE NO MORE THAN 8 <FORMAL PARAMETER>S IN A <FORMAL PARAMETER LIST>.

6. REFER TO APPENDIX VI FOR RULES CONCERNING CONDITIONAL INCLUSION CARDS WITHIN DEFINES.

THE FOLLOWING SYNTAX ILLUSTRATES THE FORMAT USED IN THE INVOCATION OF A <DEFINE IDENTIFIER>:

```
<DEFINE INVOCATION> ::=      <SIMPLE DEFINE IDENTIFIER>
                           / <PARAMETRIC DEFINE IDENTIFIER>
                             (<DEFINE ACTUAL PARAMETER LIST>)
                           / <PARAMETRIC DEFINE IDENTIFIER>
                             [<DEFINE ACTUAL PARAMETER LIST>]

<SIMPLE DEFINE
IDENTIFIER> ::=              <DEFINE IDENTIFIER>

<PARAMETRIC
DEFINE IDENTIFIER> ::=       <DEFINE IDENTIFIER>

<DEFINE ACTUAL
PARAMETER LIST> ::=          <DEFINE ACTUAL PARAMETER>
                           / <DEFINE ACTUAL PARAMETER>,
                             <DEFINE ACTUAL PARAMETER LIST>

<DEFINE ACTUAL
PARAMETER> ::=               <WELL-FORMED CONSTRUCT>
```

A <DEFINE INVOCATION> MAY OCCUR ANYWHERE WITHIN AN SDL PROGRAM EXCEPT IN THE CASES LISTED ABOVE IN RESTRICTION 4. AS INDICATED BY THE ABOVE BNF, THE ACTUAL PARAMETERS OF A DEFINE ARE NOT CONFINED TO CONSTANTS AND VARIABLES BUT MAY HAVE A WIDE RANGE OF CONSTRUCTS. FOR EXAMPLE, THE <DEFINE STATEMENT> MENTIONED ABOVE:

```
DEFINE C(M) AS# X:=M; A #;
```

MIGHT BE INVOKED AS FOLLOWS:

```
C(Z;BUMP I[R,S]);
```

WHICH EXPANDS TO:

```
X:=Z; BUMP I[R,S]; IF X>10 THEN PROCX;
```

THE FOLLOWING RESTRICTIONS APPLY TO THE USE OF THE <DEFINE INVOCATION>:

1. NO UNPAIRED BRACKETING SYMBOLS, I.E., () OR [], MAY APPEAR.

2. WITHIN A <DEFINE ACTUAL PARAMETER LIST>, COMMAS NOT ENCLOSED WITHIN PAIRED BRACKETING SYMBOLS ACT TO DELIMIT THE <DEFINE ACTUAL PARAMETER>S. THEREFORE A <WELL-FORMED CONSTRUCT> NOT ENCLOSED IN BRACKETING SYMBOLS MAY NOT CONTAIN COMMAS. FOR EXAMPLE:

```
DEFINE X(A,B) AS # A(B) #;
```

AND INVOKED AS:

```
Z:=X(M,Q,R,S);
```

WOULD RESULT IN THE ERROR MESSAGE:

DEFINE INVOCATION HAS TOO MANY PARAMETERS

PROPER INVOCATION IS POSSIBLE BY REMOVING THE PARENS FROM THE DEFINE AND PLACING THEM IN THE INVOCATION:

```
DEFINE X(A,B) AS # A B #;
Z:=X(M,(Q,R,S));
```

3. COMMENTS ARE ALLOWED BUT WILL BE DELETED FROM THE ACTUAL PARAMETER TEXT.

FORWARD DECLARATION

```
<DECLARATION STATEMENT> ::=   .../<FORWARD DECLARATION>/...

<FORWARD DECLARATION> ::=     FORWARD <COMPOUND PROCEDURE HEAD>

<COMPOUND PROCEDURE
HEAD> ::=                     <PROCEDURE HEAD>
                              <FORMAL PARAMETER DECLARATION
                              STATEMENT LIST>

<PROCEDURE HEAD> ::=          <BASIC PROCEDURE HEAD>
                              <PROCEDURE TYPE PART>;

<BASIC PROCEDURE HEAD> ::=    <PROCEDURE NAME>
                              <FORMAL PARAMETER PART>

<PROCEDURE NAME> ::=          PROCEDURE <PROCEDURE IDENTIFIER>

<PROCEDURE IDENTIFIER> ::=    <TYPED PROCEDURE IDENTIFIER>
                            / <NON-TYPED PROCEDURE IDENTIFIER>

<TYPED PROCEDURE
IDENTIFIER> ::=               <IDENTIFIER>

<NON-TYPED PROCEDURE
IDENTIFIER> ::=               <IDENTIFIER>

<FORMAL PARAMETER PART> ::=   <EMPTY>
                            / (<FORMAL PARAMETER LIST>)

<FORMAL PARAMETER LIST> ::=   <FORMAL PARAMETER>
                            / <FORMAL PARAMETER>,
                              <FORMAL PARAMETER LIST>

<FORMAL PARAMETER> ::=        <IDENTIFIER>

<PROCEDURE TYPE PART> ::=     <EMPTY>
                            / <FORMAL TYPE PART>

<FORMAL TYPE PART> ::=        <TYPE PART>
                            / <TYPE VARYING PART>

<TYPE PART> ::=               FIXED
                            / CHARACTER <FIELD SIZE>
                            / BIT <FIELD SIZE>

<TYPE VARYING PART> ::=       VARYING
                            / BIT VARYING
                            / CHARACTER VARYING
```

```
<FORMAL PARAMETER DECLA-
RATION STATEMENT LIST> ::=    <EMPTY>
                            / <FORMAL PARAMETER DECLARATION STATEMENT>;
                              <FORMAL PARAMETER DECLARATION
                              STATEMENT LIST>

<FORMAL PARAMETER
DECLARATION STATEMENT> ::=    FORMAL <FORMAL ELEMENT>
                            / FORMAL.VALUE <FORMAL ELEMENT>
                            / <FORMAL PARAMETER DECLARATION STATEMENT>,
                              <FORMAL ELEMENT>

<FORMAL ELEMENT> ::=          (<FORMAL IDENTIFIER LIST>)
                              <FORMAL TYPE PART>
                            / <FORMAL IDENTIFIER>
                              <FORMAL TYPE PART>

<FORMAL INDENTIFIER
LIST> ::=                     <FORMAL IDENTIFIER>
                            / <FORMAL IDENTIFIER LIST>,
                              <FORMAL IDENTIFIER>

<FORMAL IDENTIFIER> ::=       <COMPLEX IDENTIFIER>
                            / <VARYING ARRAY SPECIFIER>

<COMPLEX IDENTIFIER> ::=      <SIMPLE IDENTIFIER>
                            / <ARRAY IDENTIFIER>
                              <ARRAY BOUND>

<VARYING ARRAY
SPECIFIER> ::=                <ARRAY IDENTIFIER>
                              <VARYING ARRAY BOUND>

<VARYING ARRAY BOUND> ::=     (*)
```

BEFORE A PROCEDURE MAY BE CALLED, SDL SPECIFIES THAT IT MUST HAVE BEEN PREVIOUSLY DECLARED. A CONTRADICTION ARISES WHEN ONE PROCEDURE CALLS ANOTHER PROCEDURE WHICH IN TURN REFERENCES THE FIRST. IN THIS CASE, WHICHEVER PROCEDURE APPEARS FIRST MUST NECESSARILY CONTAIN AT LEAST ONE REFERENCE TO THE SECOND WHICH HAS NOT YET BEEN DECLARED.

THE <FORWARD DECLARATION> ALLOWS THE PROGRAMMER TO USE RECURSIVE REFERENCES BY PROVIDING A TEMPORARY PROCEDURE DECLARATION. THE <FORWARD DECLARATION>, HOWEVER, DOES NOT ELIMINATE THE NEED FOR THE NORMAL PROCEDURE DECLARATION WHICH MUST FOLLOW IN THE PROGRAM AND MUST HAVE THE SAME SCOPE.

THE PARAMETERS MENTIONED IN THE <FORWARD DECLARATION> MUST BE THE SAME FORMAL PARAMETERS (IN TYPE AND SIZE, BUT NOT IN NAME)

THAT THE PROCEDURE ITSELF WILL DECLARE.

PROCEDURES MAY BE EITHER TYPED OR NON-TYPED DEPENDING ON THEIR USE. FORMAL DATA TYPES MAY EITHER BE STATIC OR VARYING, AGAIN DEPENDING ON THE PROGRAM. THESE SPECIFICATIONS WILL BE DISCUSSED IN THE SECTION ENTITLED "THE PROCEDURE STATEMENT".

THE FOLLOWING EXAMPLES ILLUSTRATE THE USE OF THE <FORWARD DECLARATION>:

```
FORWARD PROCEDURE X CHARACTER VARYING;
FORWARD PROCEDURE J(K,L,M);
        FORMAL K(*) BIT VARYING,
        FORMAL L(15) CHARACTER (8),
        FORMAL M FIXED;
```

USE STATEMENT

```
<USE STATEMENT> ::=          USE (<SIMPLE IDENTIFIER LIST>)
                             OF <DEFINE IDENTIFIER>

<SIMPLE IDENTIFIER
LIST> ::=                    <SIMPLE IDENTIFIER>
                           / <SIMPLE IDENTIFIER>,
                             <SIMPLE IDENTIFIER LIST>

<SIMPLE IDENTIFIER> ::=      <IDENTIFIER>

<DEFINE IDENTIFIER> ::=      <IDENTIFIER>
```

THE PURPOSE OF THE <USE STATEMENT> IS TO ALLOW THE PROGRAMMER TO DECLARE SPECIFIC ELEMENTS IN A DEFINED STRUCTURE WITHIN A PROCEDURE. BY SPECIFYING ONLY THE DESIRED ELEMENTS, THE NAME STACK SIZE IS KEPT TO A MINIMUM, AND PROGRAM MAINTENANCE IS SIMPLIFIED. THE COMPILER WILL GENERATE THE STRUCTURE USING FILLERS AND THE SPECIFIED ELEMENTS.

THE FOLLOWING RESTRICTIONS APPLY TO THE <USE STATEMENT>:

1. IT MUST APPEAR WITHIN A PROCEDURE (I.E., ON A LEXIC LEVEL GREATER THAN 0).

2. THE REFERENCED <DEFINE IDENTIFIER> MUST DEFINE ONE STRUCTURED DECLARE STATEMENT.

3. THE STRUCTURE MAY NOT CONTAIN ARRAYS.

4. THE OUTERMOST LEVEL OF THE STRUCTURE (01) MUST BE A "DUMMY REMAPS".

EXAMPLE:

```
DEFINE X AS #
DECLARE 01 DUMMY REMAPS A, % MIGHT ALSO REMAP BASE
           02 B    BIT(5),
             03 B1 BIT(2),
             03 B2 BIT(3),
           02 C    CHARACTER(10),
           02 D    BIT(1),
           02 E    FIXED,
           02 F    BIT(24)#;
PROCEDURE FIRST;
     USE (C,D) OF X;
```

FROM THE ABOVE <USE STATEMENT> THE COMPILER WILL GENERATE THE FOLLOWING STRUCTURE:

```
01  DUMMY REMAPS A,
    02  FILLER   BIT(5),
       03 FILLER BIT(2),
       03 FILLER BIT(3),
    02  C        CHARACTER(10),
    02  D        BIT(1),
    02  FILLER   FIXED,
    02  FILLER   BIT(24);
```

NOTE THAT FILLER WAS SUBSTITUTED FOR THE GROUP ITEM B. THIS WOULD NORMALLY GENERATE A SYNTAX ERROR, AND IS ALLOWABLE ONLY IN THE <USE STATEMENT>.

PROCEDURE STATEMENT

```
<PROCEDURE STATEMENT
LIST> ::=                       <EMPTY>
                              / <PROCEDURE STATEMENT>;
                                <PROCEDURE STATEMENT LIST>

<PROCEDURE STATEMENT> ::=       <PROCEDURE DEFINITION>
                              / <SEGMENT STATEMENT>
                                <PROCEDURE STATEMENT>

<PROCEDURE DEFINITION> ::=      <COMPOUND PROCEDURE HEAD>
                                <PROCEDURE BODY>

<SEGMENT STATEMENT>             SEE "THE SEGMENT STATEMENT"

<PROCEDURE BODY> ::=            <DECLARATION STATEMENT LIST>
                                <PROCEDURE STATEMENT LIST>
                                <PROCEDURE EXECUTABLE STATEMENT LIST>
                                <PROCEDURE ENDING>
```

PROCEDURES ARE SELF-CONTAINED FUNCTIONAL UNITS WITHIN AN SDL PROGRAM WHICH MAY BE ACCESSED ACCORDING TO SPECIFIC RULES DISCUSSED UNDER "BASIC STRUCTURE OF THE SDL PROGRAM". PROCEDURES MAY BE CREATED BY PRECEDING SELF-CONTAINED STATEMENTS WITH A <COMPOUND PROCEDURE HEAD>, AND TERMINATING IT WITH A <PROCEDURE ENDING>.

THE <PROCEDURE DEFINITION> IS COMPOSED OF THREE BASIC PARTS: THE HEADING, BODY, AND ENDING. IDENTIFIERS DECLARED IN A PROCEDURE MAY BE ACCESSED ONLY IN THE PROCEDURE IN WHICH THEY ARE DECLARED, AND IN PROCEDURES NESTED WITHIN THE DECLARING PROCEDURE.

PROCEDURES MAY BE EITHER "TYPED" OR "NON-TYPED". A "TYPED" PROCEDURE RETURNS SOME VALUE OF THE TYPE SPECIFIED IN THE PROCEDURE DECLARATION TO THE EXPRESSION WHERE THE PROCEDURE WAS INVOKED. SEE "VALUE VARIABLES" FOR DETAILS. A "NON-TYPED" PROCEDURE PERFORMS A FUNCTION, DOES NOT RETURN A VALUE, AND IS INVOKED IN AN <EXECUTE PROCEDURE STATEMENT>. SEE "EXECUTE PROCEDURE STATEMENT".

THE SYNTAX FOR THE PROCEDURE HEADING IS:

```
<COMPOUND PROCEDURE
HEAD> ::=                       <PROCEDURE HEAD>
```

```
                                <FORMAL PARAMETER DECLARATION
                                STATEMENT LIST>

<PROCEDURE HEAD> ::=            <BASIC PROCEDURE HEAD>
                                <PROCEDURE TYPE PART>;

<BASIC PROCEDURE HEAD> ::=      <PROCEDURE NAME>
                                <FORMAL PARAMETER PART>

<PROCEDURE NAME> ::=              PROCEDURE <PROCEDURE IDENTIFIER>
                                / INTRINSIC <INTRINSIC IDENTIFIER>

<PROCEDURE IDENTIFIER> ::=      <TYPED PROCEDURE IDENTIFIER>
                                / <NON-TYPED PROCEDURE IDENTIFIER>

<TYPED PROCEDURE
IDENTIFIER> ::=                 <IDENTIFIER>

<NON-TYPED PROCEDURE
IDENTIFIER> ::=                 <IDENTIFIER>

<INTRINSIC IDENTIFIER>:=        <TYPED INTRINSIC IDENTIFIER>
                                / <NON-TYPED INTRINSIC IDENTIFIER>

<TYPED INTRINSIC
IDENTIFIER>::=                  <IDENTIFIER>

<NON-TYPED INTRINSIC
IDENTIFIER>::=                  <IDENTIFIER>

<FORMAL PARAMETER PART> ::=     <EMPTY>
                                / (<FORMAL PARAMETER LIST>)

<FORMAL PARAMETER LIST> ::=     <FORMAL PARAMETER>
                                / <FORMAL PARAMETER>,
                                  <FORMAL PARAMETER LIST>

<FORMAL PARAMETER> ::=          <IDENTIFIER>

<PROCEDURE TYPE PART> ::=       <EMPTY>
                                / <FORMAL TYPE PART>

<FORMAL TYPE PART> ::=          <TYPE PART>
                                / <TYPE VARYING PART>

<TYPE PART> ::=                   FIXED
                                / CHARACTER <FIELD SIZE>
                                / BIT <FIELD SIZE>

<TYPE VARYING PART> ::=           VARYING
                                / BIT VARYING
                                / CHARACTER VARYING

<FORMAL PARAMETER DECLA-
```

```
RATION STATEMENT LIST> ::=    <EMPTY>
                            / <FORMAL PARAMETER DECLARATION STATEMENT>;
                              <FORMAL PARAMETER DECLARATION
                              STATEMENT LIST>

<FORMAL PARAMETER
DECLARATION STATEMENT> ::=    FORMAL <FORMAL ELEMENT>
                            / FORMAL.VALUE <FORMAL ELEMENT>
                            / <FORMAL PARAMETER DECLARATION STATEMENT>,
                              <FORMAL ELEMENT>

<FORMAL ELEMENT> ::=          (<FORMAL IDENTIFIER LIST>)
                              <FORMAL TYPE PART>
                            / <FORMAL IDENTIFIER>
                              <FORMAL TYPE PART>

<FORMAL INDENTIFIER
LIST> ::=                     <FORMAL IDENTIFIER>
                            / <FORMAL IDENTIFIER LIST>,
                              <FORMAL IDENTIFIER>

<FORMAL IDENTIFIER> ::=       <COMPLEX IDENTIFIER>
                            / <VARYING ARRAY SPECIFIER>

<COMPLEX IDENTIFIER> ::=      <SIMPLE IDENTIFIER>
                            / <ARRAY IDENTIFIER>
                              <ARRAY BOUND>

<VARYING ARRAY
SPECIFIER> ::=                <ARRAY IDENTIFIER>
                              <VARYING ARRAY BOUND>

<VARYING ARRAY BOUND> ::=     (*)
```

THE PROCEDURE HEADING, I.E., <COMPOUND PROCEDURE HEAD>, CONTAINS THE <PROCEDURE NAME>, FORMAL PARAMETERS (IF ANY), AND THE <PROCEDURE TYPE PART>, I.E., THE FIELD TYPE OF THE VALUE TO BE RETURNED IF THE PROCEDURE IS "TYPED". FOR EXAMPLE:

```
PROCEDURE X (M,N) FIXED;
   FORMAL (M,N) VARYING;
```

WHICH CORRESPONDS TO THE FOLLOWING SYNTAX:

```
PROCEDURE <TYPED PROCEDURE IDENTIFIER>
          (<FORMAL PARAMETER>,<FORMAL PARAMETER>)
          <PROCEDURE TYPE PART>;
   FORMAL (<FORMAL IDENTIFIER>,<FORMAL IDENTIFIER>)
          <FORMAL TYPE PART>;
```

IN THIS CASE, THE VALUE RETURNED TO THE POINT OF INVOCATION SHOULD BE FIXED. THERE IS, HOWEVER, NO CHECK FOR THIS AT COMPILE TIME. IF THE CONTROL CARD OPTION "FORMALCHECK" IS PRESENT, THE RETURNED VALUES WILL BE CHECKED AGAINST THE PROCEDURE TYPE AT RUN TIME.

THE "NON-TYPED" PROCEDURE FOLLOWS THE SAME FORMAT EXCEPT THAT THE <PROCEDURE TYPE PART> IS OMITTED SINCE NO VALUE IS RETURNED. FOR INSTANCE:

```
PROCEDURE A (J,K,L);
   FORMAL J FIXED, (K,L) BIT VARYING;
```

WHICH SYNTACTICALLY IS THE SAME AS:

```
PROCEDURE <NON-TYPED PROCEDURE IDENTIFIER>
          (<FORMAL PARAMETER>,<FORMAL PARAMETER>,
          <FORMAL PARAMETER>);
   FORMAL <FORMAL IDENTIFIER> <FORMAL TYPE PART>,
          (<FORMAL IDENTIFIER>,<FORMAL IDENTIFIER>)
          <FORMAL TYPE PART>;
```

THE FIELD TYPE OF FORMAL PARAMETERS (I.E., COMPONENTS OF THE <FORMAL TYPE PART>) MAY BE STATIC (BIT, CHARACTER, OR FIXED) OR VARIABLE (BIT VARYING, CHARACTER VARYING, OR VARYING).

OFTEN HOWEVER, IT IS IMPOSSIBLE TO DETERMINE THE DATA TYPE AT COMPILE TIME ESPECIALLY IF THE ACTUAL PARAMETERS ARE PASSED TO THE PROCEDURE FROM DIFFERENT POINTS IN THE PROGRAM AND UNDER DIFFERING CIRCUMSTANCES. SDL ALLOWS THE USER TO SPECIFY VARIABLE DATA FIELDS IN THE FORMAL DECLARATION. THE ACTUAL PARAMETERS PASSED TO THAT PROCEDURE WILL PROVIDE THE SPECIFICS. THUS FORMALS MAY BE DECLARED AS "BIT VARYING", "CHARACTER VARYING", OR "VARYING".

IN A VARIABLE BIT OR CHARACTER FIELD, THE TYPE OF DATA PASSED MUST BE THAT WHICH IS SPECIFIED (I.E., BIT OR CHARACTER). THE LENGTH, HOWEVER, REMAINS VARIABLE. FORMALS SPECIFIED AS "VARYING" MAY ACCEPT ANY TYPE OF DATA OF ANY LENGTH.

THE DATA TYPES OF CORRESPONDING FORMAL AND ACTUAL PARAMETERS WILL NOT BE CHECKED AT COMPILE TIME AND ONLY AT RUN TIME WHEN "FORMALCHECK" HAS BEEN SPECIFIED AS A CONTROL CARD OPTION.

VARYING FORMALS MAY BE REMAPPED, BUT IT IS THE PROGRAMMER-S RESPONSIBILITY TO ENSURE THAT THE REMAPPED FORMAL PARAMETER AND ITS CORRESPONDING ACTUAL PARAMETER MATCH. A WARNING MESSAGE

WILL APPEAR IN THE SOURCE LISTING WHERE THE REMAPPING HAS OCCURRED.

SDL ALSO ALLOWS FORMALLY DECLARED ARRAYS TO HAVE A VARIABLE NUMBER OF ELEMENTS BY SUBSTITUTING "*" FOR THE NUMBER FOLLOWING THE <ARRAY IDENTIFIER>. FOR INSTANCE:

```
PROCEDURE X (A,B);
   FORMAL A (*) FIXED, B (*) VARYING;
```

INTRINSICS

THE WORD "INTRINSIC" MAY BE USED INTERCHANGEABLY WITH THE WORD "PROCEDURE". IT IS, HOWEVER, INTENDED ONLY FOR USE BY THE SDL GROUP IN ORDER TO PROVIDE SDL INTRINSICS.

THE USE OF "INTRINSIC" FORCES THE INTRINSIC TO HAVE AS ENTRY POINT THE DISPLACEMENT 0 WITHIN A NEW SEGMENT.

***

THE BODY OF THE PROCEDURE FOLLOWS THE HEADING. INCLUDED ARE DECLARATION OF LOCAL DATA (DISCUSSED UNDER "THE DECLARATION STATEMENT"), NESTED PROCEDURES (ALSO SEE "BASIC STRUCTURE OF THE SDL PROGRAM"), EXECUTABLE STATEMENTS, AND AN ENDING. THE SYNTAX FOR THE <PROCEDURE EXECUTABLE STATEMENT LIST> FOLLOWS:

```
<PROCEDURE BODY> ::=          <DECLARATION STATEMENT LIST>
                              <PROCEDURE STATEMENT LIST>
                              <PROCEDURE EXECUTABLE STATEMENT LIST>
                              <PROCEDURE ENDING>

<PROCEDURE EXECUTABLE
STATEMENT LIST> ::=             <PROCEDURE EXECUTABLE STATEMENT>
                              / <PROCEDURE EXECUTABLE STATEMENT>
                                <PROCEDURE EXECUTABLE STATEMENT LIST>

<PROCEDURE EXECUTABLE
STATEMENT> ::=                  <EXECUTABLE STATEMENT>
                              / <RETURN STATEMENT>
                              / <SEGMENT STATEMENT>
                                <PROCEDURE EXECUTABLE STATEMENT>
```

THE <EXECUTABLE STATEMENT>S WILL BE DISCUSSED IN THE SECTION ENTITLED "EXECUTABLE STATEMENTS". AS INDICATED BY THE ABOVE SYNTAX, EXECUTABLE STATEMENTS WITHIN A PROCEDURE MAY BE SEGMENTED. HOWEVER, A PROCEDURE MUST END IN THE SAME SEGMENT IN WHICH IT BEGINS. FOR OTHER SEGMENTATION RESTRICTIONS SEE "THE SEGMENT STATEMENT".

THE SYNTAX FOR THE <RETURN STATEMENT> IS:

```
<RETURN STATEMENT> ::=        <TYPED PROCEDURE RETURN STATEMENT>
```

```
                                / <NON-TYPED PROCEDURE RETURN STATEMENT>

<TYPED PROCEDURE
RETURN STATEMENT> ::=             RETURN <EXPRESSION>

<NON-TYPED PROCEDURE
RETURN STATEMENT> ::=             RETURN
                                / RETURN.AND.ENABLE.INTERRUPTS
```

THE <RETURN STATEMENT> TAKES ONE OF TWO FORMS DEPENDING ON THE TYPE OF THE PROCEDURE ENCOMPASSING IT. IF THE PROCEDURE IS "TYPED", AN <EXPRESSION> MUST BE RETURNED TO THE POINT OF INVOCATION. IN A "NON-TYPED" PROCEDURE, ONLY A SIMPLE RETURN IS NEEDED. FOR EXPRESSION SPECIFICATIONS REFER TO THE SECTIONS ENTITLED "EXPRESSIONS" AND "PRIMARIES".

TYPE CHECKING ON A <RETURN STATEMENT> IS DONE ONLY AT RUN TIME WHEN "FORMALCHECK" APPEARS AS A CONTROL CARD OPTION.

WITHIN ANY GIVEN PROCEDURE (AT ANY LEXIC LEVEL), CERTAIN STATEMENTS ARE NESTED WITHIN OTHER STATEMENTS AND ARE ACCESSED, MUCH LIKE A PROCEDURE, BY AN ADDRESS GENERATED BY THE LARGER STATEMENT. THE MOST GENERAL NESTING LEVEL IS ZERO, AND THE NESTING LEVEL OF ANY STATEMENT APPEARS ON AN SDL LISTING UNDER THE COLUMN "NL". THE MOST COMMON INSTANCE OF STATEMENTS OCCURRING AT NESTING LEVEL 1 OR GREATER ARE:

1. THE CONDITIONALLY EXECUTED STATEMENTS FOLLOWING "THEN" AND "ELSE" IN THE <IF STATEMENT>.

2. STATEMENTS CONTAINED WITHIN A <CASE STATEMENT>.

3. <DO-GROUP>S.

IF THE COMPILER CANNOT FIND A <RETURN STATEMENT> ON NL 0, IT WILL GENERATE ONE DIRECTLY PRECEDING THE <PROCEDURE ENDING>. THIS IS MERELY A SAFETY MEASURE TO INSURE THAT A PROCEDURE CAN ALWAYS BE PROPERLY EXITED.

A COMPILER-GENERATED RETURN WORKS ESSENTIALLY IN THE SAME MANNER AS AN EXPLICIT RETURN. IN A NON-TYPED PROCEDURE, CONTROL IS RETURNED TO THE POINT OF THE PROCEDURE-S INVOCATION. IN A TYPED PROCEDURE, THE FOLLOWING VALUES ARE RETURNED.

IF THE PROCEDURE IS TYPED:	THE COMPILER WILL RETURN:
BIT	BITS CONTAINING 0 OF LENGTH SPECIFIED
CHARACTER	BLANKS OF LENGTH SPECIFIED
FIXED	FIXED ZERO
BIT VARYING	8-BITS OF ZERO
CHARACTER VARYING	ONE BLANK
VARYING	FIXED ZERO

RETURN.AND.ENABLE.INTERRUPTS IS FOR MCP USE ONLY. IT WILL CAUSE A NORMAL PROCEDURE EXIT TO TAKE PLACE, AND WILL ENABLE INTERRUPTS AS WELL.

* * *

THE <PROCEDURE ENDING> IS THE FINAL STATEMENT OF A PROCEDURE, AND THE SYNTAX IS:

```
<PROCEDURE ENDING> ::=      END
                          / END <PROCEDURE IDENTIFIER>
```

THE IDENTIFIER FOLLOWING THE RESERVED WORD "END" IS OPTIONAL. ITS SOLE PURPOSE IS TO SIMPLIFY THE DOCUMENTATION OF THE PROGRAM. IF AN IDENTIFIER IS SUPPLIED BY THE USER, THE COMPILER WILL PERFORM A SYNTAX CHECK TO GUARANTEE THAT THE <PROCEDURE ENDING> IS APPROPRIATELY PLACED.

ASSIGNMENT STATEMENTS AND EXPRESSIONS

```
<ASSIGNMENT STATEMENT> ::=    <ADDRESS VARIABLE>
                              <REPLACE>
                              <EXPRESSION>

<ADDRESS VARIABLE> ::=        SEE "ADDRESS VARIABLES"

<REPLACE> ::=                 ←/ :=

<EXPRESSION LIST> ::=           <EXPRESSION>
                              / <EXPRESSION>,
                                <EXPRESSION LIST>

<EXPRESSION> ::=                <STRING EXPRESSION>
                              / <STRING EXPRESSION>
                                CAT <EXPRESSION>

<STRING EXPRESSION> ::=         <LOGICAL FACTOR>
                              / <LOGICAL FACTOR>
                                <OR-ING OPERATOR>
                                <STRING EXPRESSION>

<OR-ING OPERATOR> ::=           OR  /  EXOR

<LOGICAL FACTOR> ::=            <LOGICAL SECONDARY>
                              / <LOGICAL SECONDARY>
                                AND <LOGICAL FACTOR>

<LOGICAL SECONDARY> ::=         <LOGICAL PRIMARY>
                              / NOT <LOGICAL PRIMARY>

<LOGICAL PRIMARY> ::=           <ARITHMETIC EXPRESSION>
                              / <ARITHMETIC EXPRESSION>
                                <RELATION>
                                <ARITHMETIC EXPRESSION>

<RELATION> ::=                  <  /  ≤  /  =  /  ≠  /  ≥  /  >  /
                                LSS  /  LEQ  /  EQL  /  NEQ  /
                                GEQ  /  GTR

<ARITHMETIC
EXPRESSION> ::=                 <TERM>
                              / <TERM>
                                <ADDITIVE OPERATOR>
                                <ARITHMETIC EXPRESSION>

<ADDITIVE OPERATOR> ::=         +  /  -
```

```
<TERM>::=                       <SIGNED PRIMARY>
                              / <SIGNED PRIMARY>
                                <MULTIPLICATIVE OPERATOR>
                                <TERM>

<MULTIPLICATIVE
OPERATOR> ::=                   *  /  MOD  /  <SLASH>

<SIGNED PRIMARY>::=             <PRIMARY>
                              / <UNARY OPERATOR>
                                <PRIMARY>

<UNARY OPERATOR> ::=            + / -
```

THE ALGORITHM WHICH COMPILES AN SDL EXPRESSION MAY BE BEST UNDERSTOOD IN TERMS OF POLISH POST-FIX NOTATION. POLISH NOTATION IS AN ARITHMETICAL OR LOGICAL SYSTEM USING ONLY OPERANDS AND OPERATORS ARRANGED IN A SEQUENCE WHICH ELIMINATES THE NECESSITY OF PRIMARY BOUNDARIES (I.E., PARENTHESES).

IN POLISH, OPERANDS ARE EMITTED IN THE SAME LEFT TO RIGHT ORDER THAT THEY APPEAR IN THE EXPRESSION. OPERATORS ARE EMITTED ACCORDING TO THE RULES OF OPERATOR PRECEDENCE DEFINED BY SDL.

NOTE THAT THE END RESULTS OF THE EVALUATION OF AN SDL EXPRESSION AND ITS POLISH EQUIVALENT WILL ALWAYS BE THE SAME. HOWEVER, FOR OPTIMUM USE OF THE EVALUATION STACK, THE COMPILER MAY NOT LOAD THE OPERANDS IN THE EXACT ORDER INDICATED BY THE POLISH STRING.

THE PRECEDENCE OF ANY OPERATOR IS DETERMINED BY COMPARING IT WITH THE FIRST OPERATOR TO ITS LEFT. FIGURE 3 SHOWS THE PRECEDENCE RELATIONSHIP BETWEEN ANY TWO OPERATORS WHICH MAY APPEAR IN AN SDL EXPRESSION.

THE FOLLOWING ALGORITHM IS USED TO CONVERT AN SDL EXPRESSION INTO A POLISH STRING, AND REPRESENTS THE LOGIC BY WHICH THE COMPILER TRANSLATES AN EXPRESSION. NOTE THAT THIS IS FUNCTIONALLY WHAT THE COMPILER DOES, NOT WHAT IT ACTUALLY DOES.

1. GET TOKEN FROM EXPRESSION.

2. IF TOKEN = OPERAND, THEN PLACE IN POLISH STRING AND GO TO 1.

3. IF TOKEN (IE., PRESENT OP) = RIGHT PAREN, THEN IF [TOKEN TOP OF STACK] (IE., PREVIOUS OP) =

LEFT PAREN, THEN POP STACK AND GO TO 1.

4. IF TOKEN = ET AND [TOP OF STACK] = BT, THEN EXIT.

5. IF PRECEDENCE [TOP OF STACK] < PRECEDENCE [TOKEN], THEN PUT TOKEN ON TOP OF STACK AND GO TO 1.

6. IF PRECEDENCE [TOP OF STACK] > PRECEDENCE [TOKEN], THEN PUT [TOKEN TOP OF STACK] IN POLISH AND GO TO 3.

THE FOLLOWING IS A LIST OF THE SDL OPERATORS FROM HIGHEST PRECEDENCE TO LOWEST. THIS LIST OR THE TABLE IN FIGURE 3 MAY BE USED WHEN EVALUATING AN EXPRESSION.

+, - (UNARY)
*, /, MOD
+, - (BINARY)
<, ≤, =, ≠, ≥, >
NOT
AND
OR, EXOR
CAT

1. THE ASSIGNMENT OPERATOR HAS HIGHER PRECEDENCE THAN ANY OPERATOR TO ITS LEFT AND LOWER PRECEDENCE THAN ANY TO ITS RIGHT.

2. THE ORDER OF EVALUATION OF OPERATORS HAVING EQUAL PRECEDENCE IS ALWAYS FROM LEFT TO RIGHT.

PREVIOUS OP. \ PRESENT OP.	NEG	*	+ -	=	NOT	AND	OR	CAT	:=	(	)	ET
NEG	>	>	>	>	<	>	>	>	<	<	>	>
*	<	>	>	>	<	>	>	>	<	<	>	>
+ -	<	<	>	>	<	>	>	>	<	<	>	>
=	<	<	<	>	<	>	>	>	<	<	>	>
NOT	<	<	<	<	>	>	>	>	<	<	>	>
AND	<	<	<	<	<	>	>	>	<	<	>	>
OR	<	<	<	<	<	<	>	>	<	<	>	>
CAT	<	<	<	<	<	<	<	>	<	<	>	>
:=	<	<	<	<	<	<	<	<	<	<	>	>
(	<	<	<	<	<	<	<	<	<	<	=	
)		>	>	>		>	>	>	>		>	>
BT	<	<	<	<	<	<	<	<	<	<		=

FORMULA: PRECEDENCE <PREVIOUS OP> <RELATION> PRECEDENCE <PRESENT OP>

NOTE:

NEG	UNARY OPERATORS
*	MULTIPLICATIVE OPERATORS
=	RELATIONAL OPERATORS
:=	REPLACE OPERATORS
BT	INFERRED BEGINNING TERMINATOR
ET	INFERRED ENDING TERMINATOR

FIG 3. OPERATOR PRECEDENCE TABLE

UNARY OPERATOR

+
-

THE UNARY OPERATOR ACTS UPON ONE OPERAND AND MAY NEVER APPEAR AS AN INFIX OPERATOR BETWEEN TWO OPERANDS. IT MAY APPEAR TO THE RIGHT OF ANY OTHER OPERATOR, INCLUDING ITSELF.

THE UNARY MINUS (-) GENERATES THE TWO-S COMPLEMENT OF THE OPERAND ASSOCIATED WITH IT (I.E., -X = (NOT X)+1). THE OPERAND MAY BE ANY DATA TYPE. IF IT IS FIXED, THE UNARY MINUS HAS THE EFFECT OF REVERSING THE SIGN, AND THE RESULT IS LABELED ON THE EVALUATION STACK AS FIXED.

IF THE OPERAND IS EITHER A CHARACTER OR BIT STRING, ONLY THE LOW-ORDER 24 BITS WILL BE EVALUATED. STRINGS LESS THAN 24 BITS WILL BE PADDED WITH LEADING ZEROES TO 24 BITS. THE TWO-S COMPLEMENT OF THE STRING IS GENERATED AND RETURNED TO THE STACK AS TYPE BIT. NOTE, HOWEVER, THAT THE NEGATION OF ANY BIT OR CHARACTER STRING CAN NEVER RESULT IN A VALUE LESS THAN ZERO.

THE SDL COMPILER GENERATES NO CODE FOR THE UNARY PLUS (+) WHICH EXISTS SOLELY FOR THE CONVENIENCE OF THE PROGRAMMER.

ARITHMETIC OPERATORS

+	ADDITION
-	SUBTRACTION
*	MULTIPLICATION
MOD	DIVISION YIELDING INTEGER VALUE OF REMAINDER
/	DIVISION YIELDING INTEGER VALUE OF QUOTIENT

THE ARITHMETIC OPERATORS PERFORM 24-BIT ARITHMETIC ON TWO OPERANDS OF ANY OF THE THREE DATA TYPES. SIGN ANALYSIS WILL BE DONE ONLY IF BOTH OPERANDS ARE FIXED. WITH ANY OTHER COMBINATION OF DATA TYPES, THE MAGNITUDES OF THE OPERANDS ARE EVALUATED.

FOR BOTH BIT AND CHARACTER DATA, IF THE FIELD IS GREATER THAN 24 BITS, ONLY THE LOW-ORDER 24 BITS WILL BE EVALUATED. IF THE FIELD IS LESS THAN 24 BITS, LEADING ZEROES WILL BE SUPPLIED FROM THE LEFT.

A 24-BIT RESULT WILL BE RETURNED TO THE EVALUATION STACK. IF BOTH OPERANDS ARE FIXED, THE RESULT WILL BE FIXED. OTHERWISE, THE RESULT WILL BE TYPE BIT.

SDL DIVISION RESULTS IN AN INTEGER VALUE. ANY REMAINDER IS TRUNCATED THUS:

```
7 / 3 = 2
3 / 7 = 0
```

THE MOD OPERATION IS DIVISION RESULTING IN THE INTEGER VALUE OF THE REMAINDER. IT IS EVALUATED BY THE FOLLOWING FORMULA WHERE SIGN(Y) = -1, IF Y<0 OR, +1 IF Y≥0:

```
Y MOD Z = Y-Z*(SIGN(Y/Z) * ABSOLUTE VALUE (Y/Z))
```

FOR EXAMPLE:

```
 7 MOD  3 = 7-3 * (+1 * ABS 2) = +1
-7 MOD  3 = -7-3 * (-1 * ABS(-2)) = -1
 3 MOD -7 = 3--7 * (-1 * ABS 0) = +3
-3 MOD -7 = -3--7 * (+1 * ABS 0) = -3
```

RELATIONAL OPERATORS

=	EQL	EQUAL TO
≠	NEQ	NOT EQUAL TO
>	GTR	GREATER THAN
<	LSS	LESS THAN
≥	GEQ	GREATER THAN OR EQUAL TO
≤	LEQ	LESS THAN OR EQUAL TO

THE RELATIONAL OPERATORS DO A COMPARISON BETWEEN TWO OPERANDS OF ANY DATA TYPE. A 1-BIT RESULT IS RETURNED -- ●(1)1● IF THE CONDITION IS TRUE, ●(1)0● IF THE CONDITION IS FALSE.

IF BOTH OPERANDS ARE FIXED, THE OPERATOR DOES A TRUE SIGNED COMPARE. IF BOTH OPERANDS ARE CHARACTER STRINGS, THE SHORTER ONE IS PADDED ON THE RIGHT WITH BLANKS, AND A CHARACTER BY CHARACTER MAGNITUDE COMPARE BY COLLATING SEQUENCE IS DONE.

FOR ALL OTHER OPERAND COMBINATIONS, LEADING ZEROES ARE SUPPLIED TO THE SHORTER OF THE TWO. NO SIGN ANALYSIS IS DONE, AND OPERANDS ARE TREATED AS POSITIVE MAGNITUDES.

LOGICAL OPERATORS

NOT
AND
OR
EXOR

THE LOGICAL OPERATORS PERFORM A BIT BY BIT ANALYSIS ON ALL THREE DATA TYPES. "NOT" IS CONSIDERED TO BE A UNARY OPERATOR, AND MAY APPEAR TO THE RIGHT OF ANY OTHER OPERATOR (INCLUDING ITSELF).

THE OTHER OPERATORS REQUIRE TWO OPERANDS. THE SHORTER OF THE TWO IS PADDED ON THE LEFT WITH ZEROES TO DUPLICATE THE LENGTH OF THE LARGER. THE FOLLOWING CHART ILLUSTRATES THE USE OF EACH OPERATOR.

IF X = IF Y =	0 0	0 1	1 0	1 1
NOT X =	1	1	0	0
NOT Y =	1	0	1	0
X AND Y =	0	0	0	1
X OR Y =	0	1	1	1
X EXOR Y =	0	1	1	0

IF X = 00101110 AND Y = 10101100 THEN

NOT X = 11010001

```
X AND Y =  00101100
X OR Y =   10101110
X EXOR Y = 10000010
```

REPLACE OPERATORS

```
<ASSIGNMENT STATEMENT> ::=   <ADDRESS VARIABLE>
                             <REPLACE>
                             <EXPRESSION>

<REPLACE> ::=                <-/ :=

<ASSIGNOR> ::=               <ADDRESS VARIABLE>
                             <NON-DESTRUCTIVE RELACE>
                             <EXPRESSION>

<NON-DESTRUCTIVE
REPLACE> ::=                 <REPLACE, DELETE LEFT PART>
                           / <REPLACE, DELETE RIGHT PART>

<REPLACE, DELETE
LEFT PART> ::=               <-/ :=

<REPLACE, DELETE
RIGHT PART> ::=              :<-/ ::=
```

THERE ARE TWO BASIC TYPES OF REPLACE OPERATORS: THE DESTRUCTIVE <REPLACE> ASSOCIATED WITH THE <ASSIGNMENT STATEMENT>, AND THE <NON-DESTRUCTIVE REPLACE> WHICH OCCURS ONLY WITHIN AN EXPRESSION.

THE DESTRUCTIVE <REPLACE> OPERATOR CAUSES THE EXPRESSION ON ITS RIGHT TO "REPLACE" THE VARIABLE ON ITS LEFT. THE EVALUATION STACK IS FLUSHED SINCE THIS REPLACE IS NECESSARILY THE LAST OPERATION IN THE STATEMENT.

THE <NON-DESTRUCTIVE REPLACE> TAKES TWO FORMS: "DELETE LEFT" AND "DELETE RIGHT". THE "DELETE LEFT" CAUSES THE EXPRESSION TO THE RIGHT OF THE OPERATOR TO REPLACE THE VARIABLE ON ITS LEFT. THE VARIABLE IS THEN DELETED FROM THE TOP OF THE EVALUATION STACK, AND THE EXPRESSION IS LEFT ON THE TOP OF THE STACK.

THE "DELETE RIGHT" CAUSES THE SAME REPLACEMENT. HOWEVER, THE EXPRESSION TO THE RIGHT OF THE OPERATOR IS DELETED FROM THE EVALUATION STACK, AND THE VARIABLE TO THE LEFT REMAINS ON THE TOP OF THE STACK.

THE FOLLOWING EXAMPLE ILLUSTRATES THE USE OF THE <NON-DESTRUCTIVE REPLACE>:

```
PROCEDURE GOOD  BIT VARYING;
     DECLARE X  BIT(48);
     RETURN X ::= "RESULT";
END GOOD;
PROCEDURE BAD  BIT VARYING;
     DECLARE Y  BIT(48);
     RETURN Y := "RESULT";
END BAD;
```

PROCEDURE GOOD WILL EXECUTE PROPERLY SINCE X, DECLARED AS BIT, IS ASSOCIATED WITH THE PROCEDURE TYPE--BIT VARYING. NOTICE, HOWEVER, THAT IN PROCEDURE BAD, Y IS DELETED FROM THE STACK AND THE CHARACTER STRING "RESULT" REMAINS. UNLESS THE CONTROL CARD OPTION "FORMALCHECK" IS SET AT COMPILE TIME, THERE WILL BE NO INDICATION THAT THE DATA TYPES (AS IN PROCEDURE BAD) DO NOT MATCH THE PROCEDURE TYPE. IF "FORMALCHECK" IS SPECIFIED, THE FOLLOWING EXECUTE TIME ERROR MESSAGE WILL BE PRINTED:

"TYPE ERROR IN RETURNED VALUE"

IF BOTH OPERANDS ASSOCIATED WITH ANY REPLACE OPERATOR ARE CHARACTER FIELDS, AND THE RECEIVING FIELD IS LONGER THAN THE SENDING FIELD, TRAILING BLANKS WILL BE ADDED. IF THE RECEIVING FIELD IS SHORTER, CHARACTERS WILL BE TRUNCATED FROM THE RIGHT.

WITH EVERY OTHER COMBINATION OF DATA TYPES, WHEN THE RECEIVING FIELD IS NOT EQUAL IN LENGTH TO THE SENDING FIELD, LEADING BINARY ZEROES WILL BE APPENDED TO THE LARGER RECEIVING FIELD, OR HIGH-ORDER BITS ARE TRUNCATED FROM THE LARGER SENDING FIELD.

ALSO SEE THE REVERSE STORE OPERATION IN THE SECTION ENTITLED "EXECUTE-FUNCTION STATEMENT".

CONCATENATION

DATA ITEMS MAY BE LINKED TOGETHER (CONCATENATED) BY USING THE "CAT" OPERATOR. ALTHOUGH THIS OPERATOR IS INTENDED TO CONCATENATE BIT STRINGS OR CHARACTER STRINGS, IT MAY BE USED WITH ANY COMBINATION OF DATA TYPES. THE RESULT OF ANY CONCATENATION MAY NOT BE GREATER THAN 8191 CHARACTERS OR 65535 BITS.

IF ALL THE OPERANDS ARE CHARACTER STRINGS, THE RESULT IS A CHARACTER STRING. FOR ANY OTHER COMBINATION OF DATA TYPES, THE RESULT IS A BIT STRING. FOR EXAMPLE:

```
LET   A = "B"                          1 CHARACTER
      B = @(1)101@                     3 BITS
      C = +10                          FIXED
THEN
      B CAT B = @(1)101101@            BIT STRING, LENGTH 6
      A CAT A = "BB"                   CHARACTER STRING, LENGTH 2
      A CAT B = @(1)11000010101@       BIT STRING, LENGTH 11
      B CAT C = @(3)5000000012@        BIT STRING, LENGTH 27
                                       (EXPRESSED IN OCTAL)
```

PRIMARY ELEMENTS OF THE EXPRESSION

```
<PRIMARY> ::=          <CONSTANT>
                     / <VARIABLE>
                     / (<EXPRESSION>)
                     / <CONDITIONAL EXPRESSION>
                     / <CASE EXPRESSION>
                     / <BUMPOR>
                     / <DECREMENTOR>
                     / <ASSIGNOR>

<VARIABLE> ::=         <ADDRESS VARIABLE>
                     / <VALUE VARIABLE>
```

A PRIMARY IS THE MOST BASIC COMPONENT OF THE SDL EXPRESSION. TO AVOID UNNECESSARY REPETITION, SEE "BASIC COMPONENTS OF THE SDL LANGUAGE" FOR DISCUSSION OF CONSTANTS, AND SEE "ADDRESS VARIABLES" AND "VALUE VARIABLES" FOR DISCUSSION OF VARIABLES.

CONDITIONAL EXPRESSION

```
<CONDITIONAL
EXPRESSION> ::=        IF <EXPRESSION>
                       THEN <EXPRESSION>
                       ELSE <EXPRESSION>
```

THE EXPRESSION FOLLOWING THE RESERVED WORD "IF" IS EVALUATED. IF THE LOW-ORDER BIT OF THE RESULT IS 1, THE EXPRESSION FOLLOWING "THEN" IS EVALUATED. IF IT IS ZERO, THE EXPRESSION FOLLOWING "ELSE" IS EVALUATED. UNLIKE THE <IF STATEMENT>, THE "ELSE" PART OF THE EXPRESSION MUST BE PRESENT.

CASE EXPRESSION

```
<CASE EXPRESSION> ::=          CASE <EXPRESSION>
                               OF (<EXPRESSION LIST>)

<EXPRESSION LIST> ::=          <EXPRESSION>
                             / <EXPRESSION>,
                               <EXPRESSION LIST>
```

IN THE <CASE EXPRESSION>, THE VALUE OF THE <EXPRESSION> FOLLOWING THE RESERVED WORD "CASE" IS USED AS AN INDEX INTO THE LIST OF EXPRESSIONS. THE EXPRESSION THUS SELECTED IS EVALUATED, AND THE OTHER EXPRESSIONS IN THE LIST IGNORED. THE RANGE OF THE INDEX IS FROM ZERO TO N-1, WHERE N IS THE NUMBER OF <EXPRESSION>S IN THE LIST. AN EXAMPLE OF AN <ASSIGNMENT STATEMENT> CONTAINING A <CASE EXPRESSION> FOLLOWS:

```
A:=CASE I OF (A+B, A-B, A*B, A/B, A MOD B) +
   CASE J OF (Q*F-6, 9, 34+B, (A+B) MOD B, C)
```

IF I=2 AND J=3, THE STATEMENT WILL BE EVALUATED AS FOLLOWS:

```
A:=(A*B) + (A+B) MOD B;
```

BUMP

```
<BUMPOR> ::=                   BUMP <ADDRESS VARIABLE>
                               <MODIFIER>

<MODIFIER> ::=                 <EMPTY>
                             / BY <EXPRESSION>
```

BUMPOR LEAVES ON THE EVALUATION STACK, A DESCRIPTOR OF THE VARIABLE WHICH HAS BEEN INCREMENTED BY THE VALUE OF THE MODIFYING <EXPRESSION>. IF <MODIFIER> IS <EMPTY>, THEN THE VARIABLE IS INCREMENTED BY 1. THE RESULTS OF THE FOLLOWING EXPRESSIONS (WHERE A IS AN <ARRAY IDENTIFIER>) ARE EQUIVALENT:

```
BUMP A(X+Y) BY N
A(X+Y) ::= A(X+Y) + N
```

THE ADVANTAGE OF USING <BUMPOR> IS THAT THE CODE FOR PUTTING THE DESCRIPTOR ON THE STACK IS EXECUTED ONLY ONCE. THUS IT IS MORE EFFICIENT.

LIKE ANY VARIABLE, (<BUMPOR>) WILL CAUSE A VALUE TO BE LOADED TO THE TOP OF THE STACK. HENCE:

```
            P(BUMP X BY C-D);
PASSES X BY ADDRESS BUT,

            P((BUMP X BY C-D));
PASSES X BY VALUE.
```

<BUMPOR> OPERATES ON ALL THREE DATA TYPES. CHARACTER STRINGS ARE TREATED AS IF THEY WERE BIT STRINGS. FOR FIELDS GREATER THAN 24 BITS, ONLY THE LOW-ORDER 24 BITS ARE EVALUATED. IF THE FIELD IS LESS THAN 24 BITS, IT IS PADDED WITH LEADING ZEROES TO 24 BITS.

DECREMENT

<DECREMENTOR> ::=	DECREMENT <ADDRESS VARIABLE> <MODIFIER>
<MODIFIER> ::=	<EMPTY> BY <EXPRESSION>

THE <DECREMENTOR> WORKS EXACTLY LIKE <BUMPOR> EXCEPT THAT THE VARIABLE IS DECREASED BY THE VALUE OF THE <EXPRESSION>. SEE ABOVE.

ASSIGNOR

<ASSIGNOR> ::= <ADDRESS VARIABLE>
<NON-DESTRUCTIVE REPLACE>
<EXPRESSION>

<NON-DESTRUCTIVE REPLACE> ::= <REPLACE, DELETE LEFT PART>
/ <REPLACE, DELETE RIGHT PART>

<REPLACE, DELETE LEFT PART> ::= ←/ :=

<REPLACE, DELETE RIGHT PART> ::= :←/ ::=

WITH THE EXCEPTION OF THE <NON-DESTRUCTIVE REPLACE> OPERATOR, THE <ASSIGNOR> PERFORMS THE SAME FUNCTION AS THE <ASSIGNMENT STATEMENT>. ALL THE RULES WHICH APPLY TO THE <ASSIGNMENT STATEMENT> ALSO APPLY TO THE <ASSIGNOR>. FOR DISCUSSION OF THE <NON-DESTRUCTIVE REPLACE>, SEE THE SECTION ENTITLED "THE REPLACE OPERATOR".

ADDRESS VARIABLES

```
<ADDRESS VARIABLE> ::=          <SIMPLE VARIABLE>
                              / <SUBSCRIPTED VARIABLE>
                              / <INDEXED VARIABLE>
                              / <ADDRESS-GENERATING FUNCTION DESIGNATOR>

<SIMPLE VARIABLE> ::=           <SIMPLE IDENTIFIER>

<SIMPLE IDENTIFIER> ::=         <IDENTIFIER>

<SUBSCRIPTED VARIABLE> ::=      <ARRAY IDENTIFIER>(<EXPRESSION>)

<ARRAY IDENTIFIER> ::=          <IDENTIFIER>
```

AS NOTED ABOVE, <ADDRESS VARIABLE>S MAY TAKE THE FORM OF A <SIMPLE IDENTIFIER>, OR AN <ARRAY IDENTIFIER> FOLLOWED BY AN (<EXPRESSION>) DESIGNATING THE ARRAY ELEMENT IN QUESTION. IN ADDITION, SIMPLE AND ARRAY IDENTIFIERS MAY BE INDEXED.

INDEXING

```
<INDEXED VARIABLE> ::=          <SIMPLE IDENTIFIER> <INDEX PART>
                              / <ARRAY IDENTIFIER> <INDEX PART>

<INDEX PART> ::=                [<EXPRESSION LIST>]
```

EACH OF THE EXPRESSIONS IN THE <INDEX PART> IS EVALUATED, AND THE SUM OF THESE IS FORMED. THIS WILL BE CALLED THE INDEX.

THE INDEXING OPERATION OCCURS FUNCTIONALLY AS FOLLOWS:

1. THE SIMPLE OR ARRAY DESCRIPTOR IS LOADED TO THE TOP OF THE EVALUATION STACK.

2. IF THE DESCRIPTOR IS AN ARRAY DESCRIPTOR, THEN IT IS CONVERTED TO A SIMPLE DESCRIPTOR WHICH

DESCRIBES THE FIRST (ZERO) ELEMENT OF THE ARRAY.

3. THE ADDRESS FIELD OF THE DESCRIPTOR IS MODIFIED BY ADDING TO IT THE INDEX.

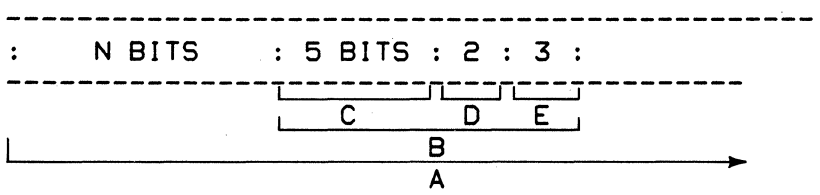
NOTE THAT SELF-RELATIVE DATA ITEMS (I.E., DATA ITEMS WHOSE LENGTH IS NOT GREATER THAN 24, WHICH ARE NOT IN A STRUCTURE, AND WHICH DO NOT REMAP SOME OTHER DATA ITEM) MAY NOT BE INDEXED.

THERE ARE TWO METHODS OF INDEXING:

1. THE DESCRIPTOR PROVIDES THE ADDRESS, AND THE INDEX PROVIDES THE OFFSET FROM THIS ADDRESS.

2. THE DESCRIPTOR PROVIDES THE OFFSET, AND THE INDEX PROVIDES THE ADDRESS.

EXAMPLE:



FIELD D MAY BE ACCESSED USING EITHER METHOD (1) OR METHOD (2). ASSUME N CONTAINS THE OFFSET TO B.

METHOD (1):

```
DECLARE
     01  A BIT(5000),
          02  BB,
               03  CC BIT(5),
               03  DD BIT(2),
               03  EE BIT(3),
     N BIT(24),
     X BIT(2);
/* THE NEXT STATEMENT WILL MOVE DD (WITH THE OFFSET
   GIVEN BY N) INTO X */
X←DD[N];
```

METHOD (2):

```
DECLARE
     A BIT(5000),
     01  BB REMAPS BASE,
          02  CC BIT(5),
          02  DD BIT(2),
          02  EE BIT(3),
     N BIT(24),
     X BIT(2);
/* THE NEXT STATEMENT WILL MOVE DD
   (WITH THE OFFSET GIVEN BY N) INTO X */
X←DD[N, DATA.ADDRESS(A)];
```

NOTE THE FOLLOWING:

1. THE STRUCTURE ABOVE, COMPRISED OF BB, CC, DD, AND EE, WHICH REMAPS BASE IS CALLED A "TEMPLATE".

2. THIS TEMPLATE MAY BE APPLIED TO ANY DATA AREA MERELY BY PROVIDING THE ADDRESS AS PART OF THE INDEX. THIS IS NOT THE CASE WHEN METHOD(1) INDEXING IS USED.

3. THE EXAMPLE ABOVE IS CONTRIVED -- IN METHOD (2), IF N CONTAINED THE ADDRESS OF B RATHER THAN THE OFFSET TO B FROM THE BEGINNING OF A, THEN THE STATEMENTS WHICH STORE D INTO X WOULD BE IDENTICAL: X←DD[N];

ADDRESS GENERATING FUNCTIONS

```
<ADDRESS-GENERATING
FUNCTION DESIGNATOR> ::=      <SUB-STRING ADDRESS DESIGNATOR>
                            / <FETCH COMMUNICATE MESSAGE
                              POINTER DESIGNATOR>
                            / <DESCRIPTOR DESIGNATOR>
                            / <DESCRIPTOR-GENERATOR DESIGNATOR>
                            / <ADDRESS-MODIFIER DESIGNATOR>
```

SUBBIT AND SUBSTR

```
<SUB-STRING ADDRESS
DESIGNATOR> ::=               <SUB-STRING FUNCTION IDENTIFIER>
                              (<STRING ADDRESS>,<OFFSET PART>)
                            / <SUB-STRING FUNCTION IDENTIFIER>
                              (<STRING ADDRESS>,<OFFSET PART>,
                              <LENGTH PART>)

<SUB-STRING FUNCTION
IDENTIFIER> ::=               SUBBIT / SUBSTR

<STRING ADDRESS> ::=          <ADDRESS GENERATOR>

<ADDRESS GENERATOR> ::=       SEE "ADDRESS GENERATOR"

<OFFSET PART> ::=             <EXPRESSION>

<LENGTH PART> ::=             <EXPRESSION>
```

SUBSTR YIELDS A SUB-STRING OF A CHARACTER STRING IDENTIFIED BY THE <STRING ADDRESS>. THE BEGINNING CHARACTER OF THE SUB-STRING IS SPECIFIED BY THE <OFFSET PART> (WHERE THE FIRST CHARACTER OF THE STRING IS ZERO). THE <LENGTH PART> SPECIFIES THE LENGTH OF THE SUB-STRING. IF OMITTED, THE REST OF THE STRING FROM THE "OFFSET" CHARACTER IS ASSUMED. FOR EXAMPLE:

```
IF X←"CHARACTER"
   C←"COALITION"
THEN
   SUBSTR(X,4)←SUBSTR(C,0,4)
YIELDS THE CHARACTER STRING:
   "CHARCOAL "
```

LIKE ALL CHARACTER-TO-CHARACTER STORE OPERATIONS, IF THE RECEIVING FIELD IS LARGER THAN THE SENDING FIELD, THE SENDING FIELD IS PADDED WITH BLANKS ON THE RIGHT. IF THE SENDING FIELD IS LONGER, CHARACTERS ARE TRUNCATED FROM THE RIGHT. NOTE THAT THIS IS A FUNCTION OF THE STORE OPERATOR AND NOT SUBSTR.

SUBBIT YIELDS A SUB-STRING OF A BIT STRING IDENTIFIED BY THE <STRING ADDRESS>. THE BEGINNING BIT OF THE SUB-STRING IS SPECIFIED BY THE <OFFSET PART> (NOTE: THE FIRST BIT OF THE STRING IS 0). THE LENGTH OF THE SUB-STRING IS SPECIFIED BY THE <LENGTH PART> WHICH, IF OMITTED, WILL BE ASSUMED TO BE THE REST OF THE STRING.

EXAMPLE:

```
IF A←@(1)00101011010@
   B←@(1)00001111010@
THEN
   SUBBIT(A,2,3) CAT SUBBIT(B,5)
RESULTS IN:
   @(1)101111010@
AND
   SUBBIT(A,3) CAT SUBBIT(B,0,6)
RESULTS IN:
   @(1)010110100000110@
```

FETCH.COMMUNICATE.MSG.PTR

<FETCH COMMUNICATE MESSAGE POINTER DESIGNATOR> ::= FETCH.COMMUNICATE.MSG.PTR

SEE THE B1700 MCP REFERENCE MANUAL FOR A DESCRIPTION OF THE RUN

STRUCTURE.

IF THE RS.MCP.BIT IS SET, THEN RS.COMMUNICATE.MSG.PTR IS ACCESSED. OTHERWISE, RS.REINSTATE.MSG.PTR IS ACCESSED. THE ACCESSED FIELD IS ASSUMED TO BE A DESCRIPTOR AND IS PLACED ON THE TOP OF THE EVALUATION STACK.

EXAMPLE:

```
DESCRIPTOR(COMM.MSG)  FETCH.COMMUNICATE.MSG.PTR;
```

COMM.MSG NOW DESCRIBES THE COMMUNICATE MESSAGE, ASSUMING THAT THE MESSAGE WAS DESCRIBED BY A NON-SELF-RELATIVE DESCRIPTOR.

DESCRIPTORS

```
<DESCRIPTOR DESIGNATOR>::=   DESCRIPTOR (<SIMPLE IDENTIFIER>)
                           / DESCRIPTOR (<ARRAY IDENTIFIER>)
```

"DESCRIPTOR" PLACES ON THE EVALUATION STACK, A DESCRIPTOR WHICH DESCRIBES THE DESCRIPTOR OF A <SIMPLE IDENTIFIER> OR AN <ARRAY IDENTIFIER>. THE DESCRIPTOR FUNCTION MAY APPEAR AS THE OBJECT OF A REPLACEMENT, THEREBY PROVIDING EASY ACCESS TO ANY PART OF A DESCRIPTOR.

EXAMPLE:

```
1.   SUBBIT(DESCRIPTOR(X),4,2)←2;

2.   DESCRIPTOR(X)←DESCRIPTOR(Y);
```

EXAMPLE (2) FORCES BOTH X AND Y TO DESCRIBE THE SAME DATA NAME. NOTE, HOWEVER, THAT IF X AND Y ARE NOT EITHER BOTH SIMPLE ITEMS OR BOTH ARRAYS, THE RESULT WILL BE INCORRECT.

MAKE.DESCRIPTOR

<DESCRIPTOR-GENERATOR DESIGNATOR> ::= MAKE.DESCRIPTOR(<EXPRESSION>)

THE VALUE WHICH IS GENERATED BY THE <EXPRESSION> IS ASSUMED TO BE A DESCRIPTOR. THIS DESCRIPTOR REPLACES ON THE EVALUATION STACK, THE DESCRIPTOR REPRESENTING THAT <EXPRESSION>. IF THE NAME-VALUE BIT OF THE EXPRESSION-S DESCRIPTOR ON THE EVALUATION STACK IS SET, THEN THE VALUE OF THE <EXPRESSION> IS REMOVED FROM THE VALUE STACK.

A <DESCRIPTOR GENERATOR DESIGNATOR> MAY APPEAR AS THE OBJECT OF A REPLACEMENT, HOWEVER THE PROGRAMMER IS RESPONSIBLE TO SEE THAT THE DESCRIPTOR BUILT GENERATES AN ADDRESS. THERE IS NO SYNTAX CHECK FOR THIS.

THE FOLLOWING EXAMPLES ILLUSTRATE THE RELATIONSHIPS BETWEEN THE DESCRIPTOR FUNCTIONS:

DESCRIPTOR(X)=VALUE.DESCRIPTOR(X),
WHERE X IS NON-SELF-RELATIVE

MAKE.DESCRIPTOR (DESCRIPTOR(X)) = X,
WHERE X IS NON-SELF-RELATIVE

MAKE.DESCRIPTOR (VALUE.DESCRIPTOR(E)) = E,
WHERE E IS AN <ADDRESS GENERATOR>

VALUE.DESCRIPTOR (MAKE.DESCRIPTOR(E)) = E,
WHERE THE VALUE OF E IS A VALID <ADDRESS GENERATOR>

NEXT-PREVIOUS.ITEM

```
<ADDRESS-MODIFIER
DESIGNATOR> ::=              <ADDRESS-MODIFIER FUNCTION IDENTIFIER>
                             (<SIMPLE IDENTIFIER>)

<ADDRESS-MODIFIER
FUNCTION IDENTIFIER> ::=       NEXT.ITEM
                             / PREVIOUS.ITEM
```

THE NEXT.ITEM FUNCTION CAUSES THE LENGTH FIELD OF THE DESCRIPTOR REPRESENTED BY THE <SIMPLE IDENTIFIER> TO BE ADDED TO THE ADDRESS FIELD OF THAT DESCRIPTOR. THIS MODIFIED DESCRIPTOR IS PUT BACK ONTO THE NAME STACK, AND ALSO MOVED TO THE TOP OF THE EVALUATION STACK. MOVING THE MODIFIED DESCRIPTOR TO THE EVALUATION STACK IS, IN EFFECT, A LOAD ADDRESS OF THE NEW ITEM DESCRIBED BY THE <SIMPLE IDENTIFIER>. HENCE, "NEXT.ITEM" MAY BE USED AS THE OBJECT OF A REPLACEMENT. FOR EXAMPLE, THE FOLLOWING STATEMENTS:

```
DECLARE 01 CHAR.STRING CHARACTER(1000),
     02 NEXT.CHAR CHARACTER(1);
NEXT.ITEM (NEXT.CHAR)←"D";
```

HAVE THE EFFECT OF STORING "D" INTO THE SECOND CHARACTER OF CHAR.STRING, WHICH IS:

```
SUBSTR(CHAR.STRING,1,1)
```

THE PREVIOUS.ITEM FUNCTION IS IDENTICAL TO NEXT.ITEM EXCEPT THAT A SUBTRACTION (OF LENGTH FROM ADDRESS) IS PERFORMED.

ADDRESS GENERATORS

```
<ADDRESS
GENERATOR LIST> ::=          <ADDRESS GENERATOR>
                           / <ADDRESS GENERATOR>,
                             <ADDRESS GENERATOR LIST>

<ADDRESS GENERATOR> ::=      <ADDRESS VARIABLE>
                           / <BUMPOR>
                           / <DECREMENTOR>
                           / <CONDITIONAL ADDRESS GENERATOR>
                           / <CASE ADDRESS GENERATOR>
                           / <ADDRESS-GENERATING ASSIGNOR>

<BUMPOR> ::=                 SEE "BUMPOR"

<DECREMENTOR> ::=            SEE "DECREMENTOR"

<CONDITIONAL ADDRESS
GENERATOR> ::=               IF <EXPRESSION>
                             THEN <ADDRESS GENERATOR>
                             ELSE <ADDRESS GENERATOR>

<CASE ADDRESS
GENERATOR> ::=               CASE <EXPRESSION>
                             OF (<ADDRESS GENERATOR LIST>)

<ADDRESS-GENERATING
ASSIGNOR> ::=                <ADDRESS VARIABLE>
                             <REPLACE, DELETE LEFT PART>
                             <ADDRESS GENERATOR>
                           / <ADDRESS VARIABLE>
                             <REPLACE, DELETE RIGHT PART>
                             <EXPRESSION>
```

THE <ADDRESS GENERATOR> INCLUDES ANY PRIMARY WHICH LEAVES AN ADDRESS ON THE TOP OF THE EVALUATION STACK. SEE "PRIMARY ELEMENTS OF THE EXPRESSION" FOR MORE EXPLICIT DETAIL.

VALUE VARIABLES

```
<VALUE VARIABLE> ::=          <VALUE-GENERATING FUNCTION DESIGNATOR>
                            / <TYPED PROCEDURE DESIGNATOR>
                            / (<ADDRESS VARIABLE>)
                            / <FILE DESIGNATOR>

<FILE DESIGNATOR>::=          <FILE IDENTIFIER>
                            / <SWITCH FILE IDENTIFIER> (<EXPRESSION>)

<TYPED PROCEDURE
DESIGNATOR> ::=               <TYPED PROCEDURE IDENTIFIER>
                              <ACTUAL PARAMETER PART>

<TYPED PROCEDURE
IDENTIFIER> ::=               <IDENTIFIER>

<ACTUAL PARAMETER PART> ::=   <EMPTY>
                            / (<ACTUAL PARAMETER LIST>)

<ACTUAL PARAMETER LIST> ::=   <ACTUAL PARAMETER>
                            / <ACTUAL PARAMETER>,
                              <ACTUAL PARAMETER LIST>

<ACTUAL PARAMETER> ::=        <EXPRESSION>
                            / <ARRAY DESIGNATOR>

<ARRAY DESIGNATOR> ::=        <ARRAY IDENTIFIER>

<ARRAY IDENTIFIER> ::=        <IDENTIFIER>
```

NOTICE FROM THE ABOVE SYNTAX THAT ANY <ADDRESS VARIABLE> ENCLOSED IN PARENS, SUCH AS (SUBBIT (A,I,J)), WILL BE TREATED AS A VALUE VARIABLE.

THE VALUE GENERATED BY A <FILE DESIGNATOR> IS THE FPB NUMBER OF THE SPECIFIED FILE. A WARNING MESSAGE WILL BE ISSUED.

THE TYPED PROCEDURE (A PROCEDURE WHICH RETURNS A VALUE) IS INVOKED WITHIN AN EXPRESSION ACCORDING TO THE ABOVE SYNTAX. THE PROCEDURE IDENTIFIER, FOLLOWED BY ITS PARAMETERS (IF ANY), ENCLOSED WITHIN PARENS, IS TREATED AS AN OPERAND IN THE EXPRESSION. FOR DETAILS ON PASSING PARAMETERS, SEE "ADDRESS AND

VALUE PARAMETERS". THE PROCEDURE IS EVALUATED AND THE RETURNED
VALUE REPLACES THE <TYPED PROCEDURE DESIGNATOR>. FOR EXAMPLE:

```
DECLARE Z FIXED;
PROCEDURE X(A,B) FIXED;
     FORMAL (A,B) FIXED;
               .
               .
               .
END X;
Z := X(BUMP M,R)+1;
```

VALUE GENERATING FUNCTIONS

<VALUE-GENERATING
FUNCTION DESIGNATOR> ::= <SWAP DESIGNATOR>
/ <SUB-STRING VALUE DESIGNATOR>
/ <DISPATCH DESIGNATOR>
/ <LOCATION DESIGNATOR>
/ <CONVERT DESIGNATOR>
/ <LENGTH DESIGNATOR>
/ <MEMORY SIZE DESIGNATOR>
/ <DESCRIPTOR-VALUE-GENERATOR DESIGNATOR>
/ <INTERROGATE INTERRUPT STATUS DESIGNATOR>
/ <DECIMAL CONVERSION DESIGNATOR>
/ <BINARY CONVERSION DESIGNATOR>
/ <TIME FUNCTION DESIGNATOR>
/ <DATE FUNCTION DESIGNATOR>
/ <NAME-OF-DAY FUNCTION DESIGNATOR>
/ <BASE REGISTER DESIGNATOR>
/ <LIMIT REGISTER DESIGNATOR>
/ <CONTROL STACK TOP DESIGNATOR>
/ <DATA ADDRESS DESIGNATOR>
/ <SEARCH.LINKED.LIST DESIGNATOR>
/ <SORT.STEP.DOWN DESIGNATOR>
/ <SORT.UNBLOCK DESIGNATOR>
/ <SORT.SEARCH DESIGNATOR>
/ <PARITY.ADDRESS DESIGNATOR>
/ <DYNAMIC MEMORY BASE DESIGNATOR>
/ <HASH CODE DESIGNATOR>
/ <NEXT TOKEN DESIGNATOR>
/ <DELIMITED TOKEN DESIGNATOR>
/ <EVALUATION STACK TOP DESIGNATOR>
/ <CONTROL STACK BITS DESIGNATOR>
/ <NAME STACK TOP DESIGNATOR>

/ <DISPLAY BASE DESIGNATOR>
/ <CONSOLE SWITCHES DESIGNATOR>
/ <SEARCH SERIAL LIST DISIGNATOR>
/ <SPO INPUT PRESENT DESIGNATOR>
/ <SEARCH.SDL.STACKS DESIGNATOR>
/ <EXECUTE DESIGNATOR>

SWAP

<SWAP DESIGNATOR> ::= SWAP (<ADDRESS GENERATOR>,<EXPRESSION>)

THE LENGTH OF THE VALUE DESCRIBED BY THE <ADDRESS GENERATOR> IS USED AS THE LENGTH, L, OF THE DATA TO BE "SWAPPED". HOWEVER, IF THE LENGTH OF THE VALUE IS GREATER THAN 24 BITS, L WILL BE 24 BITS, AND ONLY THE LOW-ORDER 24 BITS OF THE <ADDRESS GENERATOR> WILL BE MODIFIED.

THE RIGHTMOST L BITS OF THE VALUE DESCRIBED BY THE <ADDRESS GENERATOR> ARE ISOLATED, AND BECOME THE DESTINATION FIELD.

THE RIGHTMOST L BITS OF THE VALUE GENERATED BY THE <EXPRESSION> ARE ISOLATED. LEADING ZEROES ARE SUPPLIED IF THE LENGTH OF THE VALUE IS LESS THAN L BITS LONG. THIS FIELD IS KNOWN AS THE SOURCE FIELD.

THE SOURCE FIELD IS STORED INTO THE DESTINATION FIELD, THE ORIGINAL VALUE OF WHICH IS THE VALUE RETURNED. THE RETURNED VALUE IS OF TYPE BIT AND OF LENGTH L.

EXAMPLE:

```
A←0;
IF SWAP (A,1) THEN DO ... END;
              ELSE DO ... END;
```

IN THE ABOVE EXAMPLE, THE "ELSE" PART OF THE STATEMENT IS EVALUATED, SINCE A WAS ORIGINALLY SET TO 0 (I.E., FALSE). AT THE END OF THE EVALUATION, 1 HAS BEEN STORED INTO A, AND 0 RETURNED TO THE TOP OF THE EVALUATION STACK.

SUBBIT AND SUBSTR

```
<SUB-STRING VALUE
DESIGNATOR> ::=          <SUB-STRING FUNCTION IDENTIFIER>
                         (<STRING VALUE>,<OFFSET PART>)
                       / <SUB-STRING FUNCTION IDENTIFIER>
                         (<STRING VALUE>,<OFFSET PART>,
                         <LENGTH PART>)

<SUB-STRING FUNCTION
IDENTIFIER> ::=          SUBBIT / SUBSTR

<STRING VALUE> ::=       <EXPRESSION>

<OFFSET PART> ::=        <EXPRESSION>

<LENGTH PART> ::=        <EXPRESSION>
```

THE <SUB-STRING VALUE DESIGNATOR> AND THE <SUB-STRING ADDRESS DESIGNATOR> ARE IDENTICAL EXCEPT THAT THE FORMER RETURNS A VALUE IF ITS <STRING VALUE> IS NOT AN <ADDRESS GENERATOR>. PLEASE SEE "SUBBIT AND SUBSTR" UNDER "ADDRESS VARIABLES" FOR THE SPECIFICS OF THE FUNCTION.

THE FOLLOWING EXAMPLES ILLUSTRATE SOME OF THE USES OF THE <SUB-STRING VALUE DESIGNATOR>:

```
X←SUBSTR(A CAT B,5,10);
MAKE.DESCRIPTOR(@48@ CAT SUBBIT(A OR B, 0, 16) CAT X) ...;
IF SUBSTR(@06@ CAT "ABC", 0) = Y THEN ...;
```

DISPATCH

```
<DISPATCH DESIGNATOR> ::=   DISPATCH(<PORT,CHANNEL,PRIORITY>,
                            <I/O DESPRIPTOR ADDRESS>)

<PORT,CHANNEL,PRIORITY> ::= <EXPRESSION>

<I/O DESCRIPTOR
ADDRESS> ::=                <EXPRESSION>
```

THE RIGHTMOST SEVEN BITS OF THE VALUE OF <PORT, CHANNEL, PRIORITY> CONTAIN THE FOLLOWING INFORMATION FROM LEFT TO RIGHT:

3 BITS	3 BITS	1 BIT

: PORT : CHANNEL : PRIORITY :

THE RIGHTMOST 24 BITS OF THE VALUE OF THE <I/O DESCRIPTOR ADDRESS> IS THE ABSOLUTE ADDRESS OF THE I/O DESCRIPTOR.

USING THESE TWO VALUES, AN I/O OPERATION IS INITIATED. A BIT VALUE WITH THE FOLLOWING MEANINGS IS RETURNED:

0 = DISPATCH REGISTER LOCK BIT SET
1 = SUCCESSFUL DISPATCH
2 = SUCCESSFUL DISPATCH, BUT MISSING DEVICE

LOCATION

<LOCATION DESIGNATOR> ::= LOCATION (<PROCEDURE IDENTIFIER>)
/ LOCATION (<SIMPLE IDENTIFIER>)
/ LOCATION (<ARRAY IDENTIFIER>)

<PROCEDURE IDENTIFIER> ::= <IDENTIFIER>

<SIMPLE IDENTIFIER>::= <IDENTIFIER>

<ARRAY IDENTIFIER>:: <IDENTIFIER>

FOR PROCEDURES, THE <LOCATION DESIGNATOR> RETURNS A 33-BIT VALUE (TYPED BIT) CONTAINING, FROM LEFT TO RIGHT:

ADDRESS TYPE, CONTAINING @(3)6@	3 BITS
SEGMENT NUMBER	6 BITS
PAGE NUMBER	4 BITS
DISPLACEMENT	20 BITS

THIS 33-BIT VALUE IS THE ADDRESS OF THE PROCEDURE IN QUESTION.

A FORWARD DECLARATION IS REQUIRED ONLY DURING RECOMPILATION OR CREATE-MASTER FOR ANY PROCEDURE ON WHICH A LOCATION IS PERFORMED. AN ERROR IS GIVEN IF THIS IS NOT DONE .

FOR SIMPLE AND ARRAY IDENTIFIERS, THE <LOCATION DESIGNATOR> RETURNS A 16-BIT VALUE (TYPED BIT) CONTAINING, FROM LEFT TO RIGHT:

ADDRESS TYPE CONTAINING @(2)0@ 2 BITS

LEXIC LEVEL	4 BITS
OCCURRENCE NUMBER	10 BITS

CONVERT

```
<CONVERSION DESIGNATOR> ::=  CONVERT (<EXPRESSION>,
                               <CONVERSION PART>)
                           / CONV (<EXPRESSION>,
                               <CONVERSION PART>)

<CONVERSION PART> ::=          <CONVERSION TYPE>
                           / <CONVERSION TYPE>,
                               <BIT GROUP SIZE>

<CONVERSION TYPE> ::=          BIT / CHARACTER / FIXED

<BIT GROUP SIZE> ::=           1 / 2 / 3 / 4
```

THE <EXPRESSION>, WHICH MAY BE OF ANY DATA TYPE, WILL BE CONVERTED AS SPECIFIED BY THE <CONVERSION TYPE>. THE CONVERTED <EXPRESSION> WILL BE RETURNED AS A VALUE.

THE <BIT GROUP SIZE> IS USED ONLY WITH BIT-TO-CHARACTER OR CHARACTER-TO-BIT CONVERSIONS. IT SPECIFIES THE NUMBER OF BITS (OF THE BIT STRING) WHICH CORRESPOND TO A CHARACTER IN THE CHARACTER STRING.

NOTE: BIT-TO-CHARACTER CONVERSION DOES NOT YIELD DECIMAL DIGITS. IF A BIT STRING IS TO BE CONVERTED TO DECIMAL DIGITS, IT SHOULD BE STORED IN A FIXED VARIABLE, AND THE FIXED VARIABLE CONVERTED.

THE FOLLOWING TABLE SHOWS THE POSSIBLE CONVERSION COMBINATIONS:

OUTPUT:	BIT	CHARACTER	FIXED
INPUT: BIT	NO CHANGE	CONVERT TO CHAR. UNDER CONTROL OF <BIT GROUP SIZE>; IF OMITTED USE 4	RETURN 24 BITS PROVIDING LEADING ZEROES OR LEFT TRUNCATION, AS NECESSARY
CHARACTER	CONVERT TO BITS UNDER CONTROL OF <BIT GROUP SIZE>; IF OMITTED USE 4	NO CHANGE	SEE NOTE
FIXED	CHANGE TYPE TO BIT	DECIMAL CONVERSION W/ LEADING ZEROS & SIGN NOT SUPPRESSED. (7 DIGITS + SIGN).	NO CHANGE

NOTE: THE CHARACTER STRING MAY HAVE LEADING BLANKS, SIGN (OR NONE), MORE BLANKS, AND DECIMAL DIGITS. A PLUS SIGN IS IGNORED. THE DECIMAL DIGITS (ONLY THE LOW-ORDER 7) ARE CONVERTED TO A BINARY NUMBER THAT IS RIGHT JUSTIFIED IN A 24-BIT FIELD. IF THE SIGN WAS MINUS, THEN THE 2-S COMPLEMENT OF THE 24-BIT FIELD IS RETURNED.

EXAMPLES:

```
CONVERT (" -  72581",FIXED)     RETURNS -72581
CONVERT (⊛(3)752⊛,CHARACTER,4)          "1EA"
CONVERT (⊛(1)11011⊛,FIXED)              27
CONVERT ("132",BIT,2)                   ⊛(2)132⊛
CONVERT ("132",BIT,4)                   ⊛(4)132⊛
CONVERT ("2 ",BIT)                      ⊛(4)20⊛
```

LENGTH

<LENGTH DESIGNATOR> ::= LENGTH (<EXPRESSION>)

THE <LENGTH DESIGNATOR> RETURNS A 24-BIT, TYPE BIT FIELD CONTAINING THE NUMBER OF UNITS IN THE <EXPRESSION>. IF THE <EXPRESSION> IS TYPE CHARACTER, THEN EACH CHARACTER IS A UNIT. OTHERWISE, EACH BIT IS A UNIT.

MEMORY SIZE

<MEMORY SIZE DESIGNATOR> ::= S.MEM.SIZE / M.MEM.SIZE

THE REQUESTED MEMORY SIZE IS RETURNED AS A 24-BIT DATA ITEM OF TYPE BIT.

VALUE DESCRIPTOR

<DESCRIPTOR-VALUE GENERATOR DESIGNATOR> ::= VALUE.DESCRIPTOR (<ADDRESS GENERATOR>)

<ADDRESS GENERATOR> ::= SEE "ADDRESS GENERATORS"

THE <ADDRESS GENERATOR> IS REPRESENTED BY A DESCRIPTOR AT THE TOP OF THE EVALUATION STACK. THIS DESCRIPTOR IS MOVED TO THE VALUE STACK. IN ITS PLACE ON THE EVALUATION STACK IS LEFT A DESCRIPTOR DESCRIBING THE ONE JUST MOVED TO THE VALUE STACK.

THE NAME-VALUE BIT IS SET IN THE DESCRIPTOR LEFT IN THE EVALUATION STACK.

INTERROGATE INTERRUPT STATUS

<INTERROGATE INTERRUPT STATUS DESIGNATOR> ::= INTERROGATE.INTERRUPT.STATUS

A 24-BIT DATA ITEM OF TYPE BIT IS RETURNED. THE VALUE REPRESENTS THE INTERRUPT BITS OF THE M-MACHINE. THE APPLICABLE M-MACHINE INTERRUPT BITS ARE RESET. NOTE THAT THE INCN BITS WILL NOT BE RESET.

DECIMAL CONVERSION

<DECIMAL CONVERSION DESIGNATOR> ::= DECIMAL (<EXPRESSION>, <DECIMAL STRING SIZE>)

<DECIMAL STRING SIZE> ::= <EXPRESSION>

THE VALUE OF THE FIRST <EXPRESSION> FOLLOWING THE RESERVED WORD "DECIMAL" IS CONVERTED TO A STRING OF DECIMAL CHARACTERS. IF THE VALUE OF THE <EXPRESSION> GENERATES MORE THAN 24 BITS, THEN ONLY THE LOW-ORDER 24 BITS ARE USED.

THE NUMBER OF CHARACTERS RETURNED IS GIVEN BY THE VALUE OF THE <DECIMAL STRING SIZE>. A MAXIMUM OF 8 DECIMAL CHARACTERS WILL BE RETURNED, EVEN IF THE VALUE OF THE <DECIMAL STRING SIZE> IS GREATER. IF THE <DECIMAL STRING SIZE> IS LESS THAN THE NUMBER OF DECIMAL CHARACTERS, THEN CHARACTERS ARE TRUNCATED FROM THE LEFT.

BINARY CONVERSION

<BINARY CONVERSION DESIGNATOR> ::= BINARY (<EXPRESSION>)

THE <BINARY CONVERSION DESIGNATOR> RETURNS A FIXED VALUE WHICH IS THE BINARY REPRESENTATION OF THE <EXPRESSION>. THE <EXPRESSION> IS ASSUMED TO BE A CHARACTER STRING CONTAINING DECIMAL DIGITS. ONLY THE LOW-ORDER 8 CHARACTERS WILL BE CONVERTED. ZONE BITS ARE IGNORED.

IF THE CONVERSION RESULTS IN A BINARY VALUE GREATER THAN 24 BITS (I.E., IF THE DECIMAL NUMBER IS GREATER THAN 16,777,215), THEN THE LEFT-MOST BITS WILL BE TRUNCATED.

IF THE DECIMAL NUMBER IS GREATER THAN 8,388,607 (I.E., (2 EXP 23)-1), THEN THE RETURNED VALUE WILL APPEAR TO BE NEGATIVE (I.E., THE HIGH-ORDER BIT IS 1).

TIME FUNCTION

<TIME FUNCTION DESIGNATOR> ::= TIME
/ TIME (<TIME FORMAT>,<REPRESENTATION>)

<TIME FORMAT> ::= COUNTER / MILITARY / CIVILIAN

<REPRESENTATION> ::= BIT / DIGIT / CHARACTER

THE <TIME FUNCTION DESIGNATOR> RETURNS A BIT OR CHARACTER STRING WHICH IS THE TIME OF THE FUNCTION-S EXECUTION. THE <TIME FORMAT> MAY HAVE THREE BASIC FORMATS:

COUNTER	RETURNS THE TIME OF DAY IN TENTHS OF SECONDS.
MILITARY	RETURNS THE TIME OF DAY IN THE FOLLOWING FORM -- HHMMSST (WHERE T=TENTHS OF SECONDS).
CIVILIAN	RETURNS HHMMSSTAP(WHERE AP=AM OR PM).

THE TIME OF DAY MAY BE REPRESENTED IN EITHER BITS, DIGITS, OR CHARACTERS IN THE FOLLOWING FORMATS:

	BIT	DIGIT	CHARACTER
COUNTER	20 BITS	24 BITS	48 BITS
MILITARY	5+6+6+4=21	8+8+8+4=28	16+16+16+8=56
CIVILIAN	4+6+6+4+16=36	8+8+8+4+16=44	16+16+16+8+16=72

NOTE: "TIME" AND "TIME (CIVILIAN,CHARACTER)" ARE EQUIVALENT.

DATE FUNCTION

```
<DATE FUNCTION
DESIGNATOR> ::=              DATE
                           / DATE (<DATE FORMAT>,<REPRESENTATION>)

<DATE FORMAT> ::=            JULIAN / MONTH / DAY / YEAR

<REPRESENTATION> ::=         BIT / DIGIT / CHARACTER
```

THE <DATE FUNCTION DESIGNATOR> RETURNS A BIT OR CHARACTER STRING WHICH IS THE DATE OF THE EXECUTION OF THE FUNCTION.

"DATE" AND "DATE (MONTH,CHARACTER)" ARE EQUIVALENT.

THE FORMATS (IN BITS) OF THE RETURNED STRINGS ARE:

	BIT	DIGIT	CHARACTER
JULIAN (YYDDD)	7+9=16	8+12=20	16+24=40
MONTH (MMDDYY)	4+5+7=16	8+8+8=24	16+16+16=48
DAY (DDMMYY)	5+4+7=16	8+8+8=24	16+16+16=48
YEAR (YYMMDD)	7+4+5=16	8+8+8=24	16+16+16=48

```
EXAMPLE: DECLARE D CHARACTER(5);
         DDATE (JULIAN,CHARACTER);
```

NAME OF DAY

```
<NAME OF DAY FUNCTION
DESIGNATOR> ::=              NAME.OF.DAY
```

A CHARACTER STRING, WHICH IS THE NAME OF THE DAY OF THE WEEK, IS RETURNED AS A 9-CHARACTER STRING. THE NAME IS LEFT JUSTIFIED.

```
EXAMPLE: DECLARE DAY CHARACTER(9);
         DAYNAME.OF.DAY
```

BASE REGISTER

<BASE REGISTER DESIGNATOR> ::= BASE.REGISTER

A 24-BIT VALUE OF TYPE BIT IS RETURNED. THE VALUE IS THE ABSOLUTE ADDRESS OF THE BASE OF THE PROGRAM. IT SHOULD BE NOTED THAT TWO SEPARATE EXECUTIONS OF "BASE.REGISTER" MAY NOT YIELD THE SAME RESULTS, SINCE THE MCP MAY HAVE MOVED THE PROGRAM IN MEMORY.

LIMIT REGISTER

<LIMIT REGISTER DESIGNATOR> ::= LIMIT.REGISTER

THE <LIMIT REGISTER DESIGNATOR> RETURNS A 24-BIT VALUE (TYPE BIT) WHICH IS THE BASE RELATIVE ADDRESS OF THE PROGRAM-S RUN STRUCTURE. FOR FURTHER EXPLANATION, PLEASE REFER TO THE "B1500 MCP MANUAL".

CONTROL STACK TOP

<CONTROL STACK TOP DESIGNATOR> ::= CONTROL.STACK.TOP

A 24-BIT VALUE OF TYPE BIT IS RETURNED. THE VALUE IS THE BASE RELATIVE ADDRESS OF THE NEXT ENTRY TO BE PLACED ON THE CONTROL STACK.

DATA ADDRESS

<DATA ADDRESS DESIGNATOR> ::= DATA.ADDRESS (<ADDRESS GENERATOR>)

<ADDRESS GENERATOR> ::= SEE "ADDRESS GENERATORS"

THE <DATA ADDRESS DESIGNATOR> RETURNS A 24-BIT VALUE (TYPE BIT)

WHICH IS THE BASE RELATIVE ADDRESS GENERATED BY THE <ADDRESS GENERATOR>.

SEARCH.LINKED.LIST

<SEARCH.LINKED.LIST DESIGNATOR> ::=	SEARCH.LINKED.LIST (<RECORD ADDRESS>,<ARGUMENT INDEX>, <COMPARE VARIABLE>,<RELATION>, <LINK INDEX>)
<RECORD ADDRESS> ::=	<EXPRESSION>
<ARGUMENT INDEX> ::=	<EXPRESSION>
<COMPARE VARIABLE> ::=	<EXPRESSION>
<RELATION> ::=	< / ≤ / = / ≠ / ≥ / > / LSS / LEQ / EQL / NEQ / GEQ / GTR
<LINK INDEX> ::=	<EXPRESSION>

EACH OF THE FOUR EXPRESSIONS ABOVE GENERATES A 24-BIT VALUE WHICH IS LOADED TO THE TOP OF THE EVALUATION STACK. THE MEANINGS OF EACH EXPRESSION IS AS FOLLOWS:

1. THE <RECORD ADDRESS> IS THE BASE RELATIVE ADDRESS OF THE FIRST STRUCTURE TO BE EXAMINED.

2. THE <ARGUMENT INDEX> IS THE RELATIVE OFFSET AND SIZE IN THE STRUCTURE, OF THE 24 (OR LESS) BIT FIELD BEING COMPARED WITH THE <COMPARE VARIABLE>.

3. THE <COMPARE VARIABLE> IS THE CONTROL AGAINST WHICH THE SPECIFIED FIELD IN THE STRUCTURE IS COMPARED.

4. THE <RELATION> SPECIFIES THE DESIRED RELATION IN THE COMPARISON OF THE TWO VALUES.

5. THE <LINK INDEX> IS THE RELATIVE OFFSET AND SIZE IN THE STRUCTURE, OF THE 24 (OR LESS) BIT FIELD CONTAINING THE ADDRESS OF THE NEXT STRUCTURE TO BE EXAMINED (IF COMPARISON WITH THE CURRENT STRUCTURE FAILS).

THE LAST STRUCTURE IN THE LINKED LIST CONTAINS ALL 1 BITS IN THE FIELD DESCRIBED BY THE <LINK INDEX>.

THE LINKED LIST IS SEARCHED UNTIL THE DESIRED COMPARISON SUCCEEDS OR UNTIL THE COMPARISON FAILS WITH THE LAST STRUCTURE.

IF THE SEARCH SUCCEEDS, THE BASE-RELATIVE ADDRESS OF THE CURRENT STRUCTURE IS LEFT ON THE EVALUATION STACK AS A 24-BIT VALUE. IF THE SEARCH FAILS, @FFFFFF@ IS LEFT ON THE STACK.

SORT.STEP.DOWN

<SORT.STEP.DOWN DESIGNATOR> ::= SORT.STEP.DOWN (<RECORD 1>,<RECORD 2>, <KEY TABLE ADDRESS>)

<RECORD 1> ::= <EXPRESSION>

<RECORD 2> ::= <EXPRESSION>

<KEY TABLE ADDRESS> ::= <EXPRESSION>

FOR USE BY SORT ONLY.

THE <SORT.STEP.DOWN DESIGNATOR> PROVIDES THE INFORMATION NECESSARY TO COMPARE TWO RECORDS. <RECORD1> AND <RECORD 2> ARE, RESPECTIVELY, THE FIRST AND SECOND RECORDS WHICH ARE TO BE COMPARED. THE <KEY TABLE ADDRESS> SPECIFIES THE SORT KEY USED IN THE COMPARISON.

SORT.UNBLOCK

<SORT.UNBLOCK DESIGNATOR> ::= SORT.UNBLOCK (<MINI FIB ADDRESS>, <LENGTH>,<SOURCE>,<DESTINATION>)

<MINI FIB ADDRESS> ::= <ADDRESS GENERATOR>

<LENGTH> ::= <EXPRESSION>

<SOURCE> ::= <EXPRESSION>

<DESTINATION> ::= <EXPRESSION>

FOR USE BY SORT ONLY.

THE <SORT.UNBLOCK DESIGNATOR> MOVES A RECORD TO OR FROM A BUFFER, UPDATING THE BUFFER POINTER AND BLOCK COUNT. IT NORMALLY RETURNS A ZERO. WHEN THE BLOCK COUNT GOES TO ZERO, IT RESTORES THE ORIGINAL BUFFER POINTER AND BLOCK COUNT, AND RETURNS A 1, SIGNALLING THE NEED FOR AN I/O.

A BIT ON THE MINI-FIB SIGNALS SORT.UNBLOCK TO CREATE SORT TAGS. FOR THIS FUNCTION, IT USES THE SORT KEY TABLE AND SELECTS ONLY THE KEY INFORMATION TO MOVE FROM THE BUFFER; A VALUE IN THE MINI-FIB REPRESENTS THE LENGTH OF THE RECEIVING FIELD.

SORT.SEARCH

<SORT.SEARCH DESIGNATOR> ::= SORT.SEARCH (<TABLE ADDRESS>,<LIMIT>)

<TABLE ADDRESS> ::= <ADDRESS GENERATOR>

<LIMIT> ::= <EXPRESSION>

FOR USE BY SORT ONLY.

THE <SORT SEARCH DESIGNATOR> PROVIDES THE INFORMATION TO EVALUATE A RECORD FOR SORTING PURPOSES. THE <TABLE ADDRESS> CONTAINS THE ADDRESS, IN AN ARRAY OF RECORDS, OF THE FIRST RECORD TO BE EXAMINED AND THE CONDITION UNDER WHICH RECORDS WILL BE SELECTED.

THE <LIMIT> SPECIFIES THE LAST RECORD TO BE EXAMINED.

PARITY.ADDRESS

<PARITY.ADDRESS DESIGNATOR> ::= PARITY.ADDRESS

FOR MCP USE ONLY.

THE <PARITY.ADDRESS DESIGNATOR> RETURNS A 24-BIT VALUE WHICH IS THE ADDRESS OF THE FIRST PARITY ERROR ENCOUNTERED IN S-MEMORY. IF NO PARITY ERROR IS FOUND, @FFFFFF@ IS RETURNED.

DYNAMIC MEMORY BASE

<DYNAMIC MEMORY BASE DESIGNATOR> ::= DYNAMIC.MEMORY.BASE

THE <DYNAMIC MEMORY BASE DESIGNATOR> RETURNS A 24-BIT VALUE WHICH IS THE BASE RELATIVE ADDRESS OF THE PROGRAM-S DYNAMIC MEMORY. REFER TO THE SDL S-LANGUAGE DOCUMENTATION FOR DISCUSSION OF THE USE OF DYNAMIC MEMORY.

HASH CODE

<HASH CODE DESIGNATOR>::= HASH.CODE (<TOKEN>)

<TOKEN>::= <EXPRESSION>

THE HASH.CODE WILL LEAVE ON THE EVALUATION STACK A DESCRIPTOR OF TYPE BIT AND LENGTH 24. THE VALUE WILL BE A HASH CODE BASED ON THE FIRST 15 (OR LESS) CHARACTERS OF <TOKEN> AND ON THE LENGTH OF <TOKEN>. TO BE EFFECTIVE, THE VALUE GENERATED BY HASH.CODE MUST BE USED MODULO A PRIME NUMBER (WHICH IS USUALLY THE HASH TABLE SIZE).

NEXT TOKEN

<NEXT TOKEN DESIGNATOR>::= NEXT.TOKEN (<FIRST CHARACTER>, <SEPARATOR>, <NUMERIC-TO-ALPHA INDICATOR>, <RESULT>)

<FIRST CHARACTER>::= <IDENTIFIER>

<SEPARATOR>::= <CHARACTER STRING>

<NUMERIC-TO-ALPHA INDICATOR>::= SET
/ RESET

THE <FIRST CHARACTER> IS A SIMPLE IDENTIFIER WHICH DESCRIBES THE FIRST CHARACTER TO BE EXAMINED. THIS WILL USUALLY BE THE FIRST CHARACTER OF THE TOKEN. THE <SEPARATOR> IS THE TOKEN SEPARATOR: "." FOR SDL, "-" FOR COBOL, ETC. IT MUST BE A SINGLE CHARACTER; IF NONE IS NEEDED, USE "A". <NUMERIC- TO-ALPHA INDICATOR> IS SET IF SYMBOLS SUCH AS 235AB ARE ALLOWED. IT IS RESET OTHERWISE.

NEXT.TOKEN WILL LEAVE ON THE TOP OF THE EVALUATION STACK THE DESCRIPTOR OF THE NEXT TOKEN. THIS TOKEN WILL BE AN IDENTIFIER, A NUMBER, OR A SPECIAL CHARACTER. THE DESCRIPTOR OF <RESULT> WILL ALSO BE REPLACED BY THIS DESCRIPTOR. THE ADDRESS FIELD OF <FIRST CHARACTER> WILL BE CHANGED TO POINT TO THE CHARACTER FOLLOWING THIS TOKEN. NEXT.TOKEN ASSUMES THAT <FIRST CHARACTER> DESCRIBES A NON-BLANK CHARACTER.

DELIMITED TOKEN

<DELIMITED TOKEN DESIGNATOR>::= DELIMITED.TOKEN (<FIRST CHARACTER>, <DELIMITERS>, <RESULT>)

<FIRST CHARACTER>::= <IDENTIFIER>

<DELIMITERS>::= <CHARACTER STRING>
/ <BIT STRING>

<RESULT>::= <IDENTIFIER>

THE <FIRST CHARACTER> IS A SIMPLE IDENTIFIER WHICH DESCRIBES THE FIRST CHARACTER TO BE EXAMINED. <DELIMITERS> WILL GENERATE 16 BITS OF INFORMATION, EACH OF THE 8-BIT BYTES BEING USED AS A DELIMITER. FOR SDL, <DELIMITERS> WILL BE """%"; FOR COBOL, •7F03• (QUOTE FOLLOWED BY ETX).

DELIMITED.TOKEN WILL LEAVE ON THE TOP OF THE EVALUATION STACK THE DESCRIPTOR OF THE STRING OF CHARACTERS FROM (AND INCLUDING) <FIRST CHARACTER> UP TO (BUT NOT INCLUDING) WHICHEVER DELIMITER WAS FOUND. THE DESCRIPTOR OF <RESULT> WILL BE REPLACED BY THIS DESCRIPTOR. THE ADDRESS FIELD OF <FIRST CHARACTER> WILL BE CHANGED TO POINT TO THE DELIMITER WHICH STOPPED THE SCAN.

EVALUATION STACK TOP

<EVALUATION STACK TOP DESIGNATOR>::= EVALUATION.STACK.TOP

THIS FUNCTION LEAVES ON THE TOP OF THE EVALUATION STACK A 24-BIT, SELF-RELATIVE VALUE OF TYPE BIT WHICH IS THE BASE-RELATIVE ADDRESS OF THE TOP OF THE EVALUATION STACK (BEFORE EXECUTION OF THIS FUNCTION).

CONTROL STACK BITS

<CONTROL STACK BITS DESIGNATOR>::= CONTROL.STACK.BITS

THIS FUNCTION LEAVES ON THE TOP OF THE EVALUATION STACK A 24-BIT, SELF-RELATIVE VALUE OF TYPE BIT WHICH IS THE NUMBER OF BITS LEFT IN THE CONTROL STACK UNTIL OVERFLOW.

NAME STACK TOP

<NAME STACK TOP DESIGNATOR>::= NAME.STACK.TOP

THIS FUNCTION LEAVES ON THE TOP OF THE EVALUATION STACK A 24-BIT, SELF-RELATIVE VALUE OF TYPE BIT WHICH IS THE BASE-RELATIVE ADDRESS OF THE TOP OF THE NAME STACK.

DISPLAY BASE

<DISPLAY BASE DESIGNATOR>::= DISPLAY.BASE

THIS FUNCTION LEAVES ON THE TOP OF THE EVALUATION STACK A 24-BIT, SELF-RELATIVE VALUE OF TYPE BIT WHICH IS THE BASE-RELATIVE ADDRESS OF THE BASE OF THE DISPLAY.

CONSOLE SWITCHES

<CONSOLE SWITCHES DESIGNATOR>::= CONSOLE.SWITCHES

NOTE: THIS FUNCTION HAS MEANING ONLY ON THE B1726 (OR LARGER). IT LEAVES ON THE TOP OF THE EVALUATION STACK A 24-BIT, SELF-RELATIVE VALUE OF THE 24 CONSOLE SWITCHES.

SEARCH SERIAL LIST

<SEARCH SERIAL LIST DESIGNATOR>::= SEARCH.SERIAL.LIST (<SSL COMPARE VALUE>, <SSL COMPARE TYPE>, <SSL COMPARE FIELD>, <SSL FIRST ITEM>, <SSL TABLE LENGTH>, <SSL RESULT VARIABLE>)

<SSL COMPARE VALUE>::= <EXPRESSION>

<SSL COMPARE TYPE>::= </≤/=/≠/≥/>

<SSL COMPARE FIELD>::= <EXPRESSION>

<SSL FIRST ITEM>::= <EXPRESSION>

<SSL TABLE LENGTH>::= <EXPRESSION>

<SSL RESULT VARIABLE>::= <ADDRESS GENERATOR>

SEARCH.SERIAL.LIST SEARCHES A SERIAL LIST OF ITEMS BEGINNING WITH THE ITEM DESCRIBED BY <SSL FIRST ITEM>. <SSL COMPARE VALUE> IS COMPARED (AS SPECIFIED BY <SSL COMPARE TYPE>) AGAINST THE FIELD OF THE ITEM DESCRIBED BY <SSL COMPARE FIELD> (<SSL COMPARE FIELD> IS A TEMPLATE) UNTIL A MATCH HAS BEEN FOUND, OR UNTIL <SSL TABLE LENGTH> NUMBER OF BITS HAS BEEN SEARCHED.

IF THE SEARCH SUCCEEDS, THE BASE RELATIVE ADDRESS OF THE ITEM CONTAINING THE "SUCCESSFUL" <SSL COMPARE FIELD> IS STORED IN <SSL RESULT VARIABLE> AND A @(1)1@ IS RETURNED.

IF THE SEARCH FAILS, THEN THE END ADDRESS OF THE TABLE IF STORED IN <SSL RESULT VARIABLE> AND A @(1)0@ IS RETURNED.

SPO INPUT PRESENT

<SPO INPUT PRESENT DESIGNATOR>::= SPO.INPUT.PRESENT

A "SPECIAL", SPO.INPUT.PRESENT, HAS BEEN ADDED TO ALLOW THE PRESENCE OF SPO INPUT TO BE TESTED BEFORE HAVING TO PERFORM AN ACCEPT TO THE MCP.

SEARCH.SDL.STACKS

<SEARCH.SDL.STACKS DESIGNATOR>::= SEARCH.SDL.STACKS
(<STACK BASE>, <STACK TOP>,
<COMPARE BASE>, <COMPARE TOP>)

<STACK BASE>::= <EXPRESSION>

<STACK TOP>::= <EXPRESSION>

<COMPARE BASE>::= <EXPRESSION>

<COMPARE TOP>::= <EXPRESSION>

THE FOUR PARAMETERS ARE EXPECTED TO GENERATE VALUES WHICH ARE BASE-RELATIVE ADDRESSES OF THE BASE AND TOP OF A STACK OF SDL

DESCRIPTORS AND OF AN ADDRESS RANGE, RESPECTIVELY. THE STACK WILL BE SEARCHED FOR A NON-ARRAY, NON-SELF-RELATIVE SDL DESCRIPTOR WHOSE ADDRESS IS WITHIN THE GIVEN RANGE. IF THE SEARCH IS SUCCESSFUL @(1)1@ WILL BE RETURNED; OTHERWISE, @(1)0@ WILL BE RETURNED.

EXECUTE

<EXECUTE DESIGNATOR>::= EXECUTE (<EXPRESSION LIST>)

<EXPRESSION LIST>::= <EXPRESSION>
/ <EXPRESSION LIST>, <EXPRESSION>

THE VALUE OF THE LAST EXPRESSION IS EXPECTED TO BE AN OPCODE WHICH WILL THEN BE EXECUTED BY THE INTERPRETER. EXECUTE MAY BE USED AS A STATEMENT AS WELL AS A <VALUE GENERATING FUNCTION DESIGNATOR>.

ADDRESS AND VALUE PARAMETERS

ACTUAL PARAMETERS MAY BE PASSED TO A PROCEDURE EITHER BY NAME (WHICH PASSES THE ADDRESS OF THE ACTUAL PARAMETER) OR BY VALUE (WHICH PASSES A DUPLICATE COPY OF THE ACTUAL PARAMETER).

IF AN <ACTUAL PARAMETER> (SEE "VALUE VARIABLES" AND "EXECUTE PROCEDURE STATEMENT") IS PASSED BY ADDRESS, THEN ANY CHANGE TO THE CORRESPONDING <FORMAL PARAMETER> IN THE PROCEDURE WILL RESULT IN A CHANGE TO THE ORIGINAL VALUE OF THE <ACTUAL PARAMETER>.

IF A PARAMETER IS PASSED BY VALUE, THEN ONLY THE DUPLICATE COPY OF THE <ACTUAL PARAMETER> CAN BE CHANGED. THE ORIGINAL VALUE REMAINS UNALTERED, AND THE DUPLICATE COPY IS ERASED WHEN THE PROCEDURE IS EXITED.

AN <ACTUAL PARAMETER> MAY BE ANY EXPRESSION OR AN <ARRAY IDENTIFIER>. SDL HAS SPECIFIED THAT ARRAY IDENTIFIERS MAY ONLY BE PASSED BY ADDRESS. AN ARRAY ELEMENT, HOWEVER, MAY BE PASSED EITHER BY ADDRESS OR BY VALUE.

EXPRESSIONS MAY BE DIVIDED INTO TWO GROUPS:

1. THOSE WHICH MAY BE PASSED EITHER BY ADDRESS OR BY VALUE, AND

2. THOSE WHICH MAY ONLY BE PASSED BY VALUE.

AN <ADDRESS GENERATOR> IS PASSED BY ADDRESS UNLESS IT IS ENCLOSED WITHIN PARENTHESES, OR UNLESS THE FORMAL PARAMETER TO WHICH IT CORRESPONDS HAS BEEN DECLARED AS "FORMAL.VALUE". IN THESE TWO CASES <ADDRESS GENERATOR>S WILL BE LOADED BY VALUE. ALL OTHER EXPRESSIONS ARE LOADED BY VALUE ONLY.

EXAMPLES OF PARAMETERS PASSED BY ADDRESS:

```
P(BUMP X, A)
P(B(BUMP M), SUBBIT(X,5))
P(NEXT.ITEM(B), A:←C+D)
```

EXAMPLES OF PARAMETERS PASSED BY VALUE:

```
P((BUMP X), (A), 3)
P((B(BUMP M)), A+B)
P(SWAP(A,0), (SUBSTR(A,5,3)))
```

I/O CONTROL STATEMENTS

```
<I/O CONTROL STATEMENT> ::=  <OPEN STATEMENT>;
                           / <CLOSE STATEMENT>;
                           / <READ STATEMENT>
                           / <WRITE STATEMENT>
                           / <SEEK STATEMENT>;
                           / <ACCEPT STATEMENT>;
                           / <DISPLAY STATEMENT>;
                           / <SPACE STATEMENT>
                           / <SKIP STATEMENT>;
```

EACH FILE IS NUMBERED SEQUENTIALLY, BEGINNING WITH ZERO. THIS NUMBER IS THE <FILE NUMBER> AND WILL EVENTUALLY BE USED AS AN INDEX INTO THE FIB DICTIONARY. THE FILE DECLARATION WILL BE USED TO CONSTRUCT AN FPB IN THE CODE FILE.

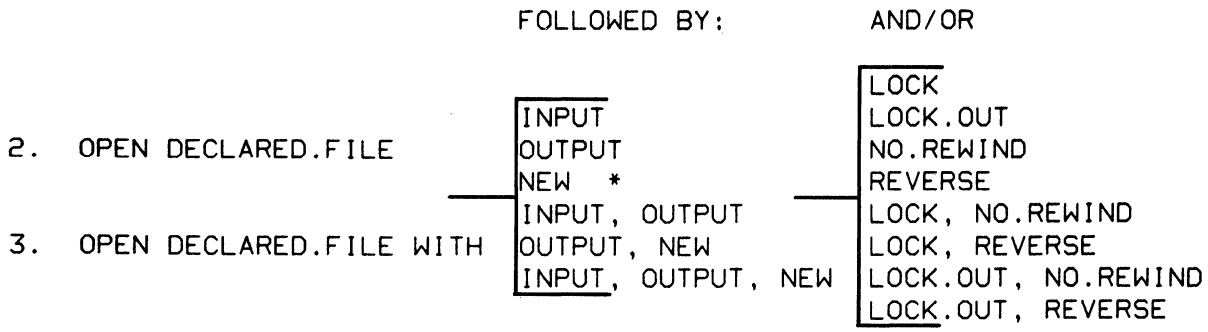
OPEN STATEMENT

```
<OPEN STATEMENT>::=          <OPEN PART>
                           / <OPEN PART>; <FILE MISSING PART>
                           / <OPEN PART>; <FILE LOCKED PART>
                           / <OPEN PART>; <FILE MISSING PART>
                             <FILE LOCKED PART>

<OPEN PART>::=               OPEN <FILE DESIGNATOR>
                             <OPEN ATTRIBUTE PART>

<FILE DESIGNATOR>::=         <FILE IDENTIFIER>
                           / <SWITCH FILE IDENTIFIER> (<EXPRESSION>)

<OPEN ATTRIBUTE PART> ::=    <EMPTY>
                           / <OPEN ATTRIBUTE LIST>
                           / WITH <OPEN ATTRIBUTE LIST>

<OPEN ATTRIBUTE LIST> ::=    <OPEN ATTRIBUTE>
                           / <OPEN ATTRIBUTE> <ATTRIBUTE SEPARATOR>
                             <OPEN ATTRIBUTE LIST>

<ATTRIBUTE SEPARATOR>::=     , / <SLASH> / <EMPTY>

<OPEN ATTRIBUTE> ::=         <INPUT-OUTPUT MODE>
                           / <LOCK MODE>
                           / <OPEN ACTION MODE>
                           / <CODE FILE MODE>
                           / <MFCU MODE>

<INPUT-OUTPUT MODE> ::=      INPUT / OUTPUT / NEW

<LOCK MODE> ::=              LOCK / LOCK.OUT

<OPEN ACTION MODE> ::=       NO.REWIND / REVERSE

<CODE FILE MODE> ::=         CODE.FILE

<MFCU MODE>::=               PUNCH / PRINT /
                             INTERPRET / STACKERS

<FILE MISSING PART>::=       ON FILE.MISSING <EXECUTABLE STATEMENT>

<FILE LOCKED PART>::=        ON FILE.LOCKED <EXECUTABLE STATEMENT>
```

FORMAT OPTIONS:

1. OPEN DECLARED.FILE;

IF NO ATTRIBUTES ARE SPECIFIED, "INPUT" IS ASSUMED.



* "NEW" ALONE ASSUMES "OUTPUT, NEW".

NOTE: THE COMBINATION "INPUT, NEW" RESULTS IN A SYNTAX ERROR.

NOTE: "CODE.FILE" IS TO BE USED ONLY BY COMPILERS.

IF THE <OPEN ATTRIBUTE>S HAVE BEEN EXPLICITLY OR IMPLICITLY INCLUDED IN THE FILE DECLARATION, THEN THE FILE NEED NOT BE EXPLICITLY OPENED HERE.

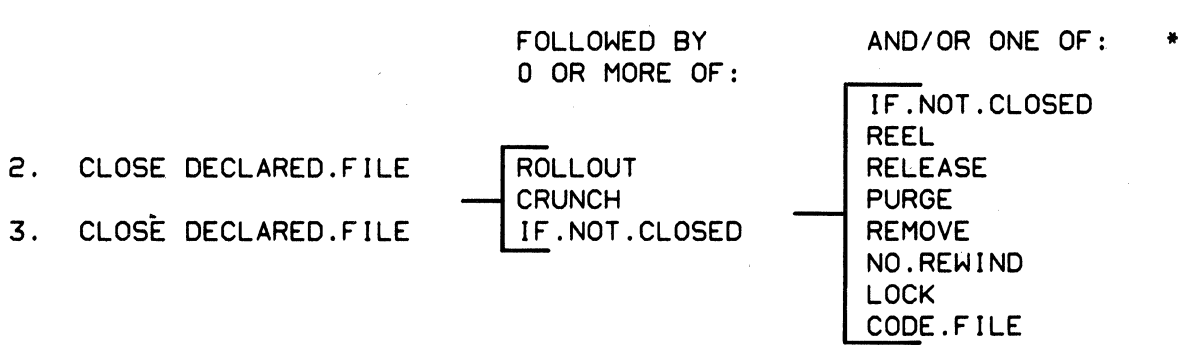
CLOSE STATEMENT

```
<CLOSE STATEMENT>::=          CLOSE <FILE DESIGNATOR>
                              <CLOSE ATTRIBUTE PART>

<FILE DESIGNATOR>::=          <FILE IDENTIFIER>
                            / <SWITCH FILE IDENTIFIER> (<EXPRESSION>)

<CLOSE ATTRIBUTE PART> ::=    <EMPTY>
                            / <CLOSE ATTRIBUTE LIST>
                            / WITH <CLOSE ATTRIBUTE LIST>

<CLOSE ATTRIBUTE LIST> ::=    <CLOSE ATTRIBUTE>
                            / <CLOSE ATTRIBUTE> <ATTRIBUTE SEPARATOR>
                              <CLOSE ATTRIBUTE LIST>

<ATTRIBUTE SEPARATOR>::=      , / <SLASH> / <EMPTY>

<CLOSE ATTRIBUTE> ::=         <CLOSE MODE>
                            / CRUNCH / ROLLOUT / IF.NOT.CLOSED

<CLOSE MODE> ::=              REEL / RELEASE / PURGE / REMOVE
                            / NO.REWIND / LOCK / CODE.FILE
```

FORMAT OPTIONS:

1. CLOSE DECLARED.FILE;

   THERE IS NO DEFAULT. IF "LOCK" IS SPECIFIED AS PART OF THE FILE ATTRIBUTES, THE FILE IS LOCKED. OTHERWISE THE FILE IS NOT LOCKED.



* IF MORE THAN ONE OPTION IS SPECIFIED, ONLY THE FINAL ONE IS USED BY THE COMPILER.

FILES NEED NOT BE EXPLICITLY CLOSED. HOWEVER, CLOSING A FILE WHEN FINISHED WITH IT WILL FREE MEMORY SPACE FOR OTHER USES.

NOTE: "CODE.FILE" IS TO BE USED ONLY BY COMPILERS. WHEN "CODE.FILE" IS USED, IT IS NOT NECESSARY TO USE "LOCK" OR "CRUNCH".

READ STATEMENT

```
<READ STATEMENT> ::=          <READ PART>;
                            / <READ PART>;<EOF PART>
                            / <READ PART>;<PARITY PART>
                            / <READ PART>; <EXCEPTION PART>

<READ PART> ::=               <READ SPECIFIER>
                            / <DISK READ SPECIFIER>
                            / <REMOTE READ SPECIFIER>
                            / <QUEUE READ SPECIFIER>

<READ SPECIFIER>::=           READ <FILE DESIGNATOR>
                              (<ADDRESS GENERATOR>)
                                        % SEE "ADDRESS VARIABLES"

<FILE DESIGNATOR> ::=         <FILE IDENTIFIER>
                            / <SWITCH FILE IDENTIFIER> (<EXPRESSION>)

<DISK READ SPECIFIER> ::=     READ <RECORD LOCK PART>
                              <FILE DESIGNATOR>
                              <RECORD ADDRESS PART>
                              (<ADDRESS GENERATOR>)

<RECORD LOCK PART> ::=        <EMPTY> / LOCK

<RECORD ADDRESS PART> ::=     <EMPTY>
                              [<RECORD ADDRESS>]

<RECORD ADDRESS> ::=          <EXPRESSION>

<REMOTE READ SPECIFIER>::=    READ <FILE DESIGNATOR>
                              <REMOTE KEY PART>
                              (<ADDRESS GENERATOR>)

<REMOTE KEY PART>::=          <EMPTY>
                            / [<REMOTE KEY>]

<REMOTE KEY>::=               <ADDRESS GENERATOR>

<QUEUE READ SPECIFIER>::=     READ <FILE DESIGNATOR>
                              <QUEUE FAMILY MEMBER PART>
                              (<ADDRESS GENERATOR>)

<QUEUE FAMILY
MEMBER PART>::=               <EMPTY>
                            / [<QUEUE FAMILY MEMBER>]

<QUEUE FAMILY MEMBER>::=      <EXPRESSION>
```

<EOF PART> ::= ON EOF <EXECUTABLE STATEMENT>

<EXCEPTION PART>::= ON EXCEPTION <EXECUTABLE STATEMENT>

THE <READ STATEMENT> PROVIDES THE NECESSARY INFORMATION TO READ A FILE: A FILE IDENTIFIER, RECORD ADDRESS, DATA INFORMATION, AND INSTRUCTIONS TO BE EXECUTED IF AN END-OF-FILE OR A PARITY ERROR IS DETECTED.

THE <READ STATEMENT> SEPARATES FILES INTO FOUR CATEGORIES: DISK FILES, REMOTE FILES, QUEUE FILES, AND ALL OTHERS (CARD, TAPE, PAPERTAPE, ETC.). THE USER HAS THE OPTION OF SPECIFYING "LOCK" FOR EXCLUSIVE USE OF THE DISK FILE RECORD. IF THE FILE ATTRIBUTES INDICATE A RANDOM DISK FILE, THE USER MAY SPECIFY <RECORD ADDRESS>. IN ALL OTHER CASES, HE NEED ONLY GIVE THE <FILE IDENTIFIER> AND <ADDRESS GENERATOR>.

IF THE FILE IS OF TYPE "REMOTE", AND THE "REMOTE.KEY" ATTRIBUTE IS SET THEN A <REMOTE KEY> MAY BE USED. (FOR THE FORMAT OF THIS, SEE THE DISCUSSION UNDER "REMOTE.KEY" IN THE FILE DECLARATION SECTION.) IF THE "REMOTE.KEY" ATTRIBUTE IS NOT SET, THEN A <REMOTE KEY> MAY NOT BE USED. AFTER PERFORMING THE READ, THE "REMOTE KEY" WILL HAVE BEEN STORED IN THE FIELD SPECIFIED AS THE <REMOTE KEY>.

IF THE FILE IS OF TYPE "QUEUE" AND IS A MULTI-QUEUE FAMILY, THEN A <QUEUE FAMILY MEMBER> MAY BE USED. THIS IS AN EXPRESSION WHOSE VALUE WILL SPECIFY WHICH MEMBER OF THE FAMILY TO READ FROM. IF THIS IS OMITTED, THEN THE OLDEST MESSAGE IN ALL OF THE QUEUES WILL BE READ.

THE <EXECUTABLE STATEMENT>S OF THE <EOF PART> AND <EXCEPTION PART> ARE CONSIDERED SUBORDINATE TO THE <READ STATEMENT>. THEREFORE, SEGMENTATION OF THESE STATEMENTS IS TEMPORARY (SEE "THE SEGMENT STATEMENT").

WRITE STATEMENT

```
<WRITE STATEMENT> ::=          <WRITE PART>;
                             / <WRITE PART>;<EOF PART>
                             / <WRITE PART>;<EXCEPTION PART>
                             / <WRITE PART>;<EOF PART> <EXCEPTION PART>

<WRITE PART> ::=               <WRITE SPECIFIER>
                             / <DISK WRITE SPECIFIER>
                             / <REMOTE WRITE SPECIFIER>
                             / <QUEUE WRITE SPECIFIER>

<WRITE SPECIFIER> ::=          WRITE <FILE DESIGNATOR>
                               <CARRIAGE CONTROL PART>
                               (<EXPRESSION>)
                             / WRITE <FILE IDENTIFIER>
                               <CARRIAGE CONTROL PART>

<FILE DESIGNATOR> ::=          <FILE IDENTIFIER>
                             / <SWITCH FILE IDENTIFIER> (<EXPRESSION>)

<CARRIAGE CONTROL PART> ::=    <EMPTY>
                             / <CARRIAGE CONTROL SPECIFIER>

<CARRIAGE CONTROL
SPECIFIER> ::=                 NO / SINGLE / DOUBLE / PAGE
                             / <SKIP-TO-CHANNEL> / NEXT

<SKIP-TO-CHANNEL> ::=          <CHANNEL NUMBER>

<CHANNEL NUMBER> ::=           1 / 2 / 3 / ... / 11 / 12

<DISK WRITE SPECIFIER> ::=     WRITE <RECORD LOCK PART>
                               <FILE DESIGNATOR>
                               <RECORD ADDRESS PART>
                               (<EXPRESSION>)

<EOF PART> ::=                 ON EOF <EXECUTABLE STATEMENT>

<EXCEPTION PART> ::=           ON EXCEPTION <EXECUTABLE STATEMENT>

<RECORD LOCK PART> ::=         <EMPTY> / LOCK

<RECORD ADDRESS PART> ::=      <EMPTY>
                             / [<RECORD ADDRESS>]

<RECORD ADDRESS> ::=           <EXPRESSION>

<REMOTE WRITE
```

```
SPECIFIER>::=                    WRITE <FILE DESIGNATOR>
                                 <REMOTE KEY PART>
                                 (<EXPRESSION>)

<REMOTE KEY PART>::=               <EMPTY>
                                 / [<REMOTE KEY>]

<REMOTE KEY>::=                  <ADDRESS GENERATOR>

<QUEUE WRITE
SPECIAL>::=                      WRITE <FILE DESIGNATOR>
                                 <QUEUE FAMILY MEMBER PART>
                                 (<ADDRESS GENERATOR>)

<FILE DESIGNATOR>::=             <FILE IDENTIFIER>
                                 <SWITCH FILE IDENTIFIER> (<EXPRESSION>)

<QUEUE FAMILY
MEMBER PART>::=                    <EMPTY>
                                 / [<QUEUE FAMILY MEMBER>]

<QUEUE FAMILY MEMBER>::=         <EXPRESSION>
```

THE <WRITE STATEMENT> PROVIDES THE NECESSARY INFORMATION TO WRITE A FILE. THE <WRITE STATEMENT> TREATS DISK FILES SEPARATELY FROM OTHER FILE TYPES BY ALLOWING THE USER THE OPTION OF LOCKING DISK FILE RECORDS, AND SPECIFYING <RECORD ADDRESS> ON HIS RANDOM DISK FILES. THE <CARRIAGE CONTROL PART> IS INTENDED FOR USE WITH A PRINTER FILE.

IF THE FILE IS OF TYPE "REMOTE", AND THE "REMOTE.KEY" ATTRIBUTE IS SET THEN A <REMOTE KEY> MAY BE USED. (FOR THE FORMAT OF THIS, SEE THE DISCUSSION UNDER "REMOTE.KEY" IN THE FILE DECLARATION SECTION.) IF THE "REMOTE.KEY" ATTRIBUTE IS NOT SET, THEN A <REMOTE KEY> MAY NOT BE USED. THE <REMOTE KEY> WILL SPECIFY THE TERMINAL TO WHICH THE WRITE IS TO BE PERFORMED.

IF THE FILE IS OF TYPE "QUEUE" AND IS A MULTI-QUEUE FAMILY, THEN A <QUEUE FAMILY MEMBER> MAY BE USED. THIS IS AN EXPRESSION WHOSE VALUE WILL SPECIFY WHICH MEMBER OF THE FAMILY TO WRITE TO.

THE <EXECUTABLE STATEMENT>S OF THE <EOF PART> AND <EXCEPTION PART> ARE CONSIDERED SUBORDINATE TO THE <WRITE STATEMENT>. THEREFORE, SEGMENTATION OF THESE STATEMENTS IS TEMPORARY (SEE "THE SEGMENT STATEMENT").

IF THE <END-OF-PAGE PART> IS SET IN THE FILE ATTRIBUTES, THEN

WHEN END-OF-PAGE IS DETECTED ON A PRINTER FILE, THE <EOF PART> WILL BE EXECUTED. THIS FACILITATES, FOR EXAMPLE, PRINTING TOTALS AND/OR HEADINGS WITHOUT KEEPING A LINE COUNTER.

EXAMPLE:

```
WRITE PRINTOUT SINGLE (PRINT.LINE);
     ON EOF DO;
               WRITE PRINTOUT; % SKIP A LINE:
               WRITE PRINTOUT PAGE (TOTALS);
               WRITE PRINTOUT DOUBLE (HEADER);
          END;
```

SEEK STATEMENT

```
<SEEK STATEMENT> ::=          SEEK <RECORD LOCK PART>
                              <FILE DESIGNATOR>
                              [<RECORD ADDRESS>]

<RECORD LOCK PART> ::=        <EMPTY> / LOCK

<FILE DESIGNATOR>::=          <FILE IDENTIFIER>
                            / <SWITCH FILE IDENTIFIER> (<EXPRESSION>)

<RECORD ADDRESS> ::=          <EXPRESSION>
```

THE <SEEK STATEMENT> CALLS UP A RECORD FROM A RANDOM DISK FILE IN PREPARATION FOR A READ ON THAT RECORD. THIS STATEMENT SHOULD ONLY BE USED WITH DISK FILES THAT ARE BEING READ USING A RANDOM ACCESS TECHNIQUE.

A <SEEK STATEMENT> PERFORMED IMMEDIATELY PRIOR TO A <READ STATEMENT> IS LESS EFFECTIVE THAN MERELY READING THE RECORD.

ACCEPT STATEMENT

```
<ACCEPT STATEMENT> ::=        ACCEPT <ADDRESS GENERATOR>
                              <END-OF-TEXT SPECIFIER>

<END-OF-TEXT
SPECIFIER> ::=                <EMPTY>
                            / , END.OF.TEXT
```

THE <ACCEPT STATEMENT> CAUSES THE EXECUTION OF A PROGRAM TO HALT UNTIL THE APPROPRIATE INFORMATION IS ENTERED VIA THE SPO BY THE OPERATOR. THE MESSAGE KEYED IN WILL BE READ INTO THE AREA SPECIFIED BY THE <ADDRESS GENERATOR> FOLLOWING THE RESERVED WORD "ACCEPT".

IF ", END.OF.TEXT" IS SPECIFIED, THE SYSTEM WILL INCLUDE THE "END OF MESSAGE" CHARACTER WITH THE MESSAGE. OTHERWISE, THE "END OF MESSAGE" CHARACTER IS OMITTED, AND THE REST OF THE SPECIFIED AREA IS FILLED WITH BLANKS.

SEE "ADDRESS VARIABLES" FOR THE SYNTAX OF THE <ADDRESS GENERATOR>.

DISPLAY STATEMENT

```
<DISPLAY STATEMENT> ::=      DISPLAY <EXPRESSION>
                             <CRUNCH SPECIFIER>

<CRUNCH SPECIFIER> ::=       <EMPTY>
                           / , CRUNCHED
```

THE <DISPLAY STATEMENT> PRINTS AN OUTPUT MESSAGE ON THE SPO. AS NOTED, THE <CRUNCH SPECIFIER> IS OPTIONAL. IF ", CRUNCHED" IS SPECIFIED, THE SYSTEM WILL DELETE TRAILING BLANKS AND ALL OCCURRENCES OF EMBEDDED BLANKS. ONE BLANK IS SUBSTITUTED FOR EACH OCCURRENCE OF MULTIPLE EMBEDDED BLANKS.

SPACE STATEMENT

```
<SPACE STATEMENT> ::=          <SPACE PART>;
                             / <SPACE PART>; <EOF PART>
                             / <SPACE PART>; <EXCEPTION PART>
                             / <SPACE PART>; <EOF PART> <EXCEPTION PART>

<SPACE PART> ::=               SPACE <FILE DESIGNATOR>
                               <SPACING SPECIFIER>

<FILE DESIGNATOR> ::=          <FILE IDENTIFIER>
                             / <SWITCH FILE IDENTIFIER> (<EXPRESSION>)

<SPACING SPECIFIER> ::=        <EXPRESSION> / TO <EXPRESSION>

<EOF PART> ::=                 ON EOF <EXECUTABLE STATEMENT>

<EXCEPTION PART>::=            ON EXCEPTION <EXECUTABLE STATEMENT>
```

THE <SPACE STATEMENT> ALLOWS THE USER TO SKIP OVER CERTAIN RECORDS IN A SEQUENTIAL FILE.

THE <SPACING SPECIFIER> MAY TAKE TWO FORMS. AN <EXPRESSION> ALONE WILL INDICATE THE NUMBER OF RECORDS TO BE SPACED. IT MAY BE A NEGATIVE NUMBER INDICATING REVERSE SPACING. "TO <EXPRESSION>" WILL ALWAYS BE A POSITIVE NUMBER AND INDICATES THE NUMBER OF THE RECORD TO SPACE TO.

THE <EXECUTABLE STATEMENT>S OF THE <EOF PART> AND <EXCEPTION PART> ARE CONSIDERED SUBORDINATE TO THE <SPACE STATEMENT>. THEREFORE, SEGMENTATION OF THESE STATEMENTS IS TEMPORARY (SEE "THE SEGMENT STATEMENT").

SKIP STATEMENT

<SKIP STATEMENT> ::= SKIP <FILE IDENTIFIER> TO <CHANNEL NUMBER>

<FILE DESIGNATOR> ::= <FILE IDENTIFIER>
/ <SWITCH FILE IDENTIFIER> (<EXPRESSION>)

<SKIP STATEMENT> ::= SKIP <FILE DESIGNATOR> TO <CHANNEL NUMBER>

<CHANNEL NUMBER> ::= 1 / 2 / 3 / ... /11 / 12

THE <SKIP STATEMENT> CAUSES THE LINE PRINTER TO SKIP TO A SPECIFIED CHANNEL NUMBER ON ITS CARRIAGE TAPE. THE CHANNEL NUMBERS CONTROL THE VERTICAL SPACING OF DATA ON A PRINTED PAGE.

EXECUTABLE STATEMENTS

```
<EXECUTABLE STATEMENT
LIST> ::=                       <EXECUTABLE STATEMENT>
                              / <EXECUTABLE STATEMENT>
                                <EXECUTABLE STATEMENT LIST>

<EXECUTABLE STATEMENT> ::=      <DO GROUP>;
                              / <GROUP TERMINATION STATEMENT>;
                              / <IF STATEMENT>
                              / <CASE STATEMENT>;
                              / <ASSIGNMENT STATEMENT>;
                              / <EXECUTE-PROCEDURE STATEMENT>;
                              / <EXECUTE-FUNCTION STATEMENT>;
                              / <I/O CONTROL STATEMENT>
                              / <MODIFY INSTRUCTION>;
                              / <NULL STATEMENT>
                              / <FILE ATTRIBUTE STATEMENT>;
                              / <STOP STATEMENT>;
                              / <ZIP STATEMENT>;
                              / <SEARCH STATEMENT>
                              / <ACCESS FILE HEADER STATEMENT>
                              / <SEND STATEMENT>
                              / <RECEIVE STATEMENT>
                              / <ARRAY PAGE TYPE STATEMENT>
                              / <WAIT STATEMENT>;
                              / <SEGMENT STATEMENT>
                                <EXECUTABLE STATEMENT>

<ASSIGNMENT STATEMENT> ::=      SEE "ASSIGNMENT STATEMENTS
                                AND EXPRESSIONS"

<I/O CONTROL STATEMENT> ::=     SEE "I/O CONTROL STATEMENTS"

<SEGMENT STATEMENT> ::=         SEE "THE SEGMENT STATEMENT"
```

DO GROUPS

```
<DO GROUP> ::=                  <GROUP HEAD>
                                <GROUP BODY>

<GROUP HEAD> ::=                <GROUP NAME>
                                <FOREVER PART>;

<GROUP NAME> ::=                DO
                              / DO <GROUP IDENTIFIER>

<FOREVER PART> ::=              <EMPTY>
                              / FOREVER

<GROUP IDENTIFIER> ::=          <IDENTIFIER>

<GROUP BODY> ::=                <EXECUTABLE STATEMENT LIST>
                                <GROUP ENDING>

<GROUP ENDING> ::=              END
                              / END <GROUP IDENTIFIER>
```

THE <DO GROUP> IS A COLLECTION OF <EXECUTABLE STATEMENT>S WHICH FUNCTIONS AS A ROUTINE. IT IS EXECUTED ONCE UNLESS "FOREVER" APPEARS AFTER THE <GROUP NAME>.

IF "FOREVER" IS PRESENT, THE <DO GROUP> WILL BE EXECUTED ITERATIVELY UNTIL A SPECIFIC CONDITION IS MET. ONLY A <GROUP TERMINATION STATEMENT> (UNDO) OR A <TYPED PROCEDURE RETURN STATEMENT> (RETURN) CAN GET THE PROGRAM OUT OF THIS LOOP. SEE THE FOLLOWING EXAMPLE:

```
DO THIS FOREVER;
   READ CARD (A); ON EOF UNDO;
   IF 55 GTR BUMP X
      THEN WRITE PRINTER (A);
      ELSE DO;
           X←1;
           WRITE PRINTER PAGE (A);
           END;
END THIS;
```

IF IT IS NECESSARY TO EXECUTE THE STATEMENTS IN A <DO GROUP> FROM DIFFERENT POINTS IN THE PROGRAM, MORE EFFICIENT CODE IS GENERATED BY MAKING THE BODY OF THE GROUP A PROCEDURE RATHER THAN BY REPEATING THE <DO GROUP>.

RESTRICTIONS:

1. IF A <GROUP IDENTIFIER> IS INCLUDED IN THE <GROUP NAME>, IT MUST ALSO APPEAR IN THE <GROUP ENDING>.

2. IF THE <GROUP NAME> DOES NOT INCLUDE AN IDENTIFIER, THE <GROUP ENDING> MUST NOT CONTAIN ONE.

3. "FOREVER" IS NOT A RESERVED WORD AND MAY APPEAR AS THE <GROUP IDENTIFIER>. "DO FOREVER;" IS CONSIDERED TO BE THE <GROUP HEAD> OF AN UN-NAMED, ITERATIVE <DO GROUP>. "DO FOREVER FOREVER" IS A LEGAL HEADING FOR A NAMED, ITERATIVE GROUP.

4. NESTED <DO GROUP>S MAY NOT HAVE DUPLICATE IDENTIFIERS. IF THIS OCCURS, A WARNING MESSAGE WILL APPEAR ON THE PROGRAM LISTING.

5. <DO GROUP>S MAY BE NESTED 32 LEVELS DEEP. HOWEVER, A <GROUP TERMINATION STATEMENT> CAN UNDO ONLY A MAXIMUM OF 16 LEVELS.

GROUP TERMINATION STATEMENT

```
<GROUP TERMINATION
STATEMENT> ::=              UNDO
                          / UNDO (*)
                          / UNDO <GROUP IDENTIFIER>

<GROUP IDENTIFIER> ::=      <IDENTIFIER>
```

THE <GROUP TERMINATION STATEMENT> WILL CAUSE THE EXECUTION OF A <DO GROUP> TO CEASE, AND WILL TRANSFER CONTROL TO THE NEXT STATEMENT FOLLOWING THE <DO GROUP> WHICH HAS BEEN "UNDONE". THE STATEMENT MAY TAKE ONE OF THREE FORMS:

1. "UNDO" WILL TRANSFER CONTROL OUT OF THE <DO GROUP> WHICH CONTAINS THE STATEMENT.

2. IF <DO GROUP>S ARE NESTED, "UNDO (*)" TRANSFERS CONTROL OUT OF THE OUTERMOST <DO GROUP>.

3. "UNDO <GROUP IDENTIFIER>" TAKES CONTROL OUT OF THE <DO GROUP> SPECIFIED BY THE IDENTIFIER.

NOTE: UNDO (*) AND UNDO <IDENTIFIER> CAN UNDO A MAXIMUM OF 16 LEVELS.

EXAMPLE:

```
1.      DO ONE;
2.         DO TWO FOREVER;
3.            IF <EXPRESSION> THEN
4.               DO THREE;
5.                  CASE <EXPRESSION>;
6.                       UNDO; /* SAME AS UNDO THREE; */
7.                       UNDO TWO;
8.                       UNDO (*); /* SAME AS UNDO ONE; */
9.                  END CASE;
10.              END THREE;
11.        END TWO;
12.     END ONE;
```

EXECUTION OF LINE 6 TRANSFERS CONTROL TO LINE 11.
EXECUTION OF LINE 7 TRANSFERS CONTROL TO LINE 12.
EXECUTION OF LINE 8 TRANSFERS CONTROL TO THE NEXT STATEMENT FOLLOWING LINE 12.

IF STATEMENT

```
<IF STATEMENT> ::=          <IF CLAUSE>
                            <EXECUTABLE STATEMENT>
                          / <IF CLAUSE>
                            <EXECUTABLE STATEMENT>
                            ELSE <EXECUTABLE STATEMENT>

<IF CLAUSE> ::=             IF <EXPRESSION> THEN
```

THE <EXPRESSION> IS EVALUATED. IF THE LOW-ORDER BIT OF THE RESULT IS 1 (I.E., TRUE), THE STATEMENT FOLLOWING "THEN" IS EXECUTED. IF THE LOW-ORDER BIT IS 0 (I.E., FALSE), THE STATEMENT FOLLOWING "ELSE" (IF PRESENT) IS EXECUTED. IF THE RESULT OF THE<EXPRESSION> IS FALSE, AND THE "ELSE" PART IS OMITTED, CONTROL IS TRANSFERRED TO THE NEXT STATEMENT AFTER THE <IF STATEMENT>.

<IF STATEMENT>S MAY BE NESTED. THE OUTERMOST <IF CLAUSE> AND THE OUTERMOST "ELSE" ARE ON NESTING LEVEL 0. THE <EXECUTABLE STATEMENT>S FOLLOWING "THEN" AND "ELSE" ARE ON NESTING LEVEL 1. NESTING MAY BE NO DEEPER THAN 32 LEVELS.

WHEN USING NESTED <IF STATEMENT>S, THE USER MUST MAINTAIN CORRESPONDENCE BETWEEN THE DELIMITERS "THEN" AND "ELSE" ON EACH LEVEL. THE INNERMOST "ELSE" WILL ALWAYS BE ASSOCIATED WITH THE INNERMOST "THEN". FROM THIS POINT CONTINUES AN OUTWARD PROGRESSION (I.E., FROM HIGHEST NESTING LEVEL TO LOWEST) OF "THEN-ELSE" ASSOCIATION.

THUS, IF AN <IF STATEMENT> ON NESTING LEVEL N IS TO HAVE AN "ELSE" ASSOCIATED WITH IT, THEN EVERY <IF STATEMENT> ON A NESTING LEVEL GREATER THAN N MUST ALSO HAVE "ELSES" ASSOCIATED WITH THEM. IF THE USER WISHES TO EXECUTE NOTHING ON A FALSE CONDITION, THEN "ELSE" FOLLOWED BY A <NULL STATEMENT> MAY BE USED.

EXAMPLE:
LET E-1, E-2, E-3, AND E-4 BE <EXPRESSION>S
LET S-2, S-3, AND S-4 BE <EXECUTABLE STATEMENT>S

```
     IF E-1
     THEN IF E-2
          THEN IF E-3
               THEN IF E-4
                    THEN S-4;
                    ELSE;
               ELSE S-3;
          ELSE S-2;
```

CASE STATEMENT

```
<CASE STATEMENT> ::=        <CASE HEAD>
                            <CASE BODY>

<CASE HEAD> ::=             CASE <EXPRESSION>;

<CASE BODY> ::=             <EXECUTABLE STATEMENT LIST>
                            <CASE ENDING>

<CASE ENDING> ::=           END CASE
```

THE <EXPRESSION> SERVES AS AN INDEX INTO THE LIST OF <EXECUTABLE STATEMENT>S. THE STATEMENT SELECTED IS EXECUTED, AND THE OTHERS IGNORED. CONTROL IS THEN TRANSFERRED TO THE STATEMENT FOLLOWING THE <CASE ENDING> UNLESS, OF COURSE, THE STATEMENT IS A "RETURN" OR AN "UNDO".

IF THERE ARE N NUMBER OF STATEMENTS IN THE LIST, THEN THE RANGE OF THE VALUE OF THE <EXPRESSION> MAY BE FROM 0 THROUGH N-1.

THE STATEMENTS IN THE LIST MAY BE ANY LEGAL <EXECUTABLE STATEMENT> ALLOWED IN SDL. IF THE USER WISHES TO EXECUTE NOTHING IN A GIVEN CASE, THE <NULL STATEMENT> IS AN APPROPRIATE STATEMENT.

EXECUTE-PROCEDURE STATEMENT

```
<EXECUTE-PROCEDURE
STATEMENT> ::=                  <NON-TYPED PROCEDURE DESIGNATOR>

<NON-TYPED PROCEDURE
DESIGNATOR> ::=                 <NON-TYPED PROCEDURE IDENTIFIER>
                                <ACTUAL PARAMETER PART>

<NON-TYPED PROCEDURE
IDENTIFIER> ::=                 <IDENTIFIER>

<ACTUAL PARAMETER PART> ::=     <EMPTY>
                              / (<ACTUAL PARAMETER LIST>)

<ACTUAL PARAMETER LIST> ::=     <ACTUAL PARAMETER>
                              / <ACTUAL PARAMETER>,
                                <ACTUAL PARAMETER LIST>

<ACTUAL PARAMETER> ::=          <EXPRESSION>
                              / <ARRAY DESIGNATOR>

<ARRAY DESIGNATOR> ::=          <ARRAY IDENTIFIER>
```

A NON-TYPED PROCEDURE, I.E., A PROCEDURE WHICH PERFORMS A FUNCTION AND DOES NOT RETURN A VALUE, IS INVOKED THROUGH AN <EXECUTE-PROCEDURE STATEMENT>. THE NAME OF THE PROCEDURE IS FOLLOWED BY ITS PARAMETERS ENCLOSED IN PARENS. REFER TO THE SECTION "ADDRESS AND VALUE PARAMETERS" FOR INFORMATION CONCERNING PASSING PARAMETERS.

FOR A DESCRIPTION OF THE INVOCATION OF TYPED PROCEDURES, SEE "VALUE VARIABLES".

EXECUTE-FUNCTION STATEMENT

```
<EXECUTE-FUNCTION
STATEMENT> ::=                  <FUNCTION DESIGNATOR>

<FUNCTION DESIGNATOR> ::=       <DUMP DESIGNATOR>
                              / <TRACE DESIGNATOR>
                              / <SAVE DESIGNATOR>
                              / <RESTORE DESIGNATOR>
                              / <FETCH DESIGNATOE>
                              / <HALT DESIGNATOR>
                              / <REINSTATE DESIGNATOR>
                              / <ACCESS-FPB DESIGNATOR>
                              / <REVERSE STORE DESIGNATOR>
                              / <READ CASSETTE DESIGNATOR>
                              / <OVERLAY DESIGNATOR>
                              / <ACCESS OVERLAY DESIGNATOR>
                              / <ERROR COMMUNICATE DESIGNATOR>
                              / <SORT DESIGNATOR>
                              / <SORT.SWAP DESIGNATOR>
                              / <INITIALIZE.VECTOR DESIGNATOR>
                              / <THREAD.VECTOR DESIGNATOR>
                              / <ENABLE.INTERRUPTS DESIGNATOR>
                              / <DISABLE.INTERRUPTS DESIGNATOR>
                              / <ACCESS FILE INFORMATION
                                DESIGNATOR>
                              / <HARDWARE MONITOR DESIGNATOR>
                              / <SAVE STATE DESIGNATOR>
                              / <DEBLANK DESIGNATOR>
                              / <FREEZE-PROGRAM DESIGNATOR>
                              / <THAW-PROGRAM DESIGNATOR>
                              / <DUMP-FOR-ANALYSIS DESIGNATOR>
                              / <COMPILE-CARD-INFO DESIGNATOR>
                              / <COMMUNICATE DESIGNATOR>
```

DUMP

```
<DUMP DESIGNATOR> ::=           DUMP
```

THE SDL STACKS WILL BE DUMPED TO THE LINE PRINTER IN SOME REASONABLE FORMAT. PROGRAM EXECUTION WILL CONTINUE AFTER THE DUMP.

TRACE

<TRACE DESIGNATOR> ::= TRACE / NOTRACE / TRACE (<EXPRESSION>)

THE "TRACE" WILL CAUSE THE SDL INSTRUCTIONS OF THE NORMAL STATE PROGRAM TO BE TRACED ON THE LINE PRINTER. "NOTRACE" WILL TURN OFF THE TRACE.

"TRACE (<EXPRESSION>)" PROVIDES GREATER CONTROL OF THE TRACING TO BE DONE. THE LOW-ORDER 10 BITS ARE USED IN THE FOLLOWING WAY (NUMBERING OF THE 10 IS FROM LEFT TO RIGHT):

BIT	USE
0	TRACE ALL COMMANDS EXCEPT THOSE WHICH MODIFY DATA OR CHANGE THE PROGRAM POINTER STACK. NORMAL STATE ONLY.
1	TRACE COMMANDS WHICH MODIFY DATA ITEMS (E.G., CLR, SNDL, ETC.). NORMAL STATE ONLY.
2	TRACE COMMANDS WHICH CHANGE THE PROGRAM POINTER STACK (E.G., IFTH, CASE, EXIT, ETC,). NORMAL STATE ONLY.
3	NOT USED.
4-6	SAME AS 0-2, BUT FOR MCP. SEVERAL MCP ROUTINES (GETSPACE, FORGETSPACE, AND OTHERS) WILL NOT BE TRACED.
7-9	SAME AS 0-2, BUT WILL TRACE THOSE MCP ROUTINES NOT TRACED BY 4-6.

NOTE THAT "TRACE(@380@)" IS THE SAME AS "TRACE", WHILE "TRACE(0) IS THE SAME AS "NOTRACE".

SAVE

<SAVE DESIGNATOR> ::= SAVE (<EXPRESSION LIST>)

EACH OF THE <EXPRESSION>S, FROM LEFT TO RIGHT, WILL BE EVALUATED, AND THE VALUE OF EACH LEFT ON THE EVALUATION STACK (AND VALUE STACK, IF NECESSARY). SEE <RESTORE DESIGNATOR>.

RESTORE

<RESTORE DESIGNATOR> ::= RESTORE (<ADDRESS GENERATOR LIST>)

<ADDRESS GENERATOR LIST> ::= SEE "ADDRESS GENERATORS"

THE <RESTORE DESIGNATOR> ASSIGNS THE CURRENT VALUE ON THE TOP OF THE EVALUATION STACK TO EACH <ADDRESS GENERATOR>, FROM RIGHT TO LEFT, IN THE LIST. THIS OPERATOR IS USED IN CONJUNCTION WITH THE <SAVE DESIGNATOR>. SEE ABOVE.

EXAMPLE:

```
SAVE (A,B,C);
      .
      .
      .
RESTORE (A,B,C);
```

NOTE THAT "RESTORE (A,B,C)" IS THE SAME AS:

```
RESTORE (C);
RESTORE (B);
RESTORE (A);
```

FETCH

<FETCH DESIGNATOR> ::= FETCH (<I/O REFERENCE ADDRESS>, <PORT, CHANNEL, PRIORITY ADDRESS>, <RESULT DESCRIPTOR ADDRESS>)

<I/O REFERENCE ADDRESS> ::= <EXPRESSION>

<PORT,CHANNEL, PRIORITY ADDRESS> ::= <ADDRESS GENERATOR>

<ADDRESS GENERATOR> ::= SEE "ADDRESS GENERATORS"

<RESULT DESCRIPTOR ADDRESS> ::= <ADDRESS GENERATOR>

THE <FETCH DESIGNATOR> FETCHES THE RESULT OF AN I/O OPERATION. IF THERE IS A HIGH PRIORITY INTERRUPT, THEN THAT INTERRUPT WILL BE REPORTED. OTHERWISE, IF THE <I/O REFERENCE ADDRESS> IS

NON-ZERO, THEN ONLY AN INTERRUPT ON AN I/O DESCRIPTOR WITH THE REFERENCE ADDRESS THE SAME AS THE <I/O REFERENCE ADDRESS> WILL BE REPORTED. IF NOT FOUND, THE FIRST INTERRUPT ENCOUNTERED (IF ANY) WILL BE REPORTED. THE PORT (BIT 3), CHANNEL (BIT 3), AND PRIORITY (BIT 1) OF THE INTERRUPT ARE STORED FROM LEFT TO RIGHT IN THE LOW-ORDER 7 BITS OF <PORT, CHANNEL, PRIORITY ADDRESS> THE I/O RESULT DESCRIPTOR REFERENCE ADDRESS IS STORED IN THE LOW-ORDER 24 BITS OF THE <RESULT DESCRIPTOR ADDRESS>. IF THERE WERE NO INTERRUPTS, THEN THESE TWO FIELDS WILL BE ZERO.

HALT

<HALT DESIGNATOR> ::= HALT (<EXPRESSION>)

THE <HALT DESIGNATOR> CAUSES THE VALUE OF THE <EXPRESSION> TO BE MOVED TO THE M-MACHINE T-REGISTER. IF THE VALUE IS LONGER THAN 24 BITS, ONLY THE LOW-ORDER 24 BITS ARE MOVED. IF THE VALUE IS LESS THAN 24 BITS, THE VALUE IS RIGHT JUSTIFIED AND LEADING ZEROES ARE ADDED.

AFTER THE VALUE IS MOVED, AN M-MACHINE HALT IS EXECUTED.

EXAMPLES:

```
DECLARE X BIT(24);
HALT (X:=HEX.SEQUENCE.NUMBER);

DECLARE X BIT(24);
HALT (SUBBIT (HEX.SEQUENCE.NUMBER, 0, 24));
```

REINSTATE

<REINSTATE DESIGNATOR> ::= REINSTATE (<REINSTATED PROGRAM>)

<REINSTATED PROGRAM> ::= <ADDRESS GENERATOR>

THE <REINSTATED PROGRAM> IS ASSUMED TO DESCRIBE THE FIELD RS.COMMUNICATE.MSG.PTR OF RS.NUCLEUS OF THE PROGRAM TO BE REINSTATED (SEE DESCRIPTION OF THE RUN STRUCTURE IN "B1710 MCP REFERENCE MANUAL").

THE REINSTATING PROGRAM-S M-MACHINE STATE IS STORED IN THE APPROPRIATE PARTS OF ITS RS.NUCLEUS. THE ADDRESS OF THE

REINSTATING PROGRAM-S RS.NUCLEUS IS STORED IN THE REINSTATED PROGRAM-S RS.COMMUNICATE.LR.

THE PROGRAM WHOSE RS.COMMUNICATE.MSG.PTR IS DESCRIBED BY <REINSTATED PROGRAM> IS THEN REINSTATED.

ACCESS-FPB

<ACCESS-FPB DESIGNATOR> ::= <ACCESS-FPB IDENTIFIER> (<FILE SPECIFIER>, <SOURCE OR DESTINATION FIELD>)

<ACCESS-FPB IDENTIFIER> ::= READ.FPB / WRITE.FPB

<FILE SPECIFIER> ::= <FILE DESIGNATOR> / <FILE NUMBER>

<FILE DESIGNATOR> ::= <FILE IDENTIFIER> / <SWITCH FILE IDENTIFIER> (<EXPRESSION>)

<FILE NUMBER> ::= <EXPRESSION>

<SOURCE OR DESTINATION FIELD> ::= <ADDRESS GENERATOR>

<ADDRESS GENERATOR> ::= SEE "ADDRESS GENERATORS"

THE FILE PARAMETER BLOCK OF THE FILE INDICATED BY THE <FILE SPECIFIER> IS READ INTO, OR WRITTEN FROM THE <SOURCE OR DESTINATION FIELD>.

NOTE THAT THE <SOURCE OR DESTINATION FIELD> SHOULD BE 1440 BITS IN LENGTH.

REVERSE STORE

<REVERSE STORE DESIGNATOR> ::= REVERSE.STORE (<ADDRESS GENERATOR LIST>,<EXPRESSION>)

<ADDRESS GENERATOR LIST> ::= SEE "ADDRESS GENERATORS"

THE REVERSE.STORE OPERATION HAS THE EFFECT OF EVALUATING MULTIPLE STORE OPERATIONS FROM LEFT TO RIGHT INSTEAD OF FROM RIGHT TO LEFT. SEE "THE REPLACE OPERATORS".

FOR EXAMPLE:

```
        REVERSE.STORE (L,M,N,P,X+1);
HAS THE SAME EFFECT AS:
        L←M;
        M←N;
        N←P;
        P←X+1;
```

WITH THE REVERSE.STORE, HOWEVER, THE DESCRIPTOR FOR EACH <ADDRESS GENERATOR> IN THE LIST IS DETERMINED ONLY ONCE.

NOTE:

```
        REVERSE.STORE (L,M,N,P,X+1);
    IS NOT THE SAME AS
        L←M←N←P←X+1;
```

READ CASSETTE

```
<READ CASSETTE
DESIGNATOR>::=              READ.CASSETTE (<DESTINATION SPECIFIER,
                            <HASH.TOTAL SPECIFIER>, <RESULT SPECIFIER>

<DESTINATION SPECIFIER>::=  <ADDRESS GENERATOR>

<HASH.TOTAL SPECIFIER>::=   HASH.TOTAL
                          / NO.HASH.TOTAL

<RESULT SPECIFIER>::=       <ADDRESS GENERATOR>
```

THE <READ CASSETTE DESIGNATOR> CAUSES THE NUMBER OF BITS SPECIFIED BY THE <DESTINATION SPECIFIER> TO BE READ FROM THE CONSOLE CASSETTE TO THE ADDRESS SPECIFIED BY THAT <DESTINATION SPECIFIER>. THIS NUMBER OF BITS MUST BE EQUAL TO THE RECORD SIZE MINUS THE HASH-TOTAL SIZE (IF IT IS PRESENT) OF 16 BITS. THE <HASH.TOTAL SPECIFIER> INDICATES WHEATHER OR NOT A HASH-TOTAL IS EXPECTED AT THE END OF THE RECORD.

A VALUE OF 0 OR 1 WILL BE LEFT IN THE <RESULT SPECIFIER> INDICATING THAT THE HASH-TOTAL WAS INCORRECT OR CORRECT, RESPECTIVELY.

OVERLAY

<OVERLAY DESIGNATOR> ::= OVERLAY (<EXPRESSION>)

THE <EXPRESSION> WILL BE USED AS AN INDEX INTO THE INTERPRETER DICTIONARY BY THE INTERPRETER SWAPPER. THE INTERPRETER DICTIONARY ENTRY WILL SPECIFY THE ACTION TO BE TAKEN. SEE THE "B1710 MCP REFERENCE MANUAL".

ACCESS OVERLAY

<ACCESS OVERLAY DESIGNATOR> ::= <ACCESS OVERLAY IDENTIFIER>(<EXPRESSION>)

<ACCESS OVERLAY IDENTIFIER> ::= READ.OVERLAY / WRITE.OVERLAY

THE VALUE OF THE <EXPRESSION> IS ASSUMED TO BE A 76-BIT FIELD WITH THE FOLLOWING FORMAT FROM HIGH-ORDER TO LOW-ORDER:

BITS	CONTENTS
0-3	EU = 0 (NOT USED)
4-27	BASE RELATIVE BEGINNING ADDRESS
28-51	BASE RELATIVE ENDING ADDRESS
52-75	DISK ADDRESS (RELATIVE TO USER AREA)

THE AREA DESCRIBED BY THE BEGINNING AND ENDING ADDRESSES IS READ TO, OR WRITTEN FROM THE USER DISK AT THE (RELATIVE) DISK ADDRESS GIVEN.

ERROR COMMUNICATE

<ERROR COMMUNICATE DESIGNATOR> ::= ERROR.COMMUNICATE (<EXPRESSION>)

THE VALUE OF THE EXPRESSION SHOULD BE OF THE FOLLOWING FORM:

2 BITS	6 BITS	16 BITS	24 BITS
0	N	0	0

WHERE N IS THE ERROR NUMBER.

THE VALUE OF THE EXPRESSION WILL BE PUT ON THE EVALUATION STACK AS A DESCRIPTOR, AND AN MCP COMMUNICATE WILL BE PERFORMED.

SORT

<SORT DESIGNATOR> ::=

<SORT INFORMATION TABLE SPECIFIER> ::= <ADDRESS GENERATOR>

<ADDRESS GENERATOR> ::= SEE "ADDRESS GENERATORS"

<SORT KEY TABLE SPECIFIER> ::= <ADDRESS GENERATOR>

<INPUT FILE DESIGNATOR ::= <FILE DESIGNATOR>

<OUTPUT FILE DESIGNATOR> ::= <FILE DESIGNATOR>

<FILE DESIGNATOR>::= <FILE IDENTIFIER>
/ <SWITCH FILE IDENTIFIER> (<EXPRESSION>)

THE <SORT DESIGNATOR> IS A COMMUNICATE WHICH REQUESTS THE TRANSFER OF RECORDS FROM THE INPUT FILE TO THE OUTPUT FILE ACCORDING TO THE SORT KEY TABLE. THE SORT INFORMATION TABLE INCLUDES CODES FOR SORT TYPE, HARDWARE AVAILABLE, AND OTHER OPTIONS.

FOR FORMATTING SPECIFICATIONS OF THE SORT INFORMATION TABLE, REFER TO SORT DOCUMENTATION.

SORT.SWAP

<SORT.SWAP DESIGNATOR> ::= SORT.SWAP (<RECORD 1>,<RECORD 2>)

<RECORD 1> ::= <ADDRESS GENERATOR>

<RECORD 2> ::= <ADDRESS GENERATOR>

WHILE THE <SORT SWAP DESIGNATOR> IS INTENDED TO BE USED BY THE SORT, ITS APPLICATION IS SUCH THAT IT MAY BE GENERALLY USEFUL.

THIS DESIGNATOR ALLOWS THE USER TO "SWAP" OR EXCHANGE TWO RECORDS IN MEMORY WITHOUT ALLOCATING A THIRD AREA FOR STORING ONE OF THE RECORDS.

SPECIFICALLY, THE RECORD POINTED TO BY <RECORD 1> IS EXCHANGED WITH THE RECORD POINTED TO BY <RECORD 2>.

NOTE: THE INTERPRETER BEING USED MUST CONTAIN THE SORT.SWAP OPERATOR.

INITIALIZE.VECTOR

<INITIALIZE.VECTOR DESIGNATOR> ::= INITIALIZE.VECTOR (<TABLE ADDRESS>)

<TABLE ADDRESS> ::= <ADDRESS GENERATOR>

FOR USE BY SORT ONLY.

THE <TABLE ADDRESS> POINTS TO THE TABLE CONTAINING THE VECTOR ADDRESS, THE VECTOR LEVEL-1 ADDRESS, THE KEY TABLE ADDRESS, AND THE VECTOR LIMIT ADDRESS.

THREAD.VECTOR

<THREAD.VECTOR DESIGNATOR> ::= THREAD.VECTOR (<TABLE ADDRESS>,<INDEX>)

<TABLE ADDRESS> ::= <ADDRESS GENERATOR>

<INDEX> ::= <EXPRESSION>

FOR USE BY SORT ONLY.

THE <TABLE ADDRESS> POINTS TO THE TABLE CONTAINING THE INFORMATION DESCRIBED UNDER "INITIALIZE.VECTOR". THE <INDEX> PROVIDES THE OFFSET FROM THE BEGINNING OF THE VECTOR TO THE NEXT RECORD TO BE USED FOR COMPARISON.

DISABLE.INTERRUPTS

<DISABLE.INTERRUPTS DESIGNATOR> ::= DISABLE.INTERRUPTS

FOR MCP USE ONLY.

THE <DISABLE INTERRUPTS DESIGNATOR> SUPPRESSES ALL INTERRUPTS UNTIL AN <ENABLE INTERRUPTS DESIGNATOR> IS ENCOUNTERED.

NOTE THAT THIS CONSTRUCT CANNOT BE EXECUTED BY NORMAL STATE PROGRAMS.

ENABLE.INTERRUPTS

<ENABLE.INTERRUPTS DESIGNATOR> ::= ENABLE.INTERRUPTS

FOR MCP USE ONLY.

THE <ENABLE INTERRUPTS DESIGNATOR> CAUSES THE MCP TO RETURN TO THE NORMAL INTERRUPT-PROCESSING MODE AFTER THE <DISABLE INTERRUPTS DESIGNATOR> HAS CHANGED THAT MODE. SEE ABOVE

NOTE THAT THIS CONSTRUCT CANNOT BE EXECUTED BY A NORMAL STATE PROGRAM.

ACCESS FILE INFORMATION

```
<ACCESS FILE INFORMATION
DESIGNATOR> ::=              ACCESS.FILE.INFORMATION (<FILE DESIGNATOR>,
                             <RETURN TYPE>,<DESTINATION>)

<FILE DESIGNATOR> ::=        <FILE IDENTIFIER>
                           / <SWITCH FILE IDENTIFIER> (<EXPRESSION>)

<RETURN TYPE> ::=            BIT / CHARACTER

<DESTINATION> ::=            <ADDRESS GENERATOR>
```

THE <ACCESS FILE INFORMATION DESIGNATOR> RETURNS THE END-OF-FILE POINTER AND THE DEVICE TYPE FROM THE FIB OF THE SPECIFIED FILE TO THE SPECIFIED DESTINATION.

THE INFORMATION MAY BE RETURNED AS EITHER BIT OR CHARACTER. THE FORMAT IS AS FOLLOWS:

```
01  DESTINATION.FIELD,
    02  EOF.POINTER      BIT(24),   % CHARACTER(8)
    02  DEVICE.TYPE      BIT(6);    % CHARACTER(2)
```

TO ENSURE THAT THE FIB EXISTS, THIS COMMUNICATE SHOULD ONLY BE USED ON OPEN FILES.

HARDWARE MONITOR

```
<HARDWARE MONITOR
DESIGNATOR> ::=              HARDWARE.MONITOR (<EXPRESSION>)
```

THE MONITOR MICRO-OPCODE WILL BE EXECUTED USING THE LOW-ORDER 8 BITS OF THE <EXPRESSION> AS ITS OPERAND. ALSO SEE APPENDIX IV.

SAVE STATE

<SAVE STATE DESIGNATOR> ::= SAVE.STATE

THE STATE OF THE INTERPRETER WILL BE STORED IN RS.M.MACHINE (SEE "B1700 MCP REFERENCE MANUAL"). EXECUTION WILL THEN CONTINUE.

DEBLANK

<DEBLANK DESIGNATOR>::= DEBLANK (<FIRST CHARACTER>)

<FIRST CHARACTER>::= <IDENTIFIER>

THE <FIRST CHARACTER> IS A SIMPLE IDENTIFIER WHICH DESCRIBES THE FIRST CHARACTER TO BE EXAMINED. DEBLANK REPEATLY INCREMENTS THE ADDRESS FIELD OF THE DESCRIPTOR FOR <FIRST CHARACTER> UNTIL <FIRST CHARACTER> DESCRIBES A NON-BLANK CHARACTER.

FREEZE PROGRAM

<FREEZE-PROGRAM DESIGNATOR>::= FREEZE.PROGRAM

EXECUTION OF THIS FUNCTION WILL PREVENT THE PROGRAM FROM BEING MOVED IN MEMORY OR FROM BEING ROLLED OUT OF MEMORY.

THAW PROGRAM

<THAW-PROGRAM DESIGNATOR>::= THAW.PROGRAM

EXECUTION OF THIS FUNCTION WILL ALLOW THE PROGRAM TO BE ROLLED OUT OF MEMORY. IT WILL NOT FORCE IT TO BE ROLLED OUT.

DUMP FOR ANALYSIS

<DUMP-FOR-
ANALYSIS DESIGNATOR>::= DUMP.FOR.ANALYSIS

EXECUTION OF THIS FUNCTION WILL CAUSE A DUMPFILE TO BE CREATED AND EXECUTION TO COUTINUE.

COMPILE CARD INFO

<COMPILE-CARD-
INFO DESIGNATOR>::= COMPILE.CARD.INFO
(<CCI DESTINATION FIELD>)

<CCI DESTINATION FIELD>::= <ADDRESS GENERATOR>

THIS FUNCTION IS INTENDED FOR USE BY THE COMPILERS ONLY. THE INFORMATION ON THE "COMPILE" CARD IS RETURNED IN THE FOLLOWING FORMAT:

OBJECT NAME	CHARACTER (30)
EXECUTE TYPE (DECIMAL)	CHARACTER (2)

"01" EXECUTE
"02" COMPILE AND GO
"03" COMPILE FOR SYNTAX
"04" COMPILE TO LIBRARY
"05" COMPILE AND SAVE
"06" GO PART OF COMPILE AND GO
"07" GO PART OF COMPILE AND SAVE

COMPILER PACK IDENTIFIER	CHARACTER (10)
COMPILER INTERPRETER NAME	CHARACTER (30)
COMPILER INTRINSIC NAME	CHARACTER (10)
COMPILER PRIORITY (DECIMAL)	CHARACTER (2)
COMPILER CHARGE NUMBER (DECIMAL)	CHARACTER (6)
COMPILER JOB NUMBER (DECIMAL)	CHARACTER (6)
COMPILER'S FIRST NAME	CHARACTER (10)
OBJECT PROGRAM'S FIRST NAME	CHARACTER (10)
COMPILER MIX NUMBER (DECIMAL)	CHARACTER (8)
COMPILATION DATE	BIT (36)

BITS	CONTENTS
0-8	JULIAN DAY (BINARY)
9-15	LAST TWO DIGITS OF YEAR (BINARY - BASE 1900)
16-35	TIME (BINARY - TENTHS OF SECONDS)

COMMUNICATE

<COMMUNICATE DESIGNATOR>::= COMMUNICATE (<EXPRESSION>)

THE <EXPRESSION> IS EXPECTED TO BE A VALID COMMUNICATE MESSAGE.
THIS IS INTENDED ONLY FOR EXPERIMENTAL TESTING OF COMMUNICATES.

MODIFY INSTRUCTION

```
<MODIFY INSTRUCTION> ::=      <CLEAR STATEMENT>
                            / <BUMP STATEMENT>
                            / <DECREMENT STATEMENT>

<CLEAR STATEMENT> ::=         CLEAR <ARRAY IDENTIFIER LIST>

ARRAY IDENTIFIER LIST> ::=    <ARRAY IDENTIFIER>
                            / <ARRAY IDENTIFIER>,
                              <ARRAY IDENTIFIER LIST>
```

AS THE SYNTAX INDICATES, THE <CLEAR STATEMENT> MAY ONLY "CLEAR" ARRAYS. IF THE ARRAY HAS BEEN DECLARED BIT OR FIXED, ZEROES ARE MOVED TO EACH ELEMENT. IF IT WAS DECLARED AS CHARACTER, BLANKS ARE MOVED TO EACH ELEMENT. PAGED ARRAYS MAY NOT BE "CLEARED".

```
<BUMP STATEMENT> ::=          BUMP <ADDRESS VARIABLE><MODIFIER>

<ADDRESS VARIABLE> ::=        SEE "ADDRESS VARIABLES"

<MODIFIER> ::=                <EMPTY>
                            / BY <EXPRESSION>

<DECREMENT STATEMENT> ::=     DECREMENT <ADDRESS VARIABLE><MODIFIER>
```

THE BUMP AND DECREMENT STATEMENTS PERFORM THE SAME FUNCTIONS AS THEIR COUNTERPARTS IN THE <EXPRESSION> (BUMPOR AND DECREMENTOR). SEE THOSE SECTIONS FOR SPECIFIC USAGE. SINCE THESE CONSTRUCTS EXIST AS STATEMENTS IN THEIR OWN RIGHTS, AND NOT MERELY AS PARTS OF THE <EXPRESSION>, THEY ARE INCLUDED HERE.

NULL STATEMENT

<NULL STATEMENT> ::= ;

THE SEMI-COLON IS CONSIDERED TO BE A STATEMENT IN ITS OWN RIGHT. IT MAY BE USED IN ANY CONSTRUCT WHERE THE SYNTAX REQUIRES THAT AN <EXECUTABLE STATEMENT> BE PRESENT, BUT THE USER WISHES TO EXECUTE NOTHING. IT IS MOST COMMONLY USED IN THE <IF STATEMENT> AND THE <CASE STATEMENT>, BUT MAY ALSO BE FUNCTIONAL IN THE READ, WRITE, AND SPACE STATEMENTS. REFER TO THE INDIVIDUAL DESCRIPTIONS FOR MORE SPECIFIC DETAILS.

EXAMPLE:

```
    CASE <EXPRESSION>;
         IF <EXPRESSION> THEN;
                         ELSE <STATEMENT>;
         ;
         DO;
         <EXECUTABLE STATEMENT LIST>
         END;
    END CASE;
```

NOTICE THAT THE ABOVE <CASE STATEMENT> CONTAINS THREE <EXECUTABLE STATEMENTS>: AN <IF STATEMENT>, A <NULL STATEMENT>, AND A <DO GROUP>. IF THE VALUE OF THE <EXPRESSION> FOLLOWING "CASE" IS 1, THEN NOTHING IS EXECUTED. IN ADDITION, THE ";" FOLLOWING "THEN" IS A <NULL STATEMENT>.

FILE ATTRIBUTE STATEMENT (CHANGE STATEMENT)

```
<FILE ATTRIBUTE
STATEMENT> ::=              CHANGE <FILE DESIGNATOR>
                            TO (<DYNAMIC FILE ATTRIBUTE LIST>)

<FILE DESIGNATOR> ::=         <FILE IDENTIFIER>
                            / <SWITCH FILE IDENTIFIER> (<EXPRESSION>)

<DYNAMIC FILE
ATTRIBUTE LIST> ::=           <DYNAMIC FILE ATTRIBUTE>
                            / <DYNAMIC FILE ATTRIBUTE>,
                              <DYNAMIC FILE ATTRIBUTE LIST>

<DYNAMIC FILE
ATTRIBUTE> ::=                <DYNAMIC MULTI-FILE IDENTIFICATION PART>
                            / <DYNAMIC FILE IDENTIFICATION PART>
                            / <DYNAMIC PACK.ID PART>
                            / <DYNAMIC DEVICE PART>
                            / <DYNAMIC TRANSLATION PART>
                            / <DYNAMIC FILE PARITY PART>
                            / <DYNAMIC VARIABLE RECORD PART>
                            / <DYNAMIC LOCK PART>
                            / <DYNAMIC BUFFERS PART>
                            / <DYNAMIC SAVE FACTOR PART>
                            / <DYNAMIC RECORD SIZE PART>
                            / <DYNAMIC RECORDS-PER-BLOCK PART>
                            / <DYNAMIC REEL NUMBER PART>
                            / <DYNAMIC NUMBER-OF-AREAS PART>
                            / <DYNAMIC BLOCKS-PER-AREA PART>
                            / <DYNAMIC ALL-AREAS-AT-OPEN PART>
                            / <DYNAMIC AREA-BY-CYLINDER PART>
                            / <DYNAMIC EU.SPECIAL PART>
                            / <DYNAMIC EU.INCREMENTED PART>
                            / <DYNAMIC USE.INPUT.BLOCKING
                              DESIGNATOR PART>
                            / <DYNAMIC SORTER-STATION PART>
                            / <DYNAMIC MULTI-PACK PART>
                            / <DYNAMIC END-OF-PAGE PART>
                            / <DYNAMIC OPEN-OPTION PART>
                            / <DYNAMIC REMOTE-KEY PART>
                            / <DYNAMIC NUMBER-OF-STATIONS PART>
                            / <DYNAMIC QUEUE-FAMILY-SIZE PART>
                            / <DYNAMIC FILE TYPE PART>
                            / <DYNAMIC WORK FILE PART>
                            / <DYNAMIC LABEL TYPE PART>
```

THE <FILE ATTRIBUTE STATEMENT> ALLOWS THE USER TO DYNAMICALLY CHANGE THE ATTRIBUTES OF HIS FILE DURING THE EXECUTION OF HIS PROGRAM. THIS STATEMENT MAY OCCUR AT ANY POINT IN THE PROGRAM,

BUT THE CHANGE WILL NOT BECOME EFFECTIVE UNTIL THE FILE IS OPENED. THAT IS, IF THE FILE IN QUESTION IS OPEN WHEN THE <FILE ATTRIBUTE STATEMENT> IS EXECUTED, THEN THE CHANGE WILL NOT OCCUR UNTIL THE FILE IS CLOSED AND RE-OPENED.

EACH <DYNAMIC FILE ATTRIBUTE> SHOULD BE CONSISTENT WITH THE FORMAT AND RESTRICTIONS OF ITS COUNTERPART LISTED IN THE "FILE DECLARATIONS". EXCEPTIONS TO THIS ARE SPECIFICALLY STATED BELOW.

IF A <DYNAMIC FILE ATTRIBUTE> IS OMITTED, THE ATTRIBUTE REMAINS AS IT WAS PREVIOUSLY SET.

IT SHOULD BE NOTED THAT THE FOLLOWING PROCESS IS MANDATORY WHEN CHANGING THE ATTRIBUTES OF AN OPEN FILE WHICH IS TO BE RE-OPENED:

1. CLOSE THE FILE WITH AN ATTRIBUTE WHICH CAUSES SPACE FOR THE FIB TO BE RETURNED: I.E., "LOCK", "RELEASE", ETC. (IF "CLOSE" IS USED WITHOUT ATTRIBUTES, THE FIB WILL NOT BE REBUILT FROM THE FPB, AND THE ATTRIBUTE WILL REMAIN UNCHANGED).

2. CHANGE THE DESIRED ATTRIBUTES.

3. OPEN THE FILE.

```
<DYNAMIC MULTI-FILE
IDENTIFICATION PART> ::=      MULTI.FILE.ID ←
                              <DYNAMIC MULTI-FILE IDENTIFICATION>

<DYNAMIC MULTI-FILE
IDENTIFICATION> ::=           <EXPRESSION>

<DYNAMIC FILE
IDENTIFICATION PART> ::=      FILE.ID←<DYNAMIC FILE IDENTIFICATION>

<DYNAMIC FILE
IDENTIFICATION> ::=           <EXPRESSION>

<DYNAMIC PACK.ID PART> ::=    PACK.ID←
                              <DYNAMIC PACK IDENTIFICATION>

<DYNAMIC PACK
IDENTIFICATION> ::=           <EXPRESSION>
```

THE <EXPRESSION>S OF THESE FOUR ATTRIBUTES ARE EACH ASSUMED TO BE CHARACTER STRINGS. IF THEY ARE BITS, HOWEVER, THEY WILL BE CONVERTED TO CHARACTERS IN THE FOLLOWING MANNER:

1. THE BITS ARE LEFT JUSTIFIED.

2. TRAILING BLANKS ARE APPENDED. HOWEVER, IF THE BITS ARE NOT A MULTIPLE OF 8, THEN THE STRING WILL APPEAR TO BE INVALID CHARACTERS.

EXAMPLE:

```
CHANGE F TO (FILE.ID←@F0E@);

WILL RESULT IN THE <FILE IDENTIFICATION>
BEING EQUAL TO:

@F0E4040404040404040404@
```

<DYNAMIC DEVICE PART> ::= DEVICE←<DYNAMIC DEVICE SPECIFIER>

<DYNAMIC DEVICE SPECIFIER> ::= <EXPRESSION>

THE LOW-ORDER 10 BITS OF THE <EXPRESSION> MUST BE CODED AS FOLLOWS (WHERE THE VARIANT IS THE HIGH ORDER 4 BITS, AND THE HARDWARE IS THE LOW-ORDER SIX):

DEVICE	HARDWARE	VARIANT
CARD	21	
MULTI.FUNCTION.CARD	4	
TAPE	27	
TAPE.9	28	
TAPE.7	25	
TAPE.PE	26	
TAPE.NRZ	24	
DISK	17	0 = SERIAL 1 = RANDOM
DISK.PACK	16	(SAME AS DISK)
DISK.FILE	12	(SAME AS DISK)
DISK.PACK.CENTURY	15	(SAME AS DISK)
DISK.PACK.CAELUS	14	(SAME AS DISK)
PRINTER	8	0 = BACKUP TAPE OR DISK 1 = BACKUP TAPE 2 = BACKUP DISK 3 = BACKUP TAPE OR DISK 4 = HARDWARE ONLY 5 = BACKUP TAPE ONLY 6 = BACKUP DISK ONLY 7 = BACKUP TAPE OR DISK ONLY
PRINTER FORMS	8	8 + PRINTER VARIANT
CARD.READER	21	
CARD.PUNCH	2	(SAME AS PRINTER)
CARD.PUNCH FORMS	2	(SAME AS PRINTER FORMS)
PUNCH	2	(SAME AS PRINTER)
PUNCH FORMS	2	(SAME AS PRINTER FORMS)
PUNCH.96	1	(SAME AS PRINTER)
PUNCH.96 FORMS	1	(SAME AS PRINTER FORMS)
READER.PUNCH	3	(SAME AS PRINTER)
READER.PUNCH FORMS	3	(SAME AS PRINTER FORMS)
READER.PUNCH.PRINTER	5	(SAME AS PRINTER)
READER.PUNCH.PRINTER FORMS	5	(SAME AS PRINTER FORMS)
PUNCH.PRINTER	18	(SAME AS PRINTER)
PUNCH.PRINTER FORMS	18	(SAME AS PRINTER FORMS)
PAPER.TAPE.PUNCH	20	(SAME AS PRINTER)
PAPER.TAPE.PUNCH FORMS	20	(SAME AS PRINTER FORMS)
PAPER.TAPE.READER	6	
READER.96	19	

SORTER.READER	10
READER.SORTER	10
SPO	22
CASSETTE	30
REMOTE	63
QUEUE	62
MFCU	4

<DYNAMIC TRANSLATION PART> ::= TRANSLATION ←
<DYNAMIC TRANSLATION SPECIFIER>

<DYNAMIC TRANSLATION SPECIFIER> ::= <EXPRESSION>

THE LOW-ORDER 3 BITS OF THE <EXPRESSION> DETERMINES THE TRANSLATION AS FOLLOWS:

000 = EBCDIC
001 = ASCII
010 = BCL

<DYNAMIC SORTER STATION PART> ::= SR.STATION ←
<DYNAMIC SORTER STATION SPECIFIER>

<DYNAMIC SORTER STATION SPECIFIER> ::= <EXPRESSION>

THE LOW-ORDER 3 BITS OF THE <EXPRESSION> DETERMINES THE TRANSLATION AS FOLLOWS:

001 = FIRST STATION
010 = SECOND STATION
111 = BOTH STATIONS

<DYNAMIC OPEN-OPTION PART>::= OPEN.OPTION ←
<DYNAMIC OPEN.OPTION SPECIFIER>

<DYNAMIC OPEN-OPTION SPECIFIER>::= <EXPRESSION>

THE LOW-ORDER 12 BITS OF THE EXPRESSION DETERMINE THE TYPE OF OPEN AS FOLLOWS (BITS ARE NUMBERED FROM LEFT TO RIGHT WITHIN THE 12):

```
BIT   FUNCTION (IF 1)
 0   = INPUT
 1   = OUTPUT
 2   = NEW
 3   = PUNCH
 4   = PRINT
 5   = NO.REWIND, INTERPRET
 6   = REVERSE, STACKERS
 7   = LOCK
 8   = LOCK.OUT
 11  = CODE.FILE
```

```
<DYNAMIC PARITY PART> ::=      PARITY←<DYNAMIC PARITY SPECIFIER>

<DYNAMIC PARITY
SPECIFIER> ::=                 <EXPRESSION>

<DYNAMIC VARIABLE
RECORD PART> ::=               VARIABLE←
                               <DYNAMIC VARIABLE RECORD SPECIFIER>

<DYNAMIC VARIABLE
RECORD SPECIFIER> ::=          <EXPRESSION>

<DYNAMIC LOCK PART> ::=        LOCK←<DYNAMIC LOCK SPECIFIER>

<DYNAMIC LOCK
SPECIFIER> ::=                 <EXPRESSION>

<DYNAMIC ALL-AREAS-
AT-OPEN PART> ::=              ALL.AREAS.AT.OPEN←
                               <DYNAMIC ALL-AREAS-AT-OPEN SPECIFIER>

<DYNAMIC ALL-AREAS-
AT-OPEN SPECIFIER> ::=         <EXPRESSION

<DYNAMIC AREA-BY
CYLINDER PART> ::=             AREA.BY.CYLINDER←
                               <DYNAMIC AREA-BY-CYLINDER SPECIFIER>

<DYNAMIC AREA-BY-
CYLINDER SPECIFIER> ::=        <EXPRESSION>

<DYNAMIC USE.INPUT.
BLOCKING PART> ::=             USE.INPUT.BLOCKING←
                               <DYNAMIC USE.INPUT.BLOCKING SPECIFIER>

<DYNAMIC USE.INPUT.
BLOCKING SPECIFIER> ::=        <EXPRESSION>

<DYNAMIC END-OF-
PAGE PART> ::=                 END.OF.PAGE.ACTION←
                               <DYNAMIC END-OF-PAGE SPECIFIER>

<DYNAMIC END-OF-
PAGE SPECIFIER> ::=            <EXPRESSION>

<DYNAMIC MULTI-
PACK PART>::=                  MULTI.PACK←
                               <DYNAMIC MULTI-PACK SPECIFIER>

<DYNAMIC MULTI-
PACK SPECIFIER> ::=            <EXPRESSION>
```

```
<DYNAMIC REMOTE-
KEY PART>::=                    REMOTE-KEY ←
                                <DYNAMIC REMOTE-KEY SPECIFIER>

<DYNAMIC REMOTE-
KEY SPECIFIER>::=               <EXPRESSION>

<DYNAMIC WORK
FILE PART>::=                   WORK.FILE ←
                                <DYNAMIC WORK FILE SPECIFIER>

<DYNAMIC WORK
FILE SPECIFIER>::=              <EXPRESSION>
```

ONLY THE LOW-ORDER BIT OF EACH OF THE ABOVE <EXPRESSION>S IS USED TO DETERMINE THE VALUE OF THE ATTRIBUTE. THE CODE DEFINITIONS ARE AS FOLLOWS:

PARITY	0	= ODD
	1	= EVEN
VARIABLE	0	= FIXED
	1	= VARIABLE
LOCK	0	= NOT LOCKED
	1	= LOCKED
ALL.AREAS.AT.OPEN	0	= ALLOCATE AREAS AS NEEDED
	1	= ALLOCATE ALL SPACE AT OPEN TIME
AREA.BY.CYLINDER	0	= PUT AREA ANYWHERE ON DISK
	1	= ONE AREA PER CYLINDER AT BEGINNING
USE.INPUT.BLOCKING	0	= TAKE ATTRIBUTES FROM FILE DECLARATION
	1	= TAKE ATTRIBUTES FROM DISK FILE HEADER SEE FILE ATTRIBUTES
END.OF.PAGE.ACTION	0	= NO DETECTION OF END-OF-PAGE
	1	= BRANCH TO <EOF PART> OF <WRITE STATEMENT> AT END OF PAGE ON PRINTER FILE
MULTI.PACK	1	= PLACE FILE ON MULTIPLE DISK PACKS
	0	= PLACE FILE ON SINGLE DISK PACK
REMOTE KEY	1	= REMOTE KEY IS PRESENT ON ALL READS AND WRITES TO THE FILE
	0	= REMOTE KEY IS NOT PRESENT
WORK.FILE	1	= INSERT JOB NUMBER IN FILE IDENTIFIER
	0	= LEAVE FILE IDENTIFIER ABOVE

```
<DYNAMIC EU.SPECIAL
PART> ::=                       EU.SPECIAL ←
                                <DYNAMIC EU.SPECIAL SPECIFIER>
                              / EU.SPECIAL ←
                                <DYNAMIC EU.SPECIAL SPECIFIER>,
                                EU.DRIVE ←
                                <DYNAMIC EU.DRIVE SPECIFIER>

<DYNAMIC EU.SPECIAL
SPECIFIER> ::=                  <EXPRESSION>

<DYNAMIC EU.DRIVE
SPECIFIER> ::=                  <EXPRESSION>

<DYNAMIC EU.
INCREMENTED PART> ::=           EU.INCREMENTED ←
                                <DYNAMIC EU.INCREMENTED SPECIFIER>
                              / EU.INCREMENTED ←
                                <DYNAMIC EU.INCREMENTED SPECIFIER>,
                                EU.INCREMENT ←
                                <DYNAMIC EU.INCREMENT SPECIFIER>

<DYNAMIC EU.INCREMENTED
SPECIFIER> ::=                  <EXPRESSION>

<DYNAMIC EU.
INCREMENT SPECIFIER> ::=        <EXPRESSION>
```

THE LOW-ORDER BIT OF THE EU.SPECIAL AND EU.INCREMENTED SPECIFIERS SERVES TO INDICATE WHETHER OR NOT THE ATTRIBUTE IS SET (0=OFF, 1=ON). IF THE ATTRIBUTE IS OFF, THEN INCLUSION OF THE EU.DRIVE AND EU.INCREMENT SPECIFIERS IS UNNECESSARY.

IF THESE ATTRIBUTES ARE SET ON, THEN THE DRIVE AND INCREMENT PARTS SHOULD BE INCLUDED, AND SHOULD CONFORM TO THE SPECIFICATIONS IN THE "FILE DECLARATIONS". IF OMITTED, THE <DYNAMIC EU.DRIVE SPECIFIER> IS NOT CHANGED. IF THE <DYNAMIC EU.INCREMENT SPECIFIER> HAS NEVER BEEN SET (I.E., IT IS 0), THEN IT IS SET TO ONE; OTHERWISE, IT TOO REMAINS UNCHANGED.

```
<DYNAMIC BUFFERS PART> ::=    BUFFERS←<DYNAMIC NUMBER OF BUFFERS>

<DYNAMIC NUMBER
OF BUFFERS> ::=               <EXPRESSION>

<DYNAMIC SAVE
FACTOR PART> ::=              SAVE←<DYNAMIC SAVE FACTOR>

<DYNAMIC SAVE FACTOR> ::=     <EXPRESSION>

<DYNAMIC RECORD
SIZE PART> ::=                RECORD.SIZE←<DYNAMIC RECORD SIZE>

<DYANMIC RECORD SIZE> ::=     <EXPRESSION>

<DYNAMIC RECORDS-
PER-BLOCK PART> ::=           RECORDS.PER.BLOCK←
                              <DYNAMIC RECORDS-PER-BLOCK>

<DYNAMIC RECORDS-
PER-BLOCK> ::=                <EXPRESSION>

<DYNAMIC REEL
NUMBER PART> ::=              REEL←<DYNAMIC REEL NUMBER>

<DYNAMIC REEL NUMBER> ::=     <EXPRESSION>

<DYNAMIC NUMBER-OF-
AREAS PART> ::=               NUMBER.OF.AREAS←
                              <DYNAMIC NUMBER-OF-AREAS>

<DYNAMIC NUMBER-
OF-AREAS> ::=                 <EXPRESSION>

<DYNAMIC BLOCKS-PER-
AREA PART> ::=                BLOCKS.PER.AREA←
                              <DYNAMIC BLOCKS-PER-AREA>

<DYNAMIC BLOCKS-PER
AREA> ::=                     <EXPRESSION>

<DYNAMIC QUEUE-FAMILY-
SIZE PART>::=                 QUEUE-FAMILY-SIZE←
                              <DYNAMIC QUEUE-FAMILY-SIZE>

<DYNAMIC QUEUE-
FAMILY-SIZE>::=               <EXPRESSION>

<DYNAMIC NUMBER-OF-
STATIONS PART>::=             NUMBER-OF-STATIONS←
                              <DYNAMIC NUMBER-OF-STATIONS SPECIFIER>
```

<DYNAMIC NUMBER-OF-
STATIONS SPECIFIER>::= <EXPRESSION>

<DYNAMIC FILE
TYPE PART>::= FILE.TYPE ←
<DYNAMIC FILE TYPE SPECIFIER>

<DYNAMIC FILE
TYPE SPECIFIER>::= <EXPRESSION>

THE VALUE OF THE EXPRESSION DETERMINES THE FILE TYPE:

VALUE	TYPE
0	DATA
7	INTERPRETER
8	CODE
9	DATA

<DYNAMIC LABEL
TYPE PART>::= LABEL TYPE ←
<DYNAMIC LABEL TYPE SPECIFIER>

<DYNAMIC LABEL
TYPE SPECIFIER>::= <EXPRESSION>

THE VALUE OF THE EXPRESSION DETERMINES THE LABEL TYPE.

VALUE	TYPE
0	BURROUGHS STANDARD LABEL
1	UNLABELED

THE ABOVE <EXPRESSION>S RETURN A BIT STRING WHICH SHOULD BE CONSISTENT WITH THE FORMATS AND RESTRICTIONS LISTED IN THE "FILE DECLARATIONS".

STOP STATEMENT

<STOP STATEMENT> ::= STOP
/ STOP <EXPRESSION>

THE <STOP STATEMENT> IS A COMMUNICATE TO THE MCP THAT THE PROGRAM HAS FINISHED. IT SHOULD NOT BE CONFUSED WITH "FINI" WHICH IS THE FINAL STATEMENT IN THE PROGRAM.

"STOP <EXPRESSION>" IS INTENDED FOR USE BY THE COMPILERS ONLY. THE <EXPRESSION> COMMUNICATES THE NUMBER OF SYNTAX ERRORS TO THE MCP.

ZIP STATEMENT

<ZIP STATEMENT> ::= ZIP <EXPRESSION>

THE <ZIP STATEMENT> ALLOWS THE USER TO PASS CONTROL INSTRUCTIONS TO THE MCP. THE <EXPRESSION> SHOULD GENERATE A CHARACTER STRING WHOSE VALUE IS A VALID MCP CONTROL STATEMENT AS DEFINED IN THE "B1700 SOFTWARE OPERATIONAL GUIDE".

SEARCH STATEMENT

```
<SEARCH STATEMENT> ::=          <SEARCH PART>
                              / <SEARCH PART>; <FILE MISSING PART>
                              / <SEARCH PART>; <FILE LOCKED PART>
                              / <SEARCH PART>; <FILE MISSING PART>
                                <FILE LOCKED PART>

<SEARCH PART> ::=               SEARCH.DIRECTORY (<SEARCH OBJECT>,
                                <SEARCH RESULT>,<SEARCH RESULT MODE>)

<SEARCH OBJECT> ::=             <ADDRESS GENERATOR>

<SEARCH RESULT> ::=             <ADDRESS GENERATOR>

<SEARCH RESULT MODE> ::=        BIT / CHARACTER

<FILE MISSING PART> ::=         ON FILE.MISSING <EXECUTABLE STATEMENT>

<FILE LOCKED PART> ::=          ON FILE.LOCKED <EXECUTABLE STATEMENT>
```

THE <SEARCH STATEMENT> ALLOWS THE USER TO EXTRACT CERTAIN INFORMATION CONTAINED IN THE DISK FILE HEADER SPECIFIED BY THE <SEARCH OBJECT>.

THE <SEARCH OBJECT> IS EXPECTED TO BE 30 CHARACTERS IN LENGTH WHERE THE FIRST 10 CHARACTERS ARE THE PACK IDENTIFICATION, THE SECOND 10 CHARACTERS ARE THE MULTI-FILE IDENTIFICATION, AND THE 3RD 10 ARE THE FILE IDENTIFICATION. FILE NAMES LESS THAN 10 CHARACTERS MUST BE LEFT-JUSTIFIED IN THEIR RESPECTIVE FIELDS WITH TRAILING BLANKS APPENDED. IF ONLY ONE FILE NAME EXISTS, THAT NAME SHOULD BE LEFT-JUSTIFIED IN THE MULTI-FILE IDENTIFICATION FIELD, AND THE FILE IDENTIFICATION SHOULD BE BLANK.

THE <SEARCH RESULT> SPECIFIES THE RECEIVING FIELD AND SHOULD BE 360 BITS LONG IF BIT MODE IS SPECIFIED, OR 59 BYTES IF CHARACTER MODE IS SPECIFIED.

THE INFORMATION IS RETURNED IN THE FOLLOWING FORMAT:

```
01  FILE.HEADER.FORMAT,
      02  OPEN.TYPE                BIT (24),  %  CHARACTER (1)
      02  NO.USERS                 BIT (24),  %  CHARACTER (2)
      02  RECORD.SIZE              BIT (24),  %  CHARACTER (4)
      02  RECORDS.PER.BLOCK        BIT (24),  %  CHARACTER (4)
      02  EOF.POINTER              BIT (24),  %  CHARACTER (8)
      02  SEGMENTS.PER.AREA        BIT (24),  %  CHARACTER (8)
      02  USER.OPEN.OUTPUT         BIT (24),  %  CHARACTER (1)
      02  FILE.TYPE                BIT (24),  %  CHARACTER (2)
      02  PERMANENT.FLAG           BIT (24),  %  CHARACTER (2)
      02  BLOCKS.PER.AREA          BIT (24),  %  CHARACTER (6)
      02  AREAS.REQUESTED          BIT (24),  %  CHARACTER (3)
      02  AREA.COUNTER             BIT (24),  %  CHARACTER (3)
      02  SAVE.FACTOR              BIT (24),  %  CHARACTER (3)
      02  CREATION.DATE            BIT (24),  %  CHARACTER (6)
      02  LAST.ACCESS.DATE         BIT (24),  %  CHARACTER (6)
```

NOTE: THIS FORMAT MAY BE SUBJECT TO CHANGE.

THE <FILE MISSING PART> AND <FILE LOCKED PART> ALLOW THE USER TO SPECIFY THE COURSE OF ACTION SHOULD EITHER OF THESE CONDITIONS ARISE.

ACCESS FILE HEADER STATEMENT

```
<ACCESS FILE HEADER
STATEMENT> ::=                <ACCESS FILE HEADER PART>;
                            / <ACCESS FILE HEADER PART>;
                              <FILE MISSING PART>
                            / <ACCESS FILE HEADER PART>;
                              <FILE LOCKED PART>
                            / <ACCESS FILE HEADER PART>;
                              <FILE MISSING PART>
                              <FILE LOCKED PART>

<ACCESS FILE HEADER
PART> ::=                     READ.FILE.HEADER
                              (<FILE NAME>, <DESTINATION FIELD>)
                            / WRITE.FILE.HEADER
                              (<FILE NAME>, <SOURCE FIELD>)

<FILE NAME> ::=               <ADDRESS GENERATOR>

<DESTINATION FIELD> ::=       <ADDRESS GENERATOR>

<SOURCE FIELD> ::=            <ADDRESS GENERATOR>

<FILE MISSING PART> ::=       ON FILE.MISSING <EXECUTABLE STATEMENT>

<FILE LOCKED PART> ::=        ON FILE.LOCKED <EXECUTABLE STATEMENT>
```

THE <ACCESS FILE HEADER STATEMENT> IS INTENDED FOR USE IN SYSTEMS PROGRAMS ONLY. IT ENABLES THE PROGRAMMER TO EITHER READ OR WRITE A FILE HEADER.

THE <FILE NAME> IS EXPECTED TO BE A 30-CHARACTER FIELD WHERE THE FIRST 10 CHARACTERS ARE THE PACK.ID, THE SECOND 10 CHARACTERS ARE THE MULTI-FILE IDENTIFICATION AND THE THIRD 10, THE FILE IDENTIFICATION. FILE NAMES LESS THAN 10 CHARACTERS ARE LEFT-JUSTIFIED IN THEIR RESPECTIVE FIELDS. IF ONLY ONE FILE NAME EXISTS, IT IS LEFT-JUSTIFIED IN THE MULTI-FILE IDENTIFICATION, AND THE FILE IDENTIFICATION SHOULD BE SET TO BLANKS.

THE <SOURCE FIELD> OR <DESTINATION FIELD> SPECIFIES, RESPECTIVELY, THE SENDING OR RECEIVING FIELD, AND IS EXPECTED TO BE 576 TO 4320 BITS IN LENGTH DEPENDING UPON THE NUMBER OF AREAS ALLOCATED. INFORMATION IS PASSED IN THE FILE HEADER FORMAT. REFER TO THE "B1700 MCP MANUAL" FOR SPECIFICS.

THE <FILE MISSING PART> AND <FILE LOCKED PART> ENABLE THE PROGRAMMER TO SPECIFY THE COURSE OF ACTION SHOULD EITHER OF THESE CONDITIONS ARISE.

NOTE THAT EXTREME CAUTION IS ADVISED WHEN WRITING A FILE HEADER.

SEND STATEMENT

```
<SEND STATEMENT> ::=          <SEND PART>;
                            / <SEND PART>; <QUEUE FULL PART>
                            / <SEND PART>; <INVALID REQUEST PART>
                            / <SEND PART>; <QUEUE FULL PART>
                              <INVALID REQUEST PART>

<SEND PART> ::=               SEND <MESSAGE SOURCE> TO <QUEUE>

<MESSAGE SOURCE> ::=          <ADDRESS GENERATOR>

<QUEUE> ::=                   <EXPRESSION>

<QUEUE FULL PART> ::=         ON Q.FULL <EXECUTABLE STATEMENT>

<INVALID REQUEST PART> ::=    ON INVALID.REQUEST <EXECUTABLE STATEMENT>
```

THE <SEND STATEMENT> ALLOWS THE USER TO COMMUNICATE WITH OTHER PROGRAMS BY PLACING A MESSAGE IN A SPECIFIED QUEUE WHICH MAY BE ACCESSED BY THE OTHER PROGRAMS WITH THE <RECEIVE STATEMENT>.

THE <MESSAGE SOURCE> SPECIFIES THE MESSAGE TO BE PLACED IN THE QUEUE. IT IS ENTERED AT THE END OF THE QUEUE AND MAY BE A MAXIMUM OF 65535 BITS IN LENGTH.

THE <QUEUE> SPECIFIES THE 20-CHARACTER NAME OF THE QUEUE, THE NAME THAT MUST BE USED BY ALL PROGRAMS ACCESSING THAT QUEUE. THE FORMAT OF THE QUEUE NAME IS THE SAME AS A FILE HEADER. THAT IS, THE FIRST 10 CHARACTERS IS THE QUEUE FAMILY NAME, AND THE SECOND 10 IS THE SUB-QUEUE NAME. NAMES LESS THAN 10 CHARACTERS ARE LEFT JUSTIFIED IN IN THEIR RESPECTIVE FIELDS. IF ONLY ONE NAME EXISTS, IT IS LEFT JUSTIFIED IN THE FAMILY QUEUE FIELD, AND SUB-QUEUE IS SET TO BLANKS.

THE <QUEUE FULL PART> WILL BE EXECUTED IF THE MAXIMUM QUEUE SIZE HAS BEEN REACHED. IF UNSPECIFIED, A QUEUE MAY CONTAIN 1023 MESSAGES. THE MAXIMUM NUMBER OF QUEUE MESSAGES MAY BE SET TO ANY AMOUNT LESS THAN 1023 WITH THE <QUEUE SIZE DESIGNATOR>. THE CURRENT POPULATION OF A QUEUE MAY BE OBTAINED WITH THE <QUEUE POPULATION DESIGNATOR>. FOR FURTHER DETAILS, SEE THE APPROPRIATE SECTIONS.

THE <INVALID REQUEST PART> WILL BE EXECUTED IF THE <SEND PART> CANNOT, FOR SOME REASON, BE RECOGNIZED BY THE MCP.

NOTE: SINCE MESSAGES ARE SENT TO THE END OF THE QUEUE AND RECEIVED FROM THE TOP, IT IS THE PROGRAMMER-S RESPONSIBILITY TO ENSURE THAT MESSAGES ARE RECEIVED IN THE CORRECT ORDER.

RECEIVE STATEMENT

```
<RECEIVE STATEMENT> ::=        <RECEIVE PART>;
                             / <RECEIVE PART>; <QUEUE EMPTY PART>
                             / <RECEIVE PART>; <INVALID REQUEST PART>
                             / <RECEIVE PART>; <QUEUE EMPTY PART>
                               <INVALID REQUEST PART>

<RECEIVE PART> ::=             RECEIVE <MESSAGE DESTINATION>
                               FROM <QUEUE>

<MESSAGE DESTINATION> ::=      <ADDRESS GENERATOR>

<QUEUE> ::=                    <EXPRESSION>

<QUEUE EMPTY PART> ::=         ON Q.EMPTY <EXECUTABLE STATEMENT>

<INVALID REQUEST PART> ::=     ON INVALID.REQUEST <EXECUTABLE STATEMENT>
```

THE <RECEIVE STATEMENT> ALLOWS THE USER TO ACCEPT MESSAGES FROM OTHER PROGRAMS BY RECEIVING THE FIRST MESSAGE FROM THE SPECIFIED QUEUE.

THE <MESSAGE DESTINATION> SPECIFIES THE FIELD RECEIVING THE MESSAGE. <QUEUE> SPECIFIES THE QUEUE NAME AND SHOULD CONFORM TO THE SPECIFICATIONS DESCRIBED UNDER "SEND STATEMENT". IF ONLY ONE NAME IS SPECIFIED, IT WILL BE TAKEN AS THE FAMILY NAME AND THE EARLIEST MESSAGE IN ANY QUEUE HAVING THAT FAMILY NAME WILL BE SELECTED. OTHERWISE THE FIRST MESSAGE IN THE SPECIFIED QUEUE WILL BE SELECTED.

THE <QUEUE EMPTY PART> IS EXECUTED WHEN THE DESIGNATED QUEUE HAS NO MESSAGES. IF THE DESIGNATED QUEUE IS NON-EXISTENT, THE QUEUE IS CREATED, AND <QUEUE EMPTY PART> IS EXECUTED.

THE <INVALID REQUEST PART> IS EXECUTED WHEN THE <RECEIVE PART> CANNOT, FOR SOME REASON, BE RECOGNIZED OR EXECUTED BY THE MCP.

NOTE: IT IS THE PROGRAMMER-S RESPONSIBILITY TO ENSURE THAT HE RECEIVES THE CORRECT MESSAGE.

ARRAY PAGE TYPE STATEMENT

```
<ARRAY PAGE TYPE
STATEMENT> ::=              <ARRAY PAGE TYPE DESIGNATOR>
                            (<PAGED ARRAY NAME>,<PAGE NUMBER>)

<ARRAY PAGE TYPE
DESIGNATOR> ::=             MAKE.READ.ONLY
                          / MAKE.READ.WRITE

<PAGED ARRAY NAME> ::=      <IDENTIFIER>

<PAGE NUMBER> ::=           <EXPRESSION>
```

THE <ARRAY PAGE TYPE STATEMENT> ALLOWS THE USER TO MARK CERTAIN PAGED ARRAY PAGES AS READ-ONLY. WHEN THIS IS DONE, A PAGE WILL NOT BE WRITTEN OUT TO DISK EVERY TIME IT IS OVERLAYED.

MAKE.READ.WRITE ALLOWS THE USER TO CHANGE INFORMATION ON A PAGED ARRAY, AND TO HAVE THAT ARRAY WRITTEN ON DISK WHEN IT IS OVERLAYED. IT IS ONLY NECESSARY TO SPECIFY MAKE.READ.WRITE AFTER A MAKE.READ.ONLY SPECIFICATION.

IT IS THE PROGRAMMER-S RESPONSIBILITY TO ENSURE THAT THE INFORMATION IN A PAGE MARKED READ-ONLY IS NOT CHANGED. IN ADDITION, THE USER IS RESPONSIBLE FOR GUARANTEEING CORRECT PAGE NUMBER SPECIFICATIONS. THERE IS NO SYNTAX CHECK FOR EITHER.

EXAMPLE:

```
DECLARE PAGED (32) P (1024) BIT(30), T1 BIT(24);
T1 ←1;
DO FOREVER;
   MAKE.READ.ONLY (P, BUMP T1);
   IF T1 = 15 THEN UNDO;
END;
  .
  .
  .
MAKE.READ.WRITE (P, 0);
```

COROUTINE STATEMENT

```
<COROUTINE STATEMENT>::=      <COROUTINE ENTRY STATEMENT>
                            / <COROUTINE EXIT STATEMENT>

<COROUTINE
ENTRY STATEMENT>::=           ENTER.COROUTINE
                              (<COROUTINE TABLE SPECIFIER>)

<COROUTINE
TABLE SPECIFIER>::=           <ADDRESS GENERATOR>

<COROUTINE
EXIT STATEMENT>::=            EXIT.COROUTINE
                              (<COROUTINE TABLE SPECIFIER>)
```

THE <COROUTINE TABLE SPECIFIER> ASSOCIATED WITH ENTER.COROUTINE AND EXIT.COROUTINE IS ASSUMED TO DESCRIBE A TABLE WITH THE FOLLOWING FORMAT:

```
DECLARE
     01  TABLE
          ,02  NUMBER.OF.ENTRIES BIT(4)
          ,02  ENTRY.ADDRESS BIT(32)
          ,02  PPS.COPY(16) BIT(32)
```

A. ENTER.COROUTINE: THE <COROUTINE TABLE SPECIFIER> IS ASSUMED TO HAVE THE FORMAT DESCRIBED ABOVE. THE CURRENT CODE ADDRESS. IS PUSHED ON TO THE PROGRAM POINTER STACK. THE NUMBER OF ELEMENTS OF PPS.COPY THAT IS SPECIFIED BY NUMBER.OF.ENTRIES IS PUSHED ONTO THE PROGRAM POINTER STACK. THE ADDRESS OF THE NEXT INSTRUCTION IS TAKEN FROM ENTRY.ADDRESS.

B. EXIT.COROUTINE: THE <COROUTINE TABLE SPECIFIER> IS ASSUMED TO DESCRIBE A TABLE OF THE FORMAT GIVEN ABOVE. THE CURRENT NESTING LEVEL IS STORED IN NUMBER.OF.ENTRIES. THE CURRENT CODE ADDRESS IS STORED IN ENTRY.ADDRESS. THE NUMBER (AS SPECIFIED BY NUMBER.OF.ENTRIES) OF ENTRIES ON THE TOP OF THE PROGRAM POINTER STACK IS COPIED TO PPS.COPY(0) THROUGH PPS.COPY(NUMBER.OF.ENTRIES-1). IF NUMBER.OF.ENTRIES IS 0. THEN NOTHING IS COPIED. AN "UNDO" IS PERFORMED, USING NUMBER.OF.ENTRIES AS THE NUMBER OF ENTRIES ON TOP OF THE PROGRAM POINTER STACK.

NOTE: UPON FIRST EXECUTION OF ENTER.COROUTINE, THE TABLE MUST ALREADY BE SET UP. THE EASIEST WAY TO ACCOMPLISH THIS IS TO MAKE THE FIRST EXECUTABLE STATEMENT IN THE COROUTINE TO BE ENTERED AN EXIT.COROUTINE STATEMENT. THE FIRST ENTRANCE TO THE COROUTINE IS THEN ACCOMPLISHED BY A CALL STATEMENT.

NOTE: THIS IS NOT A GENERAL COROUTINE MECHANISM--I.E., IT IS NOT SYMMETRIC. THE ROUTINE EXECUTING THE ENTER.COROUTINE IS A "MASTER" TO THE "SLAVE" ROUTINE WHICH CONTAINS THE EXIT.COROUTINE-S.

NOTE: EXIT.COROUTINE CAN ONLY APPEAR WITHIN PROCEDURES WITH NO PARAMETERS AND NO LOCAL DATA; I.E., THOSE PROCEDURES WHICH DO NOT CHANGE THE CONTROL STACK.

WAIT STATEMENT

```
<WAIT STATEMENT>::=        WAIT (<NUMBER OF TENTHS OF SECONDS>)
                         / WAIT (<NUMBER OF TENTHS OF SECONDS>,
                           DC.IO.COMPLETE)
                         / WAIT (,DC.IO.COMPLETE)
```

THE WAIT STATEMENT CAUSES THE PROGRAM-S EXECUTION TO BE SUSPENDED UNTIL SOME EVENT OCCURS. THE FIRST FORM WAITS FOR THE SPECIFIED NUMBER OF TENTHS OF SECONDS TO PASS. THE SECOND FORM WAITS FOR THE NUMBER OF TENTHS OF SECONDS TO PASS OR UNTIL AN I/O OPERATION COMPLETES ON A DATA-COMMUNICATIONS FILE, WHICHEVER OCCURS FIRST. THE THIRD FORM WAITS UNTIL AN I/O OPERATION COMPLETES ON A DATA-COMMUNICATIONS FILE.

APPENDIX I: SYNTAX OF THE SDL LANGUAGE

```
<PROGRAM> ::=                       <DECLARATION STATEMENT LIST>
                                    <PROCEDURE STATEMENT LIST>
                                    <EXECUTABLE STATEMENT LIST>
                                    FINI

<DECLARATION STATEMENT
LIST> ::=                           <EMPTY>
                                  / <DECLARATION STATEMENT>
                                    <DECLARATION STATEMENT LIST>

<DECLARATION STATEMENT> ::=         <DECLARE STATEMENT>;
                                  / <FILE DECLARATION STATEMENT>;
                                  / <SWITCH FILE DECLARATION
                                    STATEMENT>;
                                  / <DEFINE STATEMENT>;
                                  / <FORWARD DECLARATION>
                                  / <USE STATEMENT>;
                                  / <SEGMENT STATEMENT>
                                    <DECLARATION STATEMENT>

<SEGMENT STATEMENT> ::=             SEGMENT (<SEGMENT IDENTIFIER>);
                                  / SEGMENT.PAGE (<SEGMENT IDENTIFIER>
                                    OF <PAGE IDENTIFIER>);

<SEGMENT IDENTIFIER> ::=            <IDENTIFIER>

<PAGE IDENTIFIER> ::=               <IDENTIFIER>

<DECLARE STATEMENT> ::=             DECLARE <DECLARE ELEMENT>
                                  / <DECLARE STATEMENT>, <DECLARE ELEMENT>

<DECLARE ELEMENT> ::=               <DECLARED PART>
                                    <TYPE PART>
                                  / <STRUCTURE LEVEL NUMBER>
                                    <STRUCTURE DECLARED PART>
                                    <STRUCTURE TYPE PART>
                                  / DYNAMIC <SIMPLE IDENTIFIER>
                                    <DYNAMIC TYPE PART>
                                  / PAGED <ELEMENTS-PER-PAGE PART>
                                    <ARRAY IDENTIFIER> <ARRAY BOUND>
                                    <TYPE PART>

<DECLARED PART> ::=                 <COMPLEX IDENTIFIER>
                                  / (<COMPLEX IDENTIFIER LIST>)
                                  / <COMPLEX IDENTIFIER> REMAPS
                                    <REMAP IDENTIFIER>

<COMPLEX IDENTIFIER
LIST> ::=                           <COMPLEX IDENTIFIER>
```

```
                                / <COMPLEX IDENTIFIER>,
                                  <COMPLEX IDENTIFIER LIST>

<COMPLEX IDENTIFIER> ::=          <SIMPLE IDENTIFIER>
                                / <ARRAY IDENTIFIER> <ARRAY BOUND>

<SIMPLE IDENTIFIER> ::=           <IDENTIFIER>

<ARRAY IDENTIFIER> ::=            <IDENTIFIER>

<ARRAY BOUND>       ::=           (<NUMBER>)

<REMAP IDENTIFIER> ::=            BASE
                                / <SIMPLE IDENTIFIER>
                                / <ARRAY IDENTIFIER>

<TYPE PART> ::=                   FIXED
                                / CHARACTER <FIELD SIZE>
                                / BIT <FIELD SIZE>

<FIELD SIZE> ::=                  (<NUMBER>)

<STRUCTURE LEVEL
NUMBER> ::=                       <NUMBER>

<STRUCTURE DECLARED
PART> ::=                         <DECLARED PART>
                                / FILLER
                                / <DUMMY PART> REMAPS <REMAP IDENTIFIER>

<DUMMY PART> ::=                  DUMMY <ARRAY BOUND PART>

<ARRAY BOUND PART> ::=            <EMPTY>
                                / <ARRAY BOUND>

<STRUCTURE TYPE PART> ::=         <EMPTY>
                                / <TYPE PART>

<DYNAMIC TYPE PART> ::=           BIT <DYNAMIC FIELD SIZE>
                                / CHARACTER <DYNAMIC FIELD SIZE>

<DYNAMIC FIELD SIZE> ::=          (<EXPRESSION>)

<ELEMENTS-PER-PAGE
PART> ::=                         (<NUMBER>)

<FILE DECLARATION
STATEMENT> ::=                    FILE <FILE DECLARE ELEMENT LIST>

<FILE DECLARE
ELEMENT LIST> ::=                 <FILE DECLARE ELEMENT>
                                / <FILE DECLARE ELEMENT>,
                                  <FILE DECLARE ELEMENT LIST>
```

```
<FILE DECLARE ELEMENT> ::=   <FILE IDENTIFIER><FILE ATTRIBUTE PART>

<FILE IDENTIFIER> ::=        <IDENTIFIER>

<FILE ATTRIBUTE PART> ::=    <EMPTY>
                           / (<FILE ATTRIBUTE LIST>)

<FILE ATTRIBUTE LIST> ::=    <FILE ATTRIBUTE>
                           / <FILE ATTRIBUTE>,
                             <FILE ATTRIBUTE LIST>

<FILE ATTRIBUTE> ::=         <LABEL PART>
                           / <DEVICE PART>
                           / <MODE PART>
                           / <BUFFERS PART>
                           / <VARIABLE RECORD PART>
                           / <LOCK PART>
                           / <SAVE FACTOR PART>
                           / <RECORD SPECIFICATION PART>
                           / <REEL NUMBER PART>
                           / <DISK FILE DESCRIPTION PART>
                           / <OPEN OPTION PART>
                           / <PACK-ID PART>
                           / <ALL.AREAS.AT.OPEN PART>
                           / <AREA.BY.CYLINDER PART>
                           / <EU.ASSIGNMENT PART>
                           / <MULTI.PACK PART>
                           / <USE INPUT BLOCKING PART>
                           / <SORTER STATION PART>
                           / <END.OF.PAGE PART>
                           / <REMOTE.KEY PART>
                           / <NUMBER.OF.STATIONS PART>
                           / <QUEUE.FAMILY.SIZE PART>
                           / <FILE TYPE PART>
                           / <WORK FILE PART>
                           / <LABEL TYPE PART>

<LABEL PART> ::=             LABEL = <FILE IDENTIFICATION PART>

<FILE IDENTIFICATION
PART> ::=                    <MULTI-FILE IDENTIFICATION>
                           / <MULTI-FILE IDENTIFICATION>  <SLASH>
                             <FILE IDENTIFICATION>

<MULTI-FILE
IDENTIFICATION> ::=          <CHARACTER STRING>

<FILE IDENTIFICATION> ::=    <CHARACTER STRING>       %DEFINED ABOVE

<DEVICE PART> ::=            DEVICE = <DEVICE SPECIFIER>

<DEVICE SPECIFIER> ::=       CARD
                           / TAPE
                           / TAPE.PE
```

```
                                / TAPE.7
                                / TAPE.9
                                / TAPE.NRZ
                                / MULTI.FUNCTION.CARD
                                / DISK <ACCESS MODE>
                                / DISK.PACK <ACCESS MODE>
                                / DISK.FILE <ACCESS MODE>
                                / DISK.PACK.CENTURY <ACCESS MODE>
                                / DISK.PACK.CAELUS <ACCESS MODE>
                                / PRINTER <DEVICE OPTION>
                                / PUNCH <DEVICE OPTION>
                                / PAPER.TAPE.PUNCH <DEVICE OPTION>
                                / PAPER.TAPE.READER
                                / PUNCH.PRINTER <DEVICE OPTION>
                                / READER.PUNCH <DEVICE OPTION>
                                / READER.PUNCH.PRINTER
                                  <DEVICE OPTION>
                                / READER.96
                                / PUNCH.96 <DEVICE OPTION>
                                / SPO
                                / CARD.READER
                                / CARD.PUNCH <DEVICE OPTION>
                                / MFCU
                                / SORTER.READER
                                / READER.SORTER
                                / CASSETTE
                                / REMOTE
                                / QUEUE

<ACCESS MODE> ::=                 <EMPTY> / SERIAL / RANDOM

<DEVICE OPTION> ::=               <EMPTY>
                                / <BACKUP OPTION>
                                / <SPECIAL FORMS OPTION>
                                / <SPECIAL FORMS OPTION> <BACKUP OPTION>

<BACKUP OPTION> ::=               <BACKUP SPECIFIER>
                                / OR <BACKUP SPECIFIER>

<BACKUP SPECIFIER> ::=            BACKUP / BACKUP TAPE / BACKUP DISK

<SPECIAL FORMS OPTION> ::=        FORMS

<MODE PART> ::=                   MODE = <MODE SPECIFIER>

<MODE SPECIFIER> ::=              <FILE PARITY PART>
                                / <TRANSLATION PART>
                                / <FILE PARITY PART> <TRANSLATION PART>

<FILE PARITY PART> ::=            ODD  /  EVEN

<TRANSLATION PART>::=             EBCDIC / ASCII / BCL / BINARY

<BUFFERS PART> ::=                BUFFERS = <NUMBER OF BUFFERS>
```

```
<NUMBER OF BUFFERS> ::=          <NUMBER>

<VARIABLE RECORD PART> ::=       VARIABLE

<LOCK PART> ::=                  LOCK

<SAVE FACTOR PART> ::=           SAVE = <SAVE FACTOR>

<SAVE FACTOR> ::=                <NUMBER>

<RECORD SPECIFICATION
PART> ::=                        RECORDS = <RECORD SIZE SPECIFIER>

<RECORD SIZE SPECIFIER> ::=      <PHYSICAL RECORD SIZE>
                               / <LOGICAL RECORD SIZE> <SLASH>
                                 <LOGICAL RECORDS PER PHYSICAL RECORD>

<PHYSICAL RECORD SIZE> ::=       <NUMBER>

<LOGICAL RECORD SIZE> ::=        <NUMBER>

<LOGICAL RECORDS PER
PHYSICAL RECORD> ::=             <NUMBER>

<REEL NUMBER PART> ::=           REEL = <REEL NUMBER>

<REEL NUMBER> ::=                <NUMBER>

<DISK FILE DESCRIPTION
PART> ::=                        AREAS = <NUMBER OF AREAS> <SLASH>
                                 <PHYSICAL RECORDS PER AREA>

<NUMBER OF AREAS> ::=            <NUMBER>

<PHYSICAL RECORDS
PER AREA> ::=                    <NUMBER>

<PACK.ID PART> ::=               PACK.ID = <PACK IDENTIFICATION>

<PACK IDENTIFICATION> ::=        <CHARACTER STRING>

<OPEN OPTION> ::=                OPEN.OPTION
                                 <OPEN OPTION ATTRIBUTE LIST>

<OPEN OPTION ATTRIBUTE
LIST> ::=                        <OPEN ATTRIBUTE>
                               / <OPEN ATTRIBUTE> <SLASH>
                               / <OPEN ATTRIBUTE LIST>

<ALL.AREAS.AT.
OPEN PART> ::=                   ALL.AREAS.AT.OPEN

<AREA.BY.CYLINDER PART> ::<      AREA.BY.CYLINDER
```

```
<EU.ASSIGNMENT PART> ::=      EU.SPECIAL = <NUMBER>
                            / EU.INCREMENTED = <NUMBER>

<USE.INPUT.BLOCKING
PART> ::=                     USE.INPUT.BLOCKING

<SORTER STATION PART> ::=     SR.STATION = <NUMBER>

<END.OF.PAGE PART> ::=        END.OF.PAGE.ACTION

<REMOTE.KEY PART::=           REMOTE.KEY

<NUMBER.OF.STATIONS>::=       NUMBER.OF.STATIONS = <NUMBER>

<MULTI PACK PART>::=          MULTI.PACK

<QUEUE FAMILY.SIZE
PART>::=                      QUEUE.FAMILY.SIZE = <NUMBER>

<FILE TYPE PART>::=           FILE.TYPE = <FILE TYPE SPECIFIER>

<FILE TYPE SPECIFIER>::=      DATA
                            / INTERPRETER
                            / CODE
                            / INTRINSIC

<WORK FILE PART>::=           WORK.FILE

<LABEL TYPE PART>::=          LABEL.TYPE = <LABEL TYPE SPECIFIER>

<LABEL TYPE SPECIFIER>::=     UNLABELED

<SWITCH FILE
DECLARATION STATEMENT>::=     SWITCH.FILE <SWITCH FILE
                              DECLARE ELEMENT LIST>

<SWITCH FILE
DECLARE ELEMENT LIST>::=      <SWITCH FILE DECLARE ELEMENT>
                            / <SWITCH FILE DECLARE ELEMENT>,
                              <SWITCH FILE DECLARE ELEMENT LIST>

<SWITCH FILE
DECLARE ELEMENT> ::=          <SWITCH FILE IDENTIFIER> (<FILE
                              IDENTIFIER LIST>)

<SWITCH FILE IDENTIFIER>::=   <IDENTIFIER>

<FILE IDENTIFIER LIST>::=     <FILE IDENTIFIER>
                            / <FILE IDENTIFIER>, <FILE IDENTIFIER LIST>

<DEFINE STATEMENT> ::=        DEFINE <DEFINE ELEMENT>
                            / <DEFINE STATEMENT>,
                              <DEFINE ELEMENT>
```

```
<DEFINE ELEMENT> ::=          <DEFINE IDENTIFIER>
                              <FORMAL PARAMETER PART>
                              AS <DEFINE STRING>

<DEFINE IDENTIFIER> ::=       <IDENTIFIER>

<FORMAL PARAMETER PART> ::=   (<FORMAL PARAMETER LIST>)
                            / [<FORMAL PARAMETER LIST>]
                            / <EMPTY>

<FORMAL PARAMETER LIST> ::=   <FORMAL PARAMETER>
                            / <FORMAL PARAMETER>,
                              <FORMAL PARAMETER LIST>

<FORMAL PARAMETER> ::=        <IDENTIFIER>

<DEFINE STRING> ::=           #<WELL-FORMED CONSTRUCT>#

<WELL-FORMED CONSTRUCT> ::=   <EMPTY>
                            / <BASIC COMPONENT>
                              <WELL-FORMED CONSTRUCT>

<BASIC COMPONENT> ::=         <RESERVED WORD>          %SEE APPENDIX
                            / <IDENTIFIER>
                            / <SPECIAL CHARACTER>
                            / <COMMENT STRING>
                            / <CONSTANT>               %DEFINED ABOVE

<DEFINE INVOCATION> ::=       <SIMPLE DEFINE IDENTIFIER>
                            / <PARAMETRIC DEFINE IDENTIFIER>
                              (<DEFINE ACTUAL PARAMETER LIST>)
                            / <PARAMETRIC DEFINE IDENTIFIER>
                              [<DEFINE ACTUAL PARAMETER LIST>]

<SIMPLE DEFINE
IDENTIFIER> ::=               <IDENTIFIER>

<PARAMETRIC
DEFINE IDENTIFIER> ::=        <IDENTIFIER>

<DEFINE ACTUAL
PARAMETER LIST> ::=           <DEFINE ACTUAL PARAMETER>
                            / <DEFINE ACTUAL PARAMETER>,
                              <DEFINE ACTUAL PARAMETER LIST>

<DEFINE ACTUAL
PARAMETER> ::=                <WELL-FORMED CONSTRUCT>

<FORWARD DECLARATION> ::=     FORWARD <COMPOUND PROCEDURE HEAD>

<COMPOUND PROCEDURE
HEAD> ::=                     <PROCEDURE HEAD>
                              <FORMAL PARAMETER DECLARATION
```

```
                                  STATEMENT LIST>

<PROCEDURE HEAD> ::=              <BASIC PROCEDURE HEAD>
                                  <PROCEDURE TYPE PART>;

<BASIC PROCEDURE HEAD> ::=        <PROCEDURE NAME>
                                  <FORMAL PARAMETER PART>

<PROCEDURE NAME> ::=              PROCEDURE <PROCEDURE IDENTIFIER>

<PROCEDURE IDENTIFIER> ::=        <TYPED PROCEDURE IDENTIFIER>
                                / <NON-TYPED PROCEDURE IDENTIFIER>

<TYPED PROCEDURE
IDENTIFIER> ::=                   <IDENTIFIER>

<NON-TYPED PROCEDURE
IDENTIFIER> ::=                   <IDENTIFIER>

<INTRINSIC IDENTIFIER>:=          <TYPED INTRINSIC IDENTIFIER>
                                / <NON-TYPED INTRINSIC IDENTIFIER>

<TYPED INTRINSIC
IDENTIFIER>::=                    <IDENTIFIER>

<NON-TYPED INTRINSIC
IDENTIFIER>::=                    <IDENTIFIER>

<PROCEDURE TYPE PART> ::=         <EMPTY>
                                / <FORMAL TYPE PART>

<FORMAL TYPE PART> ::=            <TYPE PART>
                                / <TYPE VARYING PART>

<TYPE PART> ::=                   FIXED
                                / CHARACTER <FIELD SIZE>
                                / BIT <FIELD SIZE>

<TYPE VARYING PART> ::=           VARYING
                                / BIT VARYING
                                / CHARACTER VARYING

<FORMAL PARAMETER DECLA-
RATION STATEMENT LIST> ::=        <EMPTY>
                                / <FORMAL PARAMETER DECLARATION STATEMENT>;
                                  <FORMAL PARAMETER DECLARATION
                                  STATEMENT LIST>

<FORMAL PARAMETER
DECLARATION STATEMENT> ::=        FORMAL <FORMAL ELEMENT>
                                / FORMAL.VALUE <FORMAL ELEMENT>
                                / <FORMAL PARAMETER DECLARATION STATEMENT>,
                                  <FORMAL ELEMENT>
```

```
<FORMAL ELEMENT> ::=          (<FORMAL IDENTIFIER LIST>)
                              <FORMAL TYPE PART>
                            / <FORMAL IDENTIFIER>
                              <FORMAL TYPE PART>

<FORMAL INDENTIFIER
LIST> ::=                     <FORMAL IDENTIFIER>
                            / <FORMAL IDENTIFIER>,
                              <FORMAL IDENTIFIER LIST>

<FORMAL IDENTIFIER> ::=       <COMPLEX IDENTIFIER>
                            / <VARYING ARRAY SPECIFIER>

<COMPLEX IDENTIFIER> ::=      <SIMPLE IDENTIFIER>       %DEFINED ABOVE
                            / <ARRAY IDENTIFIER>        %DEFINED ABOVE
                              <ARRAY BOUND>             %DEFINED ABOVE

<VARYING ARRAY
SPECIFIER> ::=                <ARRAY IDENTIFIER>
                              <VARYING ARRAY BOUND>

<VARYING ARRAY BOUND> ::=     (*)

<USE STATEMENT> ::=           USE (<SIMPLE IDENTIFIER LIST>)
                              OF <DEFINE IDENTIFIER>

<SIMPLE IDENTIFIER
LIST> ::=                     <SIMPLE IDENTIFIER>
                            / <SIMPLE IDENTIFIER>,
                              <SIMPLE IDENTIFIER LIST>

<SIMPLE IDENTIFIER> ::=       <IDENTIFIER>

<DEFINE IDENTIFIER> ::=       <IDENTIFIER>

<PROCEDURE STATEMENT
LIST> ::=                     <EMPTY>
                            / <PROCEDURE STATEMENT>;
                              <PROCEDURE STATEMENT LIST>

<PROCEDURE STATEMENT> ::=     <PROCEDURE DEFINITION>
                            / <SEGMENT STATEMENT>
                              <PROCEDURE STATEMENT>

<PROCEDURE DEFINITION> ::=    <COMPOUND PROCEDURE HEAD> %DEFINED ABOVE
                              <PROCEDURE BODY>

<PROCEDURE BODY> ::=          <DECLARATION STATEMENT LIST>
                              <PROCEDURE STATEMENT LIST>
                              <PROCEDURE EXECUTABLE STATEMENT LIST>
                              <PROCEDURE ENDING>

<PROCEDURE EXECUTABLE
STATEMENT LIST> ::=           <PROCEDURE EXECUTABLE STATEMENT>
```

```
                                / <PROCEDURE EXECUTABLE STATEMENT>
                                  <PROCEDURE EXECUTABLE STATEMENT LIST>

<PROCEDURE EXECUTABLE
STATEMENT> ::=                    <EXECUTABLE STATEMENT>
                                / <RETURN STATEMENT>;
                                / <SEGMENT STATEMENT>      %DEFINED ABOVE
                                  <PROCEDURE EXECUTABLE STATEMENT>

<RETURN STATEMENT> ::=            <TYPED PROCEDURE RETURN STATEMENT>
                                / <NON-TYPED PROCEDURE RETURN STATEMENT>

<TYPED PROCEDURE
RETURN STATEMENT> ::=             RETURN <EXPRESSION>

<NON-TYPED PROCEDURE
RETURN STATEMENT> ::=             RETURN
                                / RETURN.AND.ENABLE.INTERRUPTS

<PROCEDURE ENDING> ::=            END
                                / END <PROCEDURE IDENTIFIER>

<EXECUTABLE STATEMENT
LIST> ::=                         <EXECUTABLE STATEMENT>
                                / <EXECUTABLE STATEMENT>
                                  <EXECUTABLE STATEMENT LIST>

<EXECUTABLE STATEMENT> ::=        <DO GROUP>;
                                / <IF STATEMENT>
                                / <CASE STATEMENT>;
                                / <ASSIGNMENT STATEMENT>;
                                / <EXECUTE-PROCEDURE STATEMENT>;
                                / <EXECUTE-FUNCTION STATEMENT>;
                                / <GROUP TERMINATION STATEMENT>;
                                / <I/O CONTROL STATEMENT>
                                / <MODIFY INSTRUCTION>;
                                / <NULL STATEMENT>
                                / <STOP STATEMENT>;
                                / <FILE ATTRIBUTE STATEMENT>;
                                / <ZIP STATEMENT>
                                / <SEARCH STATEMENT>
                                / <ACCESS FILE HEADER STATEMENT>
                                / <SEND STATEMENT>
                                / <RECEIVE STATEMENT>
                                / <ARRAY PAGE TYPE STATEMENT>
                                / <COROUTINE STATEMENT>
                                / <WAIT STATEMENT>;
                                / <SEGMENT STATEMENT>
                                  <EXECUTABLE STATEMENT>

<DO GROUP> ::=                    <GROUP HEAD>
                                  <GROUP BODY>

<GROUP HEAD> ::=                  <GROUP NAME>
```

```
                                <FOREVER PART>;

<GROUP NAME> ::=                DO
                              / DO <GROUP IDENTIFIER>

<FOREVER PART> ::=              <EMPTY>
                              / FOREVER

<GROUP IDENTIFIER> ::=          <IDENTIFIER>

<GROUP BODY> ::=                <EXECUTABLE STATEMENT LIST>%DEFINED ABOVE
                                <GROUP ENDING>

<GROUP ENDING> ::=              END
                              / END <GROUP IDENTIFIER>

<IF STATEMENT> ::=              <IF CLAUSE>
                                <EXECUTABLE STATEMENT>
                              / <IF CLAUSE>
                                <EXECUTABLE STATEMENT>
                                ELSE <EXECUTABLE STATEMENT>

<IF CLAUSE> ::=                 IF <EXPRESSION> THEN

<EXPRESSION LIST> ::=           <EXPRESSION>
                              / <EXPRESSION>,
                                <EXPRESSION LIST>

<EXPRESSION> ::=                <STRING EXPRESSION>
                              / <STRING EXPRESSION>
                                CAT <EXPRESSION>

<STRING EXPRESSION> ::=         <LOGICAL FACTOR>
                              / <LOGICAL FACTOR>
                                <OR-ING OPERATOR>
                                <STRING EXPRESSION>

<OR-ING OPERATOR> ::=           OR  /  EXOR

<LOGICAL FACTOR> ::=            <LOGICAL SECONDARY>
                              / <LOGICAL SECONDARY>
                                AND <LOGICAL FACTOR>

<LOGICAL SECONDARY> ::=         <LOGICAL PRIMARY>
                              / NOT <LOGICAL PRIMARY>

<LOGICAL PRIMARY> ::=           <ARITHMETIC EXPRESSION>
                              / <ARITHMETIC EXPRESSION>
                                <RELATION>
                                <ARITHMETIC EXPRESSION>

<RELATION> ::=                  <  /  ≤  /  =  /  ≠  /  ≥  /  >  /
                                LSS  /  LEQ  /  EQL  /  NEQ  /
                                GEQ  /  GTR
```

```
<ARITHMETIC
EXPRESSION> ::=              <TERM>
                           / <TERM>
                             <ADDITIVE OPERATOR>
                             <ARITHMETIC EXPRESSION>

<ADDITIVE OPERATOR> ::=      +  /  -

<TERM> ::=                   <SIGNED PRIMARY>
                           / <SIGNED PRIMARY>
                             <MULTIPLICATIVE OPERATOR>
                             <TERM>

<MULTIPLICATIVE
OPERATOR> ::=                *  /  MOD  /  <SLASH>

<SIGNED PRIMARY> ::=         <PRIMARY>
                           / <UNARY OPERATOR>
                             <PRIMARY>

<UNARY OPERATOR> ::=         + / -

<PRIMARY> ::=                <CONSTANT>                %DEFINED ABOVE
                           / <VARIABLE>
                           / (<EXPRESSION>)            %DEFINED ABOVE
                           / <CONDITIONAL EXPRESSION>
                           / <CASE EXPRESSION>
                           / <BUMPOR>
                           / <DECREMENTOR>
                           / <ASSIGNOR>

<VARIABLE> ::=               <ADDRESS VARIABLE>
                           / <VALUE VARIABLE>

<ADDRESS VARIABLE> ::=       <SIMPLE VARIABLE>
                           / <SUBSCRIPTED VARIABLE>
                           / <INDEXED VARIABLE>
                           / <ADDRESS-GENERATING FUNCTION DESIGNATOR>

<SIMPLE VARIABLE> ::=        <SIMPLE IDENTIFIER>

<SIMPLE IDENTIFIER> ::=      <IDENTIFIER>

<SUBSCRIPTED VARIABLE> ::=   <ARRAY IDENTIFIER>
                             (<EXPRESSION>)          %DEFINED ABOVE

<ARRAY IDENTIFIER ::=        <IDENTIFIER>            %DEFINED ABOVE

<INDEXED VARIABLE> ::=       <SIMPLE IDENTIFIER>
                             <INDEX PART>
                           / <ARRAY IDENTIFIER>
                             <INDEX PART>
```

```
<INDEX PART> ::=                [<EXPRESSION LIST>]      %DEFINED ABOVE

<ADDRESS-GENERATING
FUNCTION DESIGNATOR> ::=        <SUB-STRING ADDRESS DESIGNATOR>
                              / <FETCH COMMUNICATE MESSAGE
                                POINTER DESIGNATOR>
                              / <DESCRIPTOR DESIGNATOR>
                              / <DESCRIPTOR-GENERATOR DESIGNATOR>
                              / <ADDRESS-MODIFIER DESIGNATOR>

<SUB-STRING ADDRESS
DESIGNATOR> ::=                 <SUB-STRING FUNCTION IDENTIFIER>
                                (<STRING ADDRESS>,<OFFSET PART>)
                              / <SUB-STRING FUNCTION IDENTIFIER>
                                (<STRING ADDRESS>,<OFFSET PART>,
                                <LENGTH PART>)

<SUB-STRING FUNCTION
IDENTIFIER> ::=                 SUBSTR
                              / SUBBIT

<OFFSET PART> ::=               <EXPRESSION>

<LENGTH PART> ::=               <EXPRESSION>

<STRING ADDRESS> ::=            <ADDRESS GENERATOR>

<ADDRESS GENERATOR
LIST> ::=                       <ADDRESS GENERATOR>
                              / <ADDRESS GENERATOR>,
                                <ADDRESS GENERATOR LIST>

<ADDRESS GENERATOR> ::=         <ADDRESS VARIABLE>       %DEFINED ABOVE
                              / <BUMPOR>
                              / <DECREMENTOR>
                              / <CONDITIONAL ADDRESS GENERATOR>
                              / <CASE ADDRESS GENERATOR>
                              / <ADDRESS-GENERATING ASSIGNOR>

<BUMPOR> ::=                    BUMP <ADDRESS VARIABLE>
                                <MODIFIER>

<MODIFIER> ::=                  <EMPTY>
                              / BY <EXPRESSION>

<DECREMENTOR> ::=               DECREMENT <ADDRESS VARIABLE>
                                <MODIFIER>

<CONDITIONAL ADDRESS
GENERATOR> ::=                  IF <EXPRESSION>
                                THEN <ADDRESS GENERATOR>
                                ELSE <ADDRESS GENERATOR>

<CASE ADDRESS
```

```
GENERATOR> ::=                  CASE <EXPRESSION>
                                OF (<ADDRESS GENERATOR LIST>)

<ADDRESS-GENERATING
ASSIGNOR> ::=                   <ADDRESS VARIABLE>
                                <REPLACE, DELETE LEFT PART>
                                <ADDRESS GENERATOR>
                               / <ADDRESS VARIABLE>
                                <REPLACE, DELETE RIGHT PART>
                                <EXPRESSION>

<ASSIGNOR> ::=                   <ADDRESS VARIABLE>
                                 <NON-DESTRUCTIVE REPLACE>
                                 <EXPRESSION>

<NON-DESTRUCTIVE
REPLACE> ::=                    <REPLACE, DELETE LEFT PART>
                               / <REPLACE, DELETE RIGHT PART>

<REPLACE, DELETE
LEFT PART> ::=                 ←/ :=

<REPLACE, DELETE
RIGHT PART> ::=                 :←/    ::=

<FETCH COMMUNICATE MESSAGE
POINTER DESIGNATOR> ::=         FETCH.COMMUNICATE.MSG.PTR

<DESCRIPTOR DESIGNATOR> ::=    DESCRIPTOR (<SIMPLE IDENTIFIER>)
                              / DESCRIPTOR (<ARRAY IDENTIFIER>)

<DESCRIPTOR-GENERATOR
DESIGNATOR> ::=                 MAKE.DESCRIPTOR (<EXPRESSION>)

<ADDRESS-MODIFIER
DESIGNATOR> ::=                 <ADDRESS-MODIFIER FUNCTION IDENTIFIER>
                                (<SIMPLE IDENTIFIER>)

<ADDRESS-MODIFIER
FUNCTION IDENTIFIER> ::=        NEXT.ITEM
                               / PREVIOUS.ITEM

<VALUE VARIABLE> ::=            <VALUE-GENERATING FUNCTION DESIGNATOR>
                               / <TYPED PROCEDURE DESIGNATOR>
                               / (<ADDRESS VARIABLE>)
                               / <FILE DESIGNAROR>

<VALUE-GENERATING
FUNCTION DESIGNATOR> ::=        <SWAP DESIGNATOR>
                               / <SUB-STRING VALUE DESIGNATOR>
                               / <DISPATCH DESIGNATOR>
                               / <LOCATION DESIGNATOR>
                               / <CONVERT DESIGNATOR>
                               / <LENGTH DESIGNATOR>
```

```
                                / <MEMORY SIZE DESIGNATOR>
                                / <DESCRIPTOR-VALUE-GENERATOR DESIGNATOR>
                                / <INTERROGATE INTERRUPT STATUS DESIGNATOR>
                                / <DECIMAL CONVERSION DESIGNATOR>
                                / <BINARY CONVERSION DESIGNATOR>
                                / <TIME FUNCTION DESIGNATOR>
                                / <DATE FUNCTION DESIGNATOR>
                                / <NAME-OF-DAY FUNCTION DESIGNATOR>
                                / <BASE REGISTER DESIGNATOR>
                                / <LIMIT REGISTER DESIGNATOR
                                / <CONTROL STACK TOP DESIGNATOR>
                                / <DATA ADDRESS DESIGNATOR>
                                / <SEARCH.LINKED.LIST DESIGNATOR>
                                / <SORT.STEP.DOWN DESIGNATOR>
                                / <SORT.UNBLOCK DESIGNATOR>
                                / <SORT.SEARCH DESIGNATOR>
                                / <PARITY.ADDRESS DESIGNATOR>
                                / <DYNAMIC MEMORY BASE DESIGNATOR>
                                / <HASH CODE DESIGNATOR>
                                / <NEXT TOKEN DESIGNATOR>
                                / <DELIMITED TOKEN DESIGNATOR>
                                / <EVALUATION STACK TOP DESIGNATOR>
                                / <CONTROL STACK BITS DESIGNATOR>
                                / <NAME STACK TOP DESIGNATOR>
                                / <DISPLAY BASE DESIGNATOR>
                                / <CONSOLE SWITCHES DESIGNATOR>
                                / <SEARCH SERIAL LIST DISIGNATOR>
                                / <SPO INPUT PRESENT DESIGNATOR>
                                / <SEARCH.SDL.STACKS DESIGNATOR>
                                / <EXECUTE DESIGNATOR>

<SWAP DESIGNATOR> ::=           SWAP (<ADDRESS GENERATOR>,<EXPRESSION>)

<SUB-STRING VALUE
DESIGNATOR> ::=                 <SUBSTRING FUNCTION IDENTIFIER>
                                (<STRING VALUE>,<OFFSET PART>)
                                / <SUBSTRING FUNCTION IDENTIFIER>
                                (<STRING VALUE>,<OFFSET PART>
                                <LENGTH PART>)

<STRING VALUE> ::=              <EXPRESSION>

<DISPATCH DESIGNATOR> ::=       (<PORT,CHANNEL,PRIORITY>,
                                <I/O DESCRIPTOR ADDRESS>)

<PORT,CHANNEL,PRIORITY ::=      <EXPRESSION>

<I/O DESCRIPTOR
ADDRESS> ::=                    <EXPRESSION>

<LOCATION DESIGNATOR> ::=       LOCATION (<LOCATION PARAMETER PART>)

<LOCATION PARAMETER
PART>::=                        <PROCEDURE IDENTIFIER>
```

```
                                / <SIMPLE IDENTIFIER>
                                / <ARRAY IDENTIFIER>

<CONVERT DESIGNATOR> ::=          CONVERT (<EXPRESSION>,
                                  <CONVERSION PART>)
                                / CONV (<EXPRESSION>,
                                  <CONVERSION PART>)

<CONVERSION PART> ::=             <CONVERSION TYPE>
                                / <CONVERSION TYPE>,
                                  <BIT GROUP SIZE>

<CONVERSION TYPE> ::=             BIT / FIXED / CHARACTER

<BIT GROUP SIZE> ::=              1 / 2 / 3 / 4

<LENGTH DESIGNATOR> ::=           LENGTH (<EXPRESSION>)

<MEMORY SIZE
DESIGNATOR> ::=                   S.MEM.SIZE
                                / M.MEM.SIZE

<DESCRIPTOR-VALUE-
GENERATOR DESIGNATOR> ::=         VALUE.DESCRIPTOR (<ADDRESS GENERATOR>)

<INTERROGATE INTERRUPT
STATUS DESIGNATOR> ::=            INTERROGATE.INTERRUPT.STATUS

<DECIMAL CONVERSION
DESIGNATOR> ::=                   DECIMAL(<EXPRESSION>,<DECIMAL STRING SIZE>)

<DECIMAL STRING SIZE> ::=         <EXPRESSION>

<BINARY CONVERSION
DESIGNATOR> ::=                   BINARY (<EXPRESSION>)

<TIME FUNCTION
DESIGNATOR> ::=                   TIME
                                / TIME (<TIME FORMAT>,<REPRESENTATION>)

<TIME FORMAT> ::=                 COUNTER/MILITARY/CIVILIAN

<REPRESENTATION> ::=              BIT/DIGIT/CHARACTER

<BASE REGISTER
DESIGNATOR> ::=                   BASE.REGISTER

<LIMIT REGISTER
DESIGNATOR> ::=                   LIMIT.REGISTER

<CONTROL STACK
TOP DESIGNATOR> ::=               CONTROL.STACK.TOP

<DATE FUNCTION
```

```
DESIGNATOR> ::=                 DATE
                              / DATE (<DATE FORMAT>,<REPRESENTATION>)

<DATA FORMAT> ::=               JULIAN / MONTH / DAY / YEAR

<NAME-OF-DAY
FUNCTION DESIGNATOR> ::=        NAME.OF.DAY

<DATA ADDRESS
DESIGNATOR> ::=                 DATA.ADDRESS(<ADDRESS GENERATOR>)

<SEARCH.LINKED.LIST
DESIGNATOR> ::=                 SEARCH.LINKED.LIST
                                (<RECORD ADDRESS>,<ARGUMENT INDEX>,
                                <COMPARE VARIABLE>,<RELATION>,
                                <LINK INDEX>)

<RECORD ADDRESS> ::=            <EXPRESSION>

<ARGUMENT INDEX> ::=            <EXPRESSION>

<COMPARE VARIABLE> ::=          <EXPRESSION>

<RELATION> ::=                  < / ≤ / = / ≠ / ≥ / > /
                                LSS / LEQ / EQL / NEQ /
                                GEQ / GTR

<LINK INDEX> ::=                <EXPRESSION>

<SORT.STEP.DOWN
DESIGNATOR> ::=                 SORT.STEP.DOWN
                                (<RECORD 1>,<RECORD 2>,
                                <KEY TABLE ADDRESS>)

<RECORD 1> ::=                  <EXPRESSION>

<RECORD 2> ::=                  <EXPRESSION>

<KEY TABLE ADDRESS> ::=         <EXPRESSION>

<SORT.UNBLOCK
DESIGNATOR> ::=                 SORT.UNBLOCK (<MINI FIB ADDRESS>,
                                <LENGTH>,<SOURCE>,<DESTINATION>)

<MINI FIB ADDRESS> ::=          <ADDRESS GENERATOR>

<LENGTH> ::=                    <EXPRESSION>

<SOURCE> ::=                    <EXPRESSION>

<DESTINATION> ::=               <EXPRESSION>

<SORT.SEARCH
DESIGNATOR> ::=                 SORT.SEARCH
```

```
                                  (<TABLE ADDRESS>,<LIMIT>)

<TABLE ADDRESS> ::=               <ADDRESS GENERATOR>

<LIMIT> ::=                       <EXPRESSION>

<PARITY.ADDRESS
DESIGNATOR> ::=                   PARITY.ADDRESS

<DYNAMIC MEMORY
BASE DESIGNATOR> ::=              DYNAMIC.MEMORY.BASE

<HASH CODE DESIGNATOR>::=         HASH.CODE (<TOKEN>)

<TOKEN>::=                        <EXPRESSION>

<NEXT TOKEN DESIGNATOR>::=        NEXT.TOKEN (<FIRST CHARACTER>,
                                  <SEPARATOR>, NUMERIC-TO-ALPHA INDICATOR>,
                                  <RESULT>)

<NUMERIC-TO-ALPHA
INDICATOR>::=                       SET
                                  / RESET

<DELIMITED TOKEN
DESIGNATOR>::=                    DELIMITED.TOKEN (<FIRST CHARACTER>,
                                  <DELIMITERS>, <RESULT>)

<FIRST CHARACTER>::=              <CHARACTER STRING>
                                  / <BIT STRING>

<RESULT>::=                       <IDENTIFIER>

<EVALUATION STACK
TOP DESIGNATOR>::=                EVALUATION.STACK.TOP

<CONTROL STACK
BITS DESIGNATOR>::=               CONTROL.STACK.BITS

<NAME STACK
TOP DESIGNATOR>::=                NAME.STACK.TOP

<DISPLAY BASE
DESIGNATOR>::=                    DISPLAY.BASE

<CONSOLE SWITCHES
DESIGNATOR>::=                    CONSOLE.SWITCHES

<SEARCH SERIAL
LIST DESIGNATOR>::=               SEARCH.SERIAL.LIST (<SSL COMPARE VALUE>,
                                  <SSL COMPARE TYPE>, <SSL COMPARE FIELD>,
                                  <SSL FIRST ITEM>, <SSL TABLE LENGTH>,
                                  <SSL RESULT VARIABLE>)
```

```
<SSL COMPARE VALUE>::=          <EXPRESSION>

<SSL COMPARE TYPE>::=           </≤/=/≠/≥/>

<SSL COMPARE FIELD>::=          <EXPRESSION>

<SSL FIRST ITEM>::=             <EXPRESSION>

<SSL TABLE LENGTH>::=           <EXPRESSION>

<SSL RESULT VARIABLE>::=        <ADDRESS GENERATOR>

<SPO INPUT
PRESENT DESIGNATOR>::=          SPO.INPUT.PRESENT

<SEARCH.SDL.STACKS
DESIGNATOR>::=                  SEARCH.SDL.STACKS
                                (<STACK BASE>, <STACK TOP>,
                                <COMPARE BASE>, <COMPARE TOP>)

<STACK BASE>::=                 <EXPRESSION>

<STACK TOP>::=                  <EXPRESSION>

<COMPARE BASE>::=               <EXPRESSION>

<COMPARE TOP>::=                <EXPRESSION>

<EXECUTE DESIGNATOR>::=         EXECUTE (<EXPRESSION LIST>)

<TYPED PROCEDURE
DESIGNATOR> ::=                 <TYPED PROCEDURE IDENTIFIER>
                                <ACTUAL PARAMETER PART>

<TYPED PROCEDURE
IDENTIFIER> ::=                 <IDENTIFIER>

<ACTUAL PARAMETER PART> ::=     <EMPTY>
                              / (<ACTUAL PARAMETER LIST>)

<ACTUAL PARAMETER LIST> ::=     <ACTUAL PARAMETER>
                              / <ACTUAL PARAMETER>,
                                <ACTUAL PARAMETER LIST>

<ACTUAL PARAMETER> ::=          <EXPRESSION>
                              / <ARRAY DESIGNATOR>

<ARRAY DESIGNATOR> ::=          <ARRAY IDENTIFIER>        %DEFINED ABOVE

<CONDITIONAL
EXPRESSION> ::=                 IF <EXPRESSION>
                                THEN <EXPRESSION>
                                ELSE <EXPRESSION>
```

```
<CASE EXPRESSION> ::=          CASE <EXPRESSION>
                               OF (<EXPRESSION LIST>)

<CASE STATEMENT> ::=           <CASE HEAD>
                               <CASE BODY>

<CASE HEAD> ::=                CASE <EXPRESSION>;

<CASE BODY> ::=                <EXECUTABLE STATEMENT LIST>
                               <CASE ENDING>

<CASE ENDING> ::=              END CASE

<ASSIGNMENT STATEMENT> ::=     <ADDRESS VARIABLE>
                               <REPLACE>
                               <EXPRESSION>

<REPLACE> ::=                  <-/  :=

<EXECUTE-PROCEDURE
STATEMENT> ::=                 <NON-TYPED PROCEDURE DESIGNATOR>

<NON-TYPED PROCEDURE
DESIGNATOR> ::=                <NON-TYPED PROCEDURE IDENTIFIER>
                               <ACTUAL PARAMETER PART>

<NON-TYPED PROCEDURE
IDENTIFIER> ::=                <IDENTIFIER>

<EXECUTE-FUNCTION
STATEMENT> ::=                 <FUNCTION DESIGNATOR>

<FUNCTION DESIGNATOR> ::=      <DUMP DESIGNATOR>
                             / <TRACE DESIGNATOR>
                             / <SAVE DESIGNATOR>
                             / <RESTORE DESIGNATOR>
                             / <FETCH DESIGNATOR>
                             / <HALT DESIGNATOR>
                             / <REINSTATE DESIGNATOR>
                             / <ACCESS-FPB DESIGNATOR>
                             / <REVERSE STORE DESIGNATOR>
                             / <READ CASSETTE DESIGNATOR>
                             / <ACCESS OVERLAY DESIGNATOR>
                             / <ERROR COMMUNICATE DESIGNATOR>
                             / <SORT DESIGNATOR>
                             / <OVERLAY DESIGNATOR>
                             / <HARDWARE MONITOR DESIGNATOR>
                             / <SAVE STATE DESIGNATOR>
                             / <SORT.SWAP DESIGNATOR>
                             / <INITIALIZE.VECTOR DESIGNATOR>
                             / <THREAD.VECTOR DESIGNATOR>
                             / <ENABLE.INTERRUPTS DESIGNATOR>
                             / <DISABLE.INTERRUPTS DESIGNATOR>
                             / <ACCESS FILE INFORMATION
```

```
                                DESIGNATOR>
                              / <DEBLANK DESIGNATOR>
                              / <FREEZE-PROGRAM DESIGNATOR>
                              / <THAW-PROGRAM DESIGNATOR>
                              / <DUMP-FOR-ANALYSIS DESIGNATOR>
                              / <COMPILE-CARD-INFO DESIGNATOR>
                              / <COMMUNICATE DESIGNATOR>

<DUMP DESIGNATOR> ::=           DUMP

<TRACE DESIGNATOR> ::=          TRACE / NOTRACE
                              / TRACE (<EXPRESSION>)

<SAVE DESIGNATOR> ::=           SAVE (<EXPRESSION LIST>)

<RESTORE DESIGNATOR> ::=        RESTORE (<ADDRESS GENERATOR LIST>)

<FETCH DESIGNATOR> ::=          (<I/O REFERENCE ADDRESS>,
                              / <PORT,CHANNEL,PRIORITY ADDRESS>,
                                <RESULT DESCRIPTOR ADDRESS>)

<I/O REFERENCE ADDRESS> ::=   <EXPRESSION>

<PORT,CHANNEL,
PRIORITY ADDRESS> ::=           <ADDRSS GENERATOR>

<RESULT DESCRIPTOR
ADDRESS> ::=                    <ADDRESS GENERATOR>

<HALT DESIGNATOR> ::=           HALT (<EXPRESSION>)

<REINSTATE DESIGNATOR> ::=    REINSTATE (<REINSTATED PROGRAM>)

<REINSTATED PROGRAM> ::=        <ADDRESS GENERATOR>

<ACCESS-FPB DESIGNATOR> ::=   <ACCESS-FPB IDENTIFER>
                                (<FILE SPECIFIER>,
                                <SOURCE OR DESTINATION FIELD>)

<ACCESS FPB IDENTIFIER> ::=   READ.FPB / WRITE.FPB

<FILE SPECIFIER> ::=            <FILE DESIGNATOR>
                              / <FILE NUMBER>

<FILE NUMBER> ::=               <EXPRESSION>

<SOURCE OR DESTINATION
FIELD> ::=                      <ADDRESS GENERATOR>

<REVERSE STORE
DESIGNATOR> ::=                 REVERSE.STORE (<ADDRESS GENERATOR LIST>,
                                <EXPRESSION>)

<READ CASSETTE
```

```
DESIGNATOR>::=                  READ.CASSETTE (<DESTINATION SPECIFIER,
                                <HASH.TOTAL SPECIFIER>, <RESULT SPECIFIER

<DESTINATION SPECIFIER>::=      <ADDRESS GENERATOR>

<HASH.TOTAL SPECIFIER>::=       HASH.TOTAL
                              / NO.HASH.TOTAL

<RESULT SPECIFIER>::=           <ADDRESS GENERATOR>

<ACCESS OVERLAY
DESIGNATOR> ::=                 <ACCESS OVERLAY IDENTIFIER>(<EXPRESSION>)

<ACCESS OVERLAY
IDENTIFIER> ::=                 READ.OVERLAY / WRITE.OVERLAY

<ERROR COMMUNICATE
DESIGNATOR> ::=                 ERROR.COMMUNICATE (<EXPRESSION>)

<SORT DESIGNATOR> ::=           SORT (<SORT-INFORMATION-TABLE SPECIFIER>,
                                <SORT-KEY-TABLE SPECIFIER>,
                                <INPUT FILE DESIGNATOR>,
                                <OUTPUT FILE DESIGNATOR>)

<SORT-INFORMATION-TABLE
SPECIFIER> ::=                  <ADDRESS GENERATOR>

<SORT-KEY-TABLE
SPECIFIER> ::=                  <ADDRESS GENERATOR>

<INPUT FILE DESIGNATOR>::=      <FILE DESIGNATOR>

<OUTPUT FILE
DESIGNATOR> ::=                 <FILE DESIGNATOR>

<OVERLAY DESIGNATOR> ::=        OVERLAY (<EXPRESSION>)

<HARDWARE MONITOR
DESIGNATOR> ::=                 HARDWARE.MONITOR (<EXPRESSION>)

<SAVE STATE DESIGNATOR> ::=     SAVE.STATE

<SORT.SWAP DESIGNATOR> ::=      SORT.SWAP (<RECORD 1>,<RECORD 2>)

<RECORD 1> ::=                  <ADDRESS GENERATOR>

<RECORD 2> ::=                  <ADDRESS GENERATOR>

<INITIALIZE.VECTOR
DESIGNATOR> ::=                 INITIALIZE.VECTOR (<TABLE ADDRESS>)

<TABLE ADDRESS> ::=             <ADDRESS GENERATOR>

<THREAD.VECTOR
```

```
DESIGNATOR> ::=                 THREAD.VECTOR (<TABLE ADDRESS>,<INDEX>)

<TABLE ADDRESS> ::=             <ADDRESS GENERATOR>

<INDEX> ::=                     <EXPRESSION>

<ENABLE.INTERRUPTS
DESIGNATOR> ::=                 ENABLE.INTERRUPTS

<DISABLE.INTERRUPTS
DESIGNATOR> ::=                 DISABLE.INTERRUPTS

<ACCESS FILE INFORMATION
DESIGNATOR> ::=                 ACCESS.FILE.INFORMATION (<FILE
                                DESIGNATOR>, <RETURN TYPE>, <DESTINATION>)

<FILE DESIGNATOR> ::=             <FILE IDENTIFIER>
                                / <SWITCH FILE IDENTIFIER> (<EXPRESSION>)

<RETURN TYPE> ::=               BIT / CHARACTER

<DESTINATION> ::=               <ADDRESS GENERATOR>

<DEBLANK DESIGNATOR>::=         DEBLANK (<FIRST CHARACTER>)

<FIRST CHARACTER>::=            <IDENTIFIER>

<FREEZE-PROGRAM
DESIGNATOR>::=                  FREEZE.PROGRAM

<THAW-PROGRAM
DESIGNATOR>::=                  THAW.PROGRAM

<DUMP-FOR-
ANALYSIS DESIGNATOR>::=         DUMP.FOR.ANALYSIS

<COMPILE-CARD-
INFO DESIGNATOR>::=             COMPILE.CARD.INFO
                                (<CCI DESTINATION FIELD>)

<CCI DESTINATION FIELD>::=      <ADDRESS GENERATOR>

<COMMUNICATE DESIGNATOR>::=     COMMUNICATE (<EXPRESSION>)

<GROUP TERMINATION
STATEMENT> ::=                    UNDO
                                / UNDO (*)
                                / UNDO <GROUP IDENTIFIER>  %DEFINED ABOVE

<I/O CONTROL STATEMENT> ::=     <OPEN STATEMENT>;
                                / <CLOSE STATEMENT>;
                                / <READ STATEMENT>
                                / <WRITE STATEMENT>
                                / <SEEK STATEMENT>;
```

```
                          / <ACCEPT STATEMENT>;
                          / <DISPLAY STATEMENT>;
                          / <SPACE STATEMENT>
                          / <SKIP STATEMENT>;

<OPEN STATEMENT>::=         <OPEN PART>
                          / <OPEN PART>; <FILE MISSING PART>
                          / <OPEN PART>; <FILE LOCKED PART>
                          / <OPEN PART>; <FILE MISSING PART>
                            <FILE LOCKED PART>

<OPEN PART> ::=             OPEN <FILE DESIGNATOR>
                            <OPEN ATTRIBUTE PART>

<FILE DESIGNATOR>::=        <FILE IDENTIFIER>
                          / <SWITCH FILE IDENTIFIER> (<EXPRESSION>)

<OPEN ATTRIBUTE PART> ::=   <EMPTY>
                          / <OPEN ATTRIBUTE LIST>
                          / WITH <OPEN ATTRIBUTE LIST>

<OPEN ATTRIBUTE LIST> ::=   <OPEN ATTRIBUTE>
                          / <OPEN ATTRIBUTE> <ATTRIBUTE SEPARATOR>
                            <OPEN ATTRIBUTE LIST>

<ATTRIBUTE SEPARATOR>::=    , / <SLASH> / <EMPTY>

<OPEN ATTRIBUTE> ::=        <INPUT-OUTPUT MODE>
                          / <LOCK MODE>
                          / <OPEN ACTION MODE>
                          / <CODE FILE MODE>
                          / <MFCU MODE>

<INPUT-OUTPUT MODE> ::=     INPUT / OUTPUT / NEW

<LOCK MODE> ::=             LOCK / LOCK.OUT

<OPEN ACTION MODE> ::=      NO.REWIND / REVERSE

<CODE FILE MODE> ::=        CODE.FILE

<MFCU MODE>::=              PUNCH / PRINT /
                            INTERPRET / STACKERS

<CLOSE STATEMENT>::=        CLOSE <FILE DESIGNATOR> % DEFINED ABOVE
                            <CLOSE ATTRIBUTE PART>

<CLOSE ATTRIBUTE PART> ::=  <EMPTY>
                          / <CLOSE ATTRIBUTE LIST>
                          / WITH <CLOSE ATTRIBUTE LIST>

<CLOSE ATTRIBUTE LIST> ::=  <CLOSE ATTRIBUTE>
                          / <CLOSE ATTRIBUTE> <ATTRIBUTE SEPARATOR>
```

```
                                <CLOSE ATTRIBUTE LIST>

<ATTRIBUTE SEPARATOR>::=        , / <SLASH> / <EMPTY>

<CLOSE ATTRIBUTE> ::=           <CLOSE MODE>
                              / CRUNCH / ROLLOUT / IF.NOT.CLOSED

<CLOSE MODE> ::=                REEL / RELEASE / PURGE / REMOVE
                              / NO.REWIND / LOCK / CODE.FILE

<READ STATEMENT> ::=            <READ PART>;
                              / <READ PART>;<EOF PART>
                              / <READ PART>; <EXCEPTION PART>
                              / <READ PART>; <EOF PART> <EXCEPTION PART>

<READ PART> ::=                 <READ SPECIFIER>
                              / <DISK READ SPECIFIER>
                              / <REMOTE READ SPECIFIER>
                              / <QUEUE READ SPECIFIER>

<READ SPECIFIER> ::=            READ <FILE DESIGNATOR>
                                (<ADDRESS GENERATOR>)

<DISK READ SPECIFIER> ::=       READ <RECORD LOCK PART>
                                <FILE DESIGNATOR>
                                <RECORD ADDRESS PART>
                                (<ADDRESS GENERATOR>)

<RECORD LOCK PART> ::=          <EMPTY> / LOCK

<RECORD ADDRESS PART> ::=       <EMPTY>
                              / [<RECORD ADDRESS>]

<RECORD ADDRESS> ::=            <EXPRESSION>

<REMOTE READ SPECIFIER>::=      READ <FILE DESIGNATOR>
                                <REMOTE KEY PART>
                                (<ADDRESS GENERATOR>)

<REMOTE KEY PART>::=            <EMPTY>
                              / [<REMOTE KEY>]

<REMOTE KEY>::=                 <ADDRESS GENERATOR>

<QUEUE READ SPECIFIER> ::=      READ <FILE DESIGNATOR>
                                <QUEUE FAMILY MEMBER PART>
                                (<ADDRESS GENERATOR>)

<QUEUE FAMILY
MEMBER PART>::=                 <EMPTY>
                              / [<QUEUE FAMILY MEMBER>]

<QUEUE FAMILY MEMBER>::=        <EXPRESSION>
```

```
<EOF PART> ::=                  ON EOF <EXECUTABLE STATEMENT>

<EXCEPTION PART>::=             ON EXCEPTION <EXECUTABLE STATEMENT>

<WRITE STATEMENT> ::=             <WRITE PART>;
                                / <WRITE PART>;<EOF PART>
                                / <WRITE PART>;<EXCEPTION PART>
                                / <WRITE PART>;<EOF PART> <EXCEPTION PART>

<WRITE PART> ::=                  <WRITE SPECIFIER>
                                / <DISK WRITE SPECIFIER>
                                / <REMOTE WRITE SPECIFIER>
                                / <QUEUE WRITE SPECIFIER>

<WRITE SPECIFIER> ::=             WRITE <FILE DESIGNATOR>
                                  <CARRIAGE CONTROL PART>
                                  (<EXPRESSION>)
                                / WRITE <FILE DESIGNATOR>
                                  <CARRIAGE CONTROL PART>

<CARRIAGE CONTROL PART> ::=     <EMPTY>
                                / <CARRIAGE CONTROL SPECIFIER>

<CARRIAGE CONTROL
SPECIFIER> ::=                    NO / SINGLE / DOUBLE / PAGE
                                / <SKIP-TO-CHANNEL> / NEXT

<SKIP-TO-CHANNEL> ::=             <CHANNEL NUMBER>

<CHANNEL NUMBER> ::=              1 / 2 / 3 / ... / 11 / 12

<DISK WRITE SPECIFIER> ::=      WRITE <RECORD LOCK PART>
                                  <FILE DESIGNATOR>
                                  <RECORD ADDRESS PART>
                                  (<EXPRESSION>)

<REMOTE WRITE
SPECIFIER>::=                     WRITE <FILE DESIGNATOR>
                                  <REMOTE KEY PART>
                                  (<EXPRESSION>)

<REMOTE KEY PART>::=              <EMPTY>
                                / [<REMOTE KEY>]

<REMOTE KEY>::=                   <ADDRESS GENERATOR>

<QUEUE WRITE
SPECIFIER>::=                     WRITE <FILE DESIGNATOR>
                                  <QUEUE FAMILY MEMBER PART>
                                  (<ADDRESS GENERATOR>)

<QUEUE FAMILY
MEMBER PART>::=                   <EMPTY>
                                / [<QUEUE FAMILY MEMBER>]
```

```
<QUEUE FAMILY MEMBER>::=      <EXPRESSION>

<EXCEPTION PART> ::=          ON EXCEPTION <EXECUTABLE STATEMENT>

<SEEK STATEMENT> ::=          SEEK <RECORD LOCK PART>
                              <FILE DESIGNATOR>
                              [<RECORD ADDRESS>]

<ACCEPT STATEMENT> ::=        ACCEPT <ADDRESS GENERATOR>
                              <END-OF-TEXT SPECIFIER>

<END-OF-TEXT
SPECIFIER> ::=                  <EMPTY>
                            / , END.OF.TEXT

<DISPLAY STATEMENT> ::=       DISPLAY <EXPRESSION>
                              <CRUNCH SPECIFIER>

<CRUNCH SPECIFIER> ::=          <EMPTY>
                            / , CRUNCHED

<SPACE STATEMENT> ::=           <SPACE PART>;
                            / <SPACE PART>; <EOF PART>
                            / <SPACE PART>; <EXCEPTION PART>
                            / <SPACE PART>; <EOF PART> <EXCEPTION PART>

<SPACE PART> ::=              SPACE <FILE DESIGNATOR>
                              <SPACING SPECIFIER>

<SPACING SPECIFIER> ::=       <EXPRESSION> / TO <EXPRESSION>

<SKIP STATEMENT> ::=          SKIP <FILE DESIGNATOR> TO <CHANNEL NUMBER>

<MODIFY INSTRUCTION> ::=      <CLEAR STATEMENT>
                            / <BUMP STATEMENT>
                            / <DECREMENT STATEMENT>

<CLEAR STATEMENT> ::=         CLEAR <ARRAY IDENTIFIER LIST>

<ARRAY IDENTIFIER LIST> ::=   <ARRAY IDENTIFIER>       %DEFINED ABOVE
                            / <ARRAY IDENTIFIER>,
                            / <ARRAY IDENTIFIER LIST>

<BUMP STATEMENT> ::=          BUMP <ADDRESS VARIABLE> <MODIFIER>

<DECREMENT STATEMENT> ::=     DECREMENT <ADDRESS VARIABLE> <MODIFIER>

<NULL STATEMENT> ::=          ;

<STOP STATEMENT> ::=          STOP / STOP <EXPRESSION>


<FILE ATTRIBUTE
```

```
STATEMENT> ::=                  CHANGE <FILE DESIGNATOR>
                                TO (<DYNAMIC FILE ATTRIBUTE LIST>)

<DYNAMIC FILE
ATTRIBUTE LIST> ::=               <DYNAMIC FILE ATTRIBUTE>
                                / <DYNAMIC FILE ATTRIBUTE>,
                                  <DYNAMIC FILE ATTRIBUTE LIST>

<DYNAMIC FILE
ATTRIBUTE> ::=                    <DYNAMIC MULTI-FILE IDENTIFICATION PART>
                                / <DYNAMIC FILE IDENTIFICATION PART>
                                / <DYNAMIC DEVICE PART>
                                / <DYNAMIC FILE PARITY PART>
                                / <DYNAMIC TRANSLATION PART>
                                / <DYNAMIC BUFFERS PART>
                                / <DYNAMIC VARIABLE RECORD PART>
                                / <DYNAMIC LOCK PART>
                                / <DYNAMIC BUFFERS PART>
                                / <DYNAMIC SAVE FACTOR PART>
                                / <DYNAMIC RECORD SIZE PART>
                                / <DYNAMIC RECORDS-PER-BLOCK PART>
                                / <DYNAMIC REEL NUMBER PART>
                                / <DYNAMIC NUMBER-OF-AREAS PART>
                                / <DYNAMIC BLOCKS-PER-AREA PART>
                                / <DYNAMIC PACK.ID PART>
                                / <DYNAMIC ALL-AREAS-AT-OPEN PART>
                                / <DYNAMIC AREA-BY-CYLINDER PART>
                                / <DYNAMIC EU.SPECIAL PART>
                                / <DYNAMIC EU.INCREMENTED PART>
                                / <DYNAMIC USE.INPUT.BLOCKING PART>
                                / <DYNAMIC SORTER-STATION PART>
                                / <DYNAMIC MULTI-PACK PART>
                                / <DYNAMIC END-OF-PAGE PART>
                                / <DYNAMIC OPEN-OPTION PART>
                                / <DYNAMIC REMOTE-KEY PART>
                                / <DYNAMIC NUMBER-OF-STATIONS PART>
                                / <DYNAMIC QUEUE-FAMILY-SIZE PART>
                                / <DYNAMIC FILE TYPE PART>
                                / <DYNAMIC WORK FILE PART>
                                / <DYNAMIC LABEL TYPE PART>

<DYAMIC MULTI-FILE
IDENTIFICATION PART> ::=        MULTI.FILE.ID ←
                                <DYNAMIC MULTI-FILE IDENTIFICATION>

<DYNAMIC MULTI-FILE
IDENTIFICATION> ::=             <EXPRESSION>

<DYNAMIC FILE
IDENTIFICATION PART> ::=        FILE.ID←<DYNAMIC FILE IDENTIFICATION>

<DYNAMIC FILE
IDENTIFICATION> ::=             <EXPRESSION>
```

```
<DYNAMIC DEVICE PART> ::=     DEVICE←<DYNAMIC DEVICE SPECIFIER>

<DYNAMIC DEVICE
SPECIFIER> ::=                <EXPRESSION>

<DYNAMIC FILE
PARITY PART> ::=              PARITY←<DYNAMIC PARITY SPECIFIER>

<DYNAMIC PARITY
SPECIFIER> ::=                <EXPRESSION>

<DYNAMIC TRANSLATION
PART> ::=                     TRANSLATION←
                              <DYNAMIC TRANSLATION SPECIFIER>

<DYNAMIC TRANSLATION
SPECIFIER> ::=                <EXPRESSION>

<DYNAMIC BUFFERS PART> ::=    BUFFERS←<DYAMIC NUMBER OF BUFFERS>

<DYNAMIC NUMBER
OF BUFFERS> ::=               <EXPRESSION>

<DYNAMIC VARIABLE
RECORD PART> ::=              VARIABLE←<DYNAMIC VARIABLE RECORD
                              SPECIFIER>

<DYNAMIC VARIABLE
RECORD SPECIFIER> ::=         <EXPRESSION>

<DYNAMIC LOCK PART> ::=       LOCK←<DYNAMIC LOCK SPECIFIER>

<DYNAMIC LOCK
SPECIFIER> ::=                <EXPRESSION>

<DYNAMIC SAVE
FACTOR PART> ::=              SAVE←<DYNAMIC SAVE FACTOR>

<DYNAMIC SAVE FACTOR> ::=     <EXPRESSION>

<DYNAMIC RECORD
SIZE PART> ::=                RECORD.SIZE←<DYNAMIC RECORD SIZE>

<DYANMIC RECORD SIZE> ::=     <EXPRESSION>

<DYNAMIC REEL
NUMBER PART> ::=              REEL←<DYNAMIC REEL NUMBER>

<DYNAMIC REEL NUMBER> ::=     <EXPRESSION>

<DYNAMIC RECORDS-
PER-BLOCK PART> ::=           RECORDS.PER.BLOCK←
                              <DYNAMIC RECORDS-PER-BLOCK>
```

```
<DYNAMIC RECORDS-
PER-BLOCK> ::=                  <EXPRESSION>

<DYNAMIC NUMBER-OF-
AREAS PART> ::=                 NUMBER.OF.AREAS ←
                                <DYNAMIC NUMBER-OF-AREAS>

<DYNAMIC NUMBER-
OF-AREAS> ::=                   <EXPRESSION>

<DYNAMIC BLOCKS-PER-
AREA PART> ::=                  BLOCKS.PER.AREA ←
                                <DYNAMIC BLOCKS-PER-AREA>

<DYNAMIC BLOCKS-PER
AREA> ::=                       <EXPRESSION>

<DYNAMIC PACK.ID PART> ::=      PACK.ID ←
                                <DYNAMIC PACK IDENTIFICATION>

<DYNAMIC PACK
IDENTIFICATION> ::=             <EXPRESSION>

<DYNAMIC ALL-AREAS-
AT-OPEN PART> ::=               ALL.AREAS.AT.OPEN ←
                                <DYNAMIC ALL-AREAS-AT-OPEN SPECIFIER>

<DYNAMIC ALL-AREAS-
AT-OPEN SPECIFIER> ::=          <EXPRESSION

<DYNAMIC AREA-BY
CYLINDER PART> ::=              AREA.BY.CYLINDER ←
                                <DYNAMIC AREA-BY-CYLINDER SPECIFIER>

<DYNAMIC AREA-BY-
CYLINDER SPECIFIER> ::=         <EXPRESSION>

<DYNAMIC EU.SPECIAL
PART> ::=                       EU.SPECIAL ←
                                <DYNAMIC EU.SPECIAL SPECIFIER>
                              / EU.SPECIAL ←
                                <DYNAMIC EU.SPECIAL SPECIFIER>,
                                EU.DRIVE ←
                                <DYNAMIC EU.DRIVE PART>

<DYNAMIC EU.SPECIAL
SPECIFIER> ::=                  <EXPRESSION>

<DYNAMIC EU.DRIVE
PART> :·=                       EU.DRIVE←<DYNAMIC EU.DRIVE SPECIFIER>

<DYNAMIC EU.DRIVE
SPECIFIER> ::=                  <EXPRESSION>
```

```
<DYNAMIC EU.
INCREMENTED PART> ::=          EU.INCREMENTED ←
                               <DYNAMIC EU.INCREMENTED SPECIFIER>
                             / EU.INCREMENTED ←
                               <DYNAMIC EU.INCREMENTED SPECIFIER>,
                               EU.INCREMENT ←
                               <DYNAMIC EU.INCREMENT SPECIFIER>

<DYNAMIC EU.INCREMENTED
SPECIFIER> ::=                 <EXPRESSION>

<DYNAMIC EU.
INCREMENT SPECIFIER> ::=       <EXPRESSION>

<DYNAMIC USE.INPUT.
BLOCKING PART> ::=             USE.INPUT.BLOCKING ←
                               <DYNAMIC USE.INPUT.BLOCKING SPECIFIER>

<DYNAMIC USE.INPUT.
BLOCKING SPECIFIER> ::=        <EXPRESSION>

<DYNAMIC SORTER STATION
PART> ::=                      SR.STATION ←
                               <DYNAMIC SORTER STATION SPECIFIER>

<DYNAMIC SORTER
STATION SPECIFIER> ::=         <EXPRESSION>

<DYNAMIC MULTI-
PACK PART>::=                  MULTI-PACK ←
                               <DYNAMIC MULTI-PACK SPECIFIER>

<DYNAMIC MULTI-
PACK SPECIFIER> ::=            <EXPRESSION>

<DYNAMIC END-OF-
PAGE PART> ::=                 END.OF.PAGE.ACTION ←
                               <DYNAMIC END-OF-PAGE SPECIFIER>

<DYNAMIC END-OF-
PAGE SPECIFIER> ::=            <EXPRESSION>

<DYNAMIC OPEN-
OPTION PART>::=                OPEN.OPTION ←
                               <DYNAMIC OPEN.OPTION SPECIFIER>

<DYNAMIC OPEN-
OPTION SPECIFIER>::=           <EXPRESSION>

<DYNAMIC REMOTE-
KEY PART>::=                   REMOTE-KEY ←
                               <DYNAMIC REMOTE-KEY SPECIFIER>

<DYNAMIC REMOTE-
```

```
KEY SPECIFIER>::=                <EXPRESSION>

<DYNAMIC NUMBER-OF-
STATIONS PART>::=                NUMBER-OF-STATIONS ←
                                 <DYNAMIC NUMBER-OF-STATIONS SPECIFIER>

<DYNAMIC NUMBER-OF-
STATIONS SPECIFIER>::=           <EXPRESSION>

<DYNAMIC QUEUE-FAMILY-
SIZE PART>::=                    QUEUE-FAMILY-SIZE ←
                                 <DYNAMIC QUEUE-FAMILY-SIZE>

<DYNAMIC QUEUE-
FAMILY-SIZE>::=                  <EXPRESSION>

<DYNAMIC FILE
TYPE PART>::=                    FILE.TYPE ←
                                 <DYNAMIC FILE TYPE SPECIFIER>

<DYNAMIC FILE
TYPE SPECIFIER>::=               <EXPRESSION>

<DYNAMIC WORK
FILE PART>::=                    WORK.FILE ←
                                 <DYNAMIC WORK FILE SPECIFIER>

<DYNAMIC WORK
FILE SPECIFIER>::=               <EXPRESSION>

<DYNAMIC LABEL
TYPE PART>::=                    LABEL TYPE ←
                                 <DYNAMIC LABEL TYPE SPECIFIER>

<DYNAMIC LABEL
TYPE SPECIFIER>::=               <EXPRESSION>

<ZIP STATEMENT> ::=              ZIP <EXPRESSION>

<SEARCH STATEMENT> ::=             <SEARCH PART>
                                 / <SEARCH PART>; <FILE MISSING PART>
                                 / <SEARCH PART>; <FILE LOCKED PART>
                                 / <SEARCH PART>; <FILE MISSING PART>
                                   <FILE LOCKED PART>

<SEARCH PART> ::=                  SEARCH.DIRECTORY (<SEARCH OBJECT>,
                                   <SEARCH RESULT>,<SEARCH RESULT MODE>)

<SEARCH OBJECT> ::=                <ADDRESS GENERATOR>

<SEARCH RESULT> ::=                <ADDRESS GENERATOR>

<SEARCH RESULT MODE> ::=           BIT / CHARACTER
```

```
<FILE MISSING PART> ::=      ON FILE.MISSING <EXECUTABLE STATEMENT>

<FILE LOCKED PART> ::=       ON FILE.LOCKED <EXECUTABLE STATEMENT>

<ACCESS FILE HEADER
STATEMENT> ::=                 <ACCESS FILE HEADER PART>;
                             / <ACCESS FILE HEADER PART>;
                               <FILE MISSING PART>
                             / <ACCESS FILE HEADER PART>;
                               <FILE LOCKED PART>
                             / <ACCESS FILE HEADER PART>;
                               <FILE MISSING PART>
                               <FILE LOCKED PART>

<ACCESS FILE HEADER
PART> ::=                      READ.FILE.HEADER
                               (<FILE NAME>, <DESTINATION FIELD>)
                             / WRITE.FILE.HEADER
                               (<FILE NAME>, <SOURCE FIELD>)

<FILE NAME> ::=                <ADDRESS GENERATOR>

<DESTINATION FIELD> ::=      <ADDRESS GENERATOR>

<SOURCE FIELD> ::=           <ADDRESS GENERATOR>

<FILE MISSING PART> ::=      ON FILE.MISSING <EXECUTABLE STATEMENT>

<FILE LOCKED PART> ::=       ON FILE.LOCKED <EXECUTABLE STATEMENT>

<SEND STATEMENT> ::=           <SEND PART>;
                             / <SEND PART>; <QUEUE FULL PART>
                             / <SEND PART>; <INVALID REQUEST PART>
                             / <SEND PART>; <QUEUE FULL PART>
                               <INVALID REQUEST PART>

<SEND PART> ::=                SEND <MESSAGE SOURCE> TO <QUEUE>

<MESSAGE SOURCE> ::=         <ADDRESS GENERATOR>

<QUEUE> ::=                    <EXPRESSION>

<QUEUE FULL PART> ::=        ON Q.FULL <EXECUTABLE STATEMENT>

<INVALID REQUEST PART> ::=   ON INVALID.REQUEST <EXECUTABLE STATEMENT>

<RECEIVE STATEMENT> ::=        <RECEIVE PART>;
                             / <RECEIVE PART>; <QUEUE EMPTY PART>
                             / <RECEIVE PART>; <INVALID REQUEST PART>
                             / <RECEIVE PART>; <QUEUE EMPTY PART>
                               <INVALID REQUEST PART>

<RECEIVE PART> ::=             RECEIVE <MESSAGE DESTINATION>
                               FROM <QUEUE>
```

```
<MESSAGE DESTINATION> ::=     <ADDRESS GENERATOR>

<QUEUE> ::=                   <EXPRESSION>

<QUEUE EMPTY PART> ::=        ON Q.EMPTY <EXECUTABLE STATEMENT>

<INVALID REQUEST PART> ::=    ON INVALID.REQUEST <EXECUTABLE STATEMENT>

<ARRAY PAGE TYPE
STATEMENT> ::=                <ARRAY PAGE TYPE DESIGNATOR>
                              (<PAGED ARRAY NAME>,<PAGE NUMBER>)

<ARRAY PAGE TYPE
DESIGNATOR> ::=               MAKE.READ.ONLY
                            / MAKE.READ.WRITE

<PAGED ARRAY NAME> ::=        <IDENTIFIER>

<PAGE NUMBER> ::=             <EXPRESSION>

<COROUTINE STATEMENT>::=      <COROUTINE ENTRY STATEMENT>
                            / <COROUTINE EXIT STATEMENT>

<COROUTINE
ENTRY STATEMENT>::=           ENTER.COROUTINE
                              (<COROUTINE TABLE SPECIFIER>)

<COROUTINE
TABLE SPECIFIER>::=           <ADDRESS GENERATOR>

<COROUTINE
EXIT STATEMENT>::=            EXIT.COROUTINE
                              (<COROUTINE TABLE SPECIFIER>)

<WAIT STATEMENT>::=           WAIT (<NUMBER OF TENTHS OF SECONDS>)
                            / WAIT (<NUMBER OF TENTHS OF SECONDS>,
                              DC.IO.COMPLETE)
                            / WAIT (,DC.IO.COMPLETE)
```

APPENDIX II: RESERVED AND SPECIAL WORDS

THE FOLLOWING IS A LIST OF RESERVED WORDS IN SDL, COMPLETE AS OF NOV. 1973. THESE WORDS MAY ONLY BE USED AS RESERVED WORDS.

ACCEPT	AND	AS
BASE	BIT	BUMP
BY		
CASE	CAT	CHANGE
CHANGE.STACK.SIZES	CHARACTER	CLEAR
CLOSE		
DECLARE	DECREMENT	DEFINE
DISPLAY	DO	DUMMY
DYNAMIC		
ELSE	END	ENTER.COROUTINE
EQL	EXIT.COROUTINE	EXOR
FILE	FILLER	FINI
FIXED	FORMAL	FORMAL.VALUE
FORWARD	FROM	
GEQ	GTR	
IF	INTRINSIC	
LEQ	LOCK	LSS
MESSAGE ***	MOD	
NEG	NOT	
OF	ON	OPEN
OR		
PAGED	PROCEDURE	
QUEUE ***		
READ	READ.FILE.HEADER	RECEIVED
REMAPS	RETURN	RETURN.AND.ENABLE.INTERRUPTS
SEARCH.DIRECTORY	SEEK	SEGMENT
SEGMENT.PAGE	SEND	SKIP
SLEEP	SPACE	STOP
SUBBIT	SUBSTR	
THEN	TO	TRANSFER.MESSAGE ***
UNDO	USE	
VARYING		
WAIT	WRITE	WRITE.FILE.HEADER
ZIP		

*** DESCRIBED IN APPENDIX X: DC SDL

THE FOLLOWING IS A LIST OF SPECIAL WORDS IN SDL, COMPLETE AS OF NOV. 1973. EACH SPECIAL WORD HAS A PARTICULAR MEANING, HOWEVER IT MAY BE USED AS AN IDENTIFIER. IN THAT CASE, IT LOSES ITS SPECIAL SIGNIFICANCE IN SDL.

ACCESS.FILE.INFORMATION	ALLOCATE ***
BASE.REGISTER	BINARY
COMMUNICATE	COMPILE.CARD.INFO
CONTROL.STACK.BITS	CONTROL.STACK.TOP
CONSOLE.SWITCHES	CONV
CONVERT	
DATA.ADDRESS	DATE
DC.INITIATE.10 *, ***	DC.WAIT *, ***
DC.WRITE ***	DEBLANK
DECIMAL	DELIMITED.TOKEN
DESCRIPTOR	DE.ALLOCATE ***
DISABLE.INTERRUPTS	DISABLE.QUEUE ***
DISPATCH	DISPLAY.BASE
DUMP	DUMP.FOR.ANALYSIS
DYNAMIC.MEMORY.BASE	
ENABLE.INTERRUPTS	ENABLE.QUEUE ***
ERROR.COMMUNICATE	EVALUATION.STACK.TOP
EXECUTE	
FETCH	FETCH.COMMUNICATE.MSG.PTR
FLUSH ***	FREEZE.PROGRAM
HALT	HARDWARE.MONITOR ***
HASH.CODE	
INITIALIZE.VECTOR	INSERT ***
INTERROGATE.INTERRUPT.STATUS	
LENGTH	LIMIT.REGISTER
LOCATION	
MAKE.DESCRIPTOR	MAKE.READ.ONLY
MAKE.READ.WRITE	MCS.COMMUNICATE ***
MESSAGE.INFO ***	M.MEM.SIZE
NAME.OF.DAY	NAME.STACK.TOP
NEXT.ITEM	NEXT.TOKEN
NOTRACE	
OVERLAY	
PARITY.ADDRESS	PREVIOUS.ITEM
QUEUE.INFO ***	
READ.CASSETTE	READ.FPB
READ.OVERLAY	REINSTATE
REMOVE ***	RESTORE
REVERSE.STORE	
SAVE.STATE	SEARCH.LINKED.LIST
SEARCH.SERIAL.LIST	S.MEM.SIZE
SEARCH.SDL.STACKS	
SORT	SORT.FILE.FIXUP
SORT.RETURN	SORT.SEARCH
SORT.STEP.DOWN	SORT.SWAP
SORT.UNBLOCK	SWAP

SPO.INPUT.PRESENT
THAW.PROGRAM THREAD.VECTOR
TIME TRACE
VALUE.DESCRIPTOR
WRITE.FPB WRITE.OVERLAY

* TEMPORARY - TO BE REMOVED IN THE NEAR FUTURE

** SEE APPENDIX IV

*** DESCRIBED IN APPENDIX X: DC SDL

APPENDIX III: SDL CONTROL CARD OPTIONS

EVERY SDL CONTROL CARD MUST HAVE A "$" IN COLUMN ONE. COLUMNS 73-80 MAY BE USED AS A SEQUENCE FIELD. NOTE THAT ONCE AN OPTION HAS BEEN TURNED ON (OFF), IT WILL REMAIN ON (OFF) UNTIL EXPLICITLY TURNED OFF (ON).

CONTROL CARD OPTIONS FOR B5500

```
<CONTROL CARD> ::=              $ <CONTROL STATEMENT>

<CONTROL STATEMENT> ::=         <CONTROL OPTION LIST>
                              / <VOID OPTION>

<CONTROL OPTION LIST> ::=       <CONTROL OPTION>
                              / <CONTROL OPTION>
                                <CONTROL OPTION LIST>

<CONTROL OPTION> ::=            <CONTROL OPTION WORD>
                              / NO <CONTROL OPTION WORD>
                              / <DEBUG OPTION>
                              / <SEQUENCE OPTION>
                              / <PAGE OPTION>
                              / <UPDATE OPTION>
                              / <STACK SIZE LIST>
                              / <INTERPRETER OPTION>

<CONTROL OPTION WORD> ::=       LIST / LISTALL / SINGLE / SGL
                              / DOUBLE / CODE / CONTROL / NEW
                              / CHECK / MAP / XMAP / DETAIL
                              / AMPERSAND / SIZE
                              / HEX / PROFILE / PPROFILE
                              / NODUPLICATES

<DEBUG OPTION> ::=              DEBUG <DEBUG PARAMETER>

<DEBUG PARAMETER> ::=           <EMPTY>
                              / <NUMBER>

<NUMBER> ::=                    <UNSIGNED INTEGER, 8 OR LESS DIGITS>

<SEQUENCE OPTION> ::=           NO SEQ
                              / SEQ <SEQUENCE PARAMETERS>

<SEQUENCE PARAMETERS> ::=       <BASE>
                              / <BASE> <INCREMENT>
```

```
<BASE> ::=                          <NUMBER>

<INCREMENT> ::=                     <NUMBER>

<PAGE OPTION> ::=                   PAGE

<UPDATE OPTION> ::=                 UPDATE

<STACK SIZE LIST> ::=                 <STACK SIZE DESIGNATOR>
                                    / <STACK SIZE DESIGNATOR>
                                      <STACK SIZE LIST>

<STACK SIZE DESIGNATOR> ::=         <STACK DESIGNATOR> <STACK SIZE>

<STACK DESIGNATOR> ::=                VSSIZE / NSSIZE / ESSIZE
                                    / CSSIZE / PPSSIZE / DYNAMICSIZE

<STACK SIZE> ::=                    <NUMBER>

<VOID OPTION> ::=                   VOID <TERMINATING SEQUENCE FIELD>

<TERMINATING SEQUENCE
FIELD> ::=                            <EMPTY>
                                    / <EXACTLY 8 CHARACTERS>

<INTERPRETER OPTION> ::=              INTERPRETER <INTERPRETER NAME>

<INTERPRETER NAME> ::=                <INTERPRETER MFID>
                                    / <INTERPRETER MFID> <SLASH>
                                      <INTERPRETER IDENTIFIER>
                                    / <SLASH> <INTERPRETER IDENTIFIER>

<INTERPRETER MFID> ::=              <IDENTIFIER>

<INTERPRETER
IDENTIFIER> ::=                     <IDENTIFIER>
```

SEMANTICS:

LIST	LISTS SDL SOURCE INPUT WHICH HAS BEEN COMPILED. "NO LIST" WILL TURN OFF "LISTALL" (IF ALSO SPECIFIED) AS WELL AS "LIST".
LISTALL	LISTS ALL SDL SOURCE INPUT (WHETHER OR NOT CONDITIONALLY EXCLUDED). "LISTALL" WILL TURN ON "LIST". BUT, "NO LISTALL" WILL NOT TURN OFF "LIST".
SINGLE (SGL)	SINGLE SPACE LISTING WHEN PRINTING.

DOUBLE	DOUBLE SPACE LISTING WHEN PRINTING.
CODE	PRINT GENERATED CODE.
CONTROL	PRINT CONTROL CARDS.
NEW	CREATE NEW SOURCE FILE.
MAP	PRINT CODE MAPPING INFORMATION.
XMAP	CREATE EXTENDED CODE MAP FILE FOR POST COMPILATION ANALYSIS AND PRINTING.
DETAIL	PRINT EXPANSION OF DEFINE INVOCATIONS.
AMPERSAND	PRINT THOSE AMPERSAND CARDS WHICH ARE EXAMINED.
SIZE	PRINT SEGMENT SIZES BY NAME AT END OF COMPILE.
HEX	ADDRESSES PRINTED IN HEXADECIMAL WHEN "CODE" OR "MAP" OPTIONS ARE USED.
PROFILE	SEE APPENDIX IV
PPROFILE	SEE APPENDIX IV
DEBUG	COMPILER DEBUG USE ONLY.
SEQ	RESEQUENCE OUTPUT FILE.
CHECK	THE SOURCE INPUT WILL BE CHECKED FOR SEQUENCE ERRORS.
NODUPLICATES	NEWLY DECLARED IDENTIFIERS WILL NOT BE CHECKED FOR UNIQUENESS. THE USER MUST GUARANTEE THAT THERE ARE NO DUPLICATES BEFORE USING THIS OPTION. THIS SHOULD BE USED FOR LONG PROGRAMS ONLY.
NO SEQ	TURNS OFF THE "SEQ" OPTION.
PAGE	PAGE EJECT, IF LISTING.
UPDATE	THE PRIMARY SOURCE IS A TAPE OR DISK FILE WHICH WILL HAVE THE CARDS FROM THE CARD READER MERGED INTO IT.
VSSIZE	VALUE STACK SIZE
NSSIZE	NAME STACK SIZE
ESSIZE	EVALUATION STACK SIZE.
CSSIZE	CONTROL STACK SIZE

PPSSIZE	PROGRAM POINTER STACK SIZE
DYNAMICSIZE	AMOUNT OF CORE USED FOR PAGED ARRAY PAGES
NO	"NO" PRECEDING AN OPTION (WHICH ALLOWS IT) WILL TURN THAT OPTION OFF.
VOID	THE "VOID" CARD WILL VOID IMAGES ON THE PRIMARY INPUT FILE WHICH HAVE SEQUENCE FIELDS WHICH ARE LESS THAN OR EQUAL TO THE <TERMINATING SEQUENCE FIELD> APPEARING ON THE "VOID" CARD. IF THE <TERMINATING SEQUENCE FIELD> IS MISSING, THEN THE ONLY SOURCE IMAGE VOIDED IS THAT WITH THE SAME SEQUENCE FIELD AS THE "VOID" CARD. NOTE THAT THE "VOID" CARD MAY DELETE IMAGES IN THE SECONDARY (CARD) FILE.
INTERPRETER	CHANGE INTERPRETER ID AND/OR INTERPRETER MFID.

NOTES AND RESTRICTIONS:
---

1. UNLESS OTHERWISE SPECIFIED (THROUGH THE "UPDATE" OPTION), THE PRIMARY SOURCE OF INPUT IS THE CARD READER. ONCE "UPDATE" HAS BEEN SPECIFIED, IT IS NOT POSSIBLE TO AGAIN INDICATE "CARDS ONLY".

2. IF NO CONTROL CARDS ARE USED, THE DEFAULT OPTIONS ARE LIST, DOUBLE, AND AMPERSAND; ALL INPUT WILL BE FROM CARDS.

3. OPTIONS ARE TURNED OFF ONLY THROUGH THE APPEARANCE OF "NO" FOLLOWED BY THE OPTION WORD. NOTE THAT "NO" AND THE OPTION WORD ARE SEPARATED BY AT LEAST ONE BLANK.

CONTROL CARD OPTIONS FOR B1700

```
<CONTROL CARD> ::=              $ <CONTROL STATEMENT>

<CONTROL STATEMENT> ::=           <CONTROL OPTION LIST>
                                / <VOID OPTION>

<CONTROL OPTION LIST> ::=         <CONTROL OPTION>
                                / <CONTROL OPTION>
                                  <CONTROL OPTION LIST>

<CONTROL OPTION> ::=              <CONTROL OPTION WORD>
```

```
                                / NO <CONTROL OPTION WORD>
                                / <DEBUG OPTION>
                                / <SEQUENCE OPTION>
                                / <PAGE OPTION>
                                / <MERGE OPTION>
                                / <STACK SIZE LIST>
                                / <INTERPRETER OPTION>
                                / <INTRINSIC OPTION>
                                / <RECOMPILE OPTION>
                                / SIZE / FORMAL.CHECK

<CONTROL OPTION WORD> ::=       LIST / LISTALL / SINGLE
                                / SGL / DOUBLE / CODE
                                / CONTROL / NEW / SUPPRESS
                                / XMAP / CHECK / PROFILE / PPROFILE
                                / DETAIL / AMPERSAND / NO.DUPLICATES
                                / NO.SOURCE / MONITOR
                                / XREF / XREF.ONLY / EXPAND.DEFINES

<DEBUG OPTION> ::=              DEBUG <NUMBER>

<NUMBER> ::=                    <UNSIGNED INTEGER, 8 OR LESS DIGITS>

<SEQUENCE OPTION> ::=           NO SEQ
                                / SEQ <SEQUENCE PARAMETERS>

<SEQUENCE PARAMETERS> ::=       <BASE>
                                / <INCREMENT>
                                / <BASE> <INCREMENT>

<BASE> ::=                      <NUMBER>

<INCREMENT>::=                  + <NUMBER>

<PAGE OPTION> ::=               PAGE

<MERGE OPTION> ::=              MERGE

<STACK SIZE LIST> ::=           <STACK SIZE DESIGNATOR>
                                / <STACK SIZE DESIGNATOR>
                                  <STACK SIZE LIST>

<STACK SIZE
DESIGNATOR> ::=                 <STACK DESIGNATOR> <STACK SIZE>

<STACK DESIGNATOR> ::=          VSSIZE / NSSIZE / ESSIZE
                                / CSSIZE / PPSSIZE / DYNAMICSIZE

<STACK SIZE> ::=                <NUMBER>

<VOID OPTION> ::=               VOID <TERMINATING SEQUENCE FIELD>

<TERMINATING SEQUENCE
FIELD> ::=                      <EMPTY>
```

```
                                / <EXACTLY 8 CHARACTERS>

<INTERPRETER OPTION> ::=          INTERPRETER <INTERPRETER NAME>

<INTERPRETER NAME> ::=            <INTERPRETER MULTI-FILE IDENTIFICATION>
                                / <INTERPRETER MULTI-FILE IDENTIFICATION>
                                  <SLASH>
                                  <INTERPRETER IDENTIFICATION>

<INTERPRETER MULTI-FILE
IDENTIFICATION> ::=               <IDENTIFIER>

<INTERPRETER
IDENTIFICATION> ::=               <IDENTIFIER>

<INTRINSIC OPTION> ::=            INTRINSIC
                                  <INTRINSIC FAMILY NAME>

<INTRINSIC FAMILY
NAME> ::=                         <IDENTIFIER>

<RECOMPILE OPTION>::=             CREATE.MASTER
                                / RECOMPILE
```

SEMANTICS IN ALPHABETICAL ORDER:

AMPERSAND	PRINTS THOSE AMPERSAND CARDS WHICH ARE EXAMINED.
CHECK	THE SOURCE INPUT WILL BE CHECKED FOR SEQUENCE ERRORS.
CODE	PRINTS GENERATED CODE.
CONTROL	PRINTS CONTROL CARDS.
CREATE.MASTER	SEE APPENDIX VIII.
CSSIZE	CONTROL STACK SIZE.
DEBUG	COMPILER DEBUG USE ONLY.
DETAIL	PRINTS EXPANSION OF DEFINE INVOCATIONS.
DOUBLE	DOUBLE SPACE LISTING WHEN PRINTING.
DYNAMICSIZE	AMOUNT OF MEMORY USED FOR PAGED ARRAY PAGES.
ESSIZE	EVALUATION STACK SIZE.
FORMAL.CHECK	PROCEDURE ACTUAL PARAMETERS AND VALUES RETURNED

	FROM TYPED PROCEDURES WILL BE CHECKED RESPECTIVELY AGAINST THEIR CORRESPONDING FORMAL PARAMETERS AND PROCEDURE FORMAL TYPES.
EXPAND.DEFINES	CAUSES DEFINE EXPANSIONS TO BE CROSS-REFERENCED (USED IN CONJUNCTION WITH XREF OR XREF.ONLY).
INTERPRETER	CHANGES THE INTERPRETER ID OR MFID, OR BOTH.
INTRINSIC	CHANGES THE FAMILY NAMES OF INTRINSICS TO BE USED.
LIST	LISTS THE SOURCE INPUT WHICH WAS COMPILED. "NO LIST" WILL ALSO TURN OFF "LISTALL".
LISTALL	LISTS ALL SDL SOURCE INPUT (WHETHER OR NOT CONDITIONALLY EXCLUDED). "LISTALL" TURNS ON "LIST", BUT "NO LISTALL" WILL NOT TURN OFF "LIST".
MERGE	THE PRIMARY SOURCE FILE IS ON TAPE OR DISK WHICH WILL HAVE THE CARDS, FROM THE CARD READER, MERGED WITH IT.
MONITOR	SEE APPENDIX IX: THE SDL MONITOR FACILITY
NEW	CREATES A NEW SOURCE FILE.
NO	"NO" PRECEDING AN OPTION (WHICH ALLOWS IT) WILL TURN THAT OPTION OFF.
NO.DUPLICATES	NEWLY DECLARED IDENTIFIER WILL NOT BE CHECKED FOR UNIQUENESS. THE PROGRAMMER MUST GUARANTEE THAT THERE ARE NO DUPLICATES BEFORE USING THIS OPTION. IT WILL REDUCE COMPILE TIME FOR LARGE PROGRAMS ONLY.
NO SEQ	TURNS OFF THE "SEQ" OPTION.
NO.SOURCE	PROGRAM SOURCE IMAGES WILL NOT BE SAVED, THEREBY SHORTENING THE COMPILER WORK FILE. NO SOURCE LISTING WILL BE POSSIBLE WHEN THIS OPTION IS SPECIFIED. THIS SHOULD BE USED WITH LONG PROGRAMS ONLY.
NSSIZE	NAME STACK SIZE.
PAGE	PAGE EJECT IF LISTING.
PPSSIZE	PROGRAM POINTER STACK SIZE.
RECOMPILE	SEE APPENDIX VIII
SEG	RESEQUENCES OUTPUT FILE.

SINGLE (SGL)	SINGLE SPACES LISTING WHEN PRINTING.
SIZE	PRINTS SEGMENT SIZES BY NAME AT END OF COMPILE.
SUPPRESS	SUPPRESSES WARNING MESSAGES. TO SUPPRESS SEQUENCE ERROR MESSAGES, TURN OFF "CHECK".
VOID	THE VOID OPTION WILL VOID RECORDS IN THE PRIMARY FILE WHICH HAVE SEQUENCE FIELDS LESS THAN OR EQUAL TO THE <TERMINATING SEQUENCE FIELD>. IF THE FIELD IS OMITTED, ONLY THE RECORD WITH THE SEQUENCE NUMBER CORRESPONDING TO THE "VOID CARD" SEQUENCE NUMBER WILL BE DELETED. THE VOID OPTION WILL NOT DELETE IMAGES IN A SECONDARY (CARD) SOURCE FILE.
VSSIZE	VALUE STACK SIZE.
XMAP	CREATES AN EXTENDED CODE MAP FILE FOR POST COMPILATION ANALYSIS.
XREF	PRODUCES A CROSS-REFERENCE LISTING OF THE PROGRAM.
XREF.ONLY	PRODUCES A CROSS-REFERENCE LISTING AND THEN TERMINATES THE COMPILATION.

NOTE: ALL CONTROL CARDS MAY USE "&" IN COLUMN 1 IN PLACE OF "$". THOSE CONTROL CARDS WITH "&" IN COLUMN 1 WILL BE PERMANANTLY PLACED IN A NEW SOURCE FILE WHENEVER ONE IS MADE. THEY MAY ALSO BE CONDITIONALLY INCLUDED OR EXCLUDED DURING COMPILATION.

APPENDIX IV: PROGRAMMING OPTIMIZATION

THE FOLLOWING CONTROL CARD OPTIONS (SEE APPENDIX III) CAN BE USEFUL TO THE PROGRAMMER WHO WISHES TO DETERMINE THE MOST TIME CONSUMING PART(S) OF HIS PROGRAM. THE PURPOSE OF THESE CONTROL OPTIONS IS TO POINT OUT THE PARTS OF THE PROGRAM WHICH ARE THE MOST TIME CONSUMING AND/OR HEAVILY USED.

WHEN COMPILING ON THE B5500, IF ANY OF THESE CONTROLS ARE USED, THE FOLLOWING CARD MUST APPEAR IN THE PROCEDURE SECTION OF THE PROGRAM:

```
& LIBRARY SDLPROFILE / ARRAYPRINTER
```

IN ADDITION, THERE MUST BE A CLOSE ON THE PRINTER BEFORE "FINI" OR ANY <STOP STATEMENT>.

PPROFILE — ESTABLISHES A DYNAMIC ARRAY, EACH ELEMENT OF WHICH IS A COUNTER FOR ONE PROCEDURE. THE INDEX NUMBER FOR EACH PROCEDURE APPEARS IN THE LISTING FOLLOWING THE <PROCEDURE IDENTIFIER>. THE VALUE OF THE COUNTER WILL REFLECT THE NUMBER OF ENTRANCES TO THE PROCEDURE IN QUESTION. THOSE WITH THE HIGHEST COUNTERS SHOULD BE INVESTIGATED WITH THE "PROFILE" OPTION.

PROFILE — ESTABLISHES A DYNAMIC ARRAY, EACH ELEMENT OF WHICH IS A COUNTER FOR ONE BRANCHING OPERATION (<DO GROUP>, <IF STATEMENT>, OR <CASE STATEMENT>). THE INDEX INTO THE ARRAY WILL APPEAR IN THE LISTING FOLLOWING THE STATEMENT IN QUESTION. THOSE BRANCHES WITH THE HIGHEST COUNTER VALUES ARE THE BRANCHES MOST HEAVILY USED.

HARDWARE MONITOR

<HARDWARE MONITOR DESIGNATOR> ::= HARDWARE.MONITOR (<EXPRESSION>)

THE B1700 IS EQUIPPED WITH A HARDWARE MONITOR WHICH MAY BE MANUALLY WIRED TO SUIT THE NEEDS OF THE PROGRAMMER. THE DEVICE CAN BE USEFUL AS A TIMER OR A COUNTER TO MONITOR PROGRAM EFFICIENCY.

THE LOW-ORDER 8 BITS OF THE <EXPRESSION> IS USED AS THE LOW-ORDER 8 BITS OF THE M-INSTRUCTION "MONITOR". FOR WIRING INSTRUCTIONS OF THE HARDWARE DEVICE SEE "COMPUTER PERFORMANCE MONITOR II: SYSTEM SUMMARY MANUAL".

APPENDIX V: SYSTEM CONTROL CARDS

SYSTEM CONTROL CARDS FOR B5500

THERE ARE TWO BASIC FORMATS FOR THE SYSTEM CONTROL CARDS. THEY ARE AS FOLLOWS:

A. THE PRIMARY SOURCE FILE IS ON CARDS.

```
   ? COMPILE <PROG ID>/<USER ID> WITH [SDL|MSDL] [LIBRARY|SYNTAX] <ACCT INFO>
*  ? FILE NEWTAPE = <FILE MFID>/<FILE ID> [SERIAL|TAPE]
   ? DATA <USER ID>
*  $ NEW
   <SDL PROGRAM>
   FINI
   ? END
```

* THESE CARDS MUST BE INCLUDED IF THE FILE IS TO BE SAVED ON TAPE OR DISK.

B. THE PRIMARY SOURCE FILE IS ON TAPE OR DISK, AND THE SECONDARY SOURCE FILE IS ON CARDS.

```
   ? COMPILE <PROG ID>/<USER ID> WITH [SDL|MSDL] [LIBRARY|SYNTAX] <ACCT INFO>
   ? FILE TAPE = <PROG ID>/<USER ID> [SERIAL|TAPE]
*  ? FILE NEWTAPE = <FILE MFID>/<FILE ID> [SERIAL|TAPE]
   ? DATA <USER ID>
   $ UPDATE
*  $ NEW
   <PATCHES TO SDL PROGRAM>
   <9-S CARD> (SEQUENCE FIELD IS 99999999)
   ? END
```

* THESE CARDS MUST BE INCLUDED IF THE UPDATED FILE IS TO BE SAVED ON TAPE OR DISK.

THE " " SPECIFIES ANY INVALID CHARACTER. THE "COMPILE" CARD SHOULD SPECIFY "SDL" IF THE PROGRAM IS TO BE RUN ON ENGINEERING SYSTEM #2, OR "MSDL" IF MARKETING SYSTEM #2. NOTE THAT THIS IS A TEMPORARY DISTINCTION AND IS SUBJECT TO CHANGE.

IF OBJECT CODE IS TO BE GENERATED, THE "LIBRARY" OPTION SHOULD BE USED. THE "SYNTAX" OPTION WILL CHECK ONLY FOR SYNTAX ERRORS AND GENERATE A SOURCE LISTING.

THE FORMAT FOR THE <ACCT INFO> IS AS FOLLOWS:

#<4-DIGIT COST CENTER> <1 SPACE> <5-DIGIT PROJECT NUMBER>

AN EXAMPLE "COMPILE CARD" IS:

```
? COMPILE MYPROGRAM/SMITH WITH SDL LIBRARY #1234 56789
```

TO DUMP THE OBJECT CODE TO CARDS, THE FOLLOWING CONTROL CARDS SHOULD BE PLACED JUST BEFORE THE END-CARD:

```
? EX SDL/DUMPER <ACCT INFO>
? FILE CODEFILE = <PROG ID>/<USER ID> [SERIAL]
                                      [TAPE  ]
? COMMON = 9
```

SDL FILE NAMES

CARD	CARD INPUT FILE
TAPE	PRIMARY SOURCE FILE IF $ UPDATE IS USED
NEWTAPE	NEW SOURCE FILE IF $ NEW IS USED
LINE	PRINTER FILE

SYSTEM CONTROL CARDS FOR B1700

THERE ARE TWO BASIC DECK SETUP FORMATS. THEY ARE:

A. THE PRIMARY SOURCE FILE IS ON CARDS.

```
   ? <SYSTEM COMPILE CARD>
*  ? <FILE EQUATE CARD FOR FILE "NEWSOURCE">
   ? DATA CARDS
*  $ NEW
   <SDL PROGRAM>
   FINI
   ? END
```

* IF THE PRIMARY SOURCE FILE IS TO BE SAVED ON TAPE OR DISK, THESE CARDS MUST BE INCLUDED.

B. THE PRIMARY SOURCE FILE IS ON TAPE OR DISK.

```
   ? <SYSTEM COMPILE CARD>
   ? <FILE EQUATE CARD FOR FILE "SOURCE">
*  ? <FILE EQUATE CARD FOR FILE "NEWSOURCE">
   ? DATA CARDS
   $ MERGE
*  $ NEW
   <PATCHES TO SDL PROGRAM>
   <9-S CARD> (SEQUENCE FIELD CONTAINS 99999999)
   ? END
```

* IF THE MERGED FILE IS TO BE SAVED, THESE CARDS MUST BE INCLUDED.

NOTE REFER TO THE B1700 MCP DOCUMENTATION FOR THE EXACT FORMAT OF THE "COMPILE" AND "FILE EQUATE" CARDS.

SDL FILE NAMES

CARDS	CARD INPUT FILE
SOURCE	PRIMARY SOURCE FILE IF "$ MERGE" IS USED
NEWSOURCE	UPDATED SOURCE FILE IF "$ NEW" IS USED
LINE	LINE PRINTER FILE

APPENDIX VI: CONDITIONAL COMPILATION

THE CONDITIONAL COMPILATION FACILITY ALLOWS THE USER TO SELECTIVELY COMPILE BLOCKS OF CODE WITHOUT THE NECESSITY OF PHYSICALLY ADDING OR REMOVING RECORDS.

<CONDITIONAL INCLUSION> RECORDS ARE ALWAYS WRITTEN TO A NEW FILE (IF ONE IS CREATED), WHETHER OR NOT THEY ARE COMPILED. IF CONDITIONAL COMPILATION RECORDS ARE TO BE PRINTED WITH THE SOURCE LISTING, THEN "LISTALL" MUST APPEAR ON THE $-CARD. IF NOT SPECIFIED, ONLY THOSE CONDITIONAL COMPILATION RECORDS WHICH WERE COMPILED ARE PRINTED.

THE BNF FOR THE CONDITIONAL COMPILATION IS AS FOLLOWS:

```
<CONDITIONAL INCLUSION> ::=  <SET STATEMENT>
                           / <RESET STATEMENT>
                           / <PAGE STATEMENT>
                           / <LIBRARY STATEMENT>
                           / <IF BLOCK>

<SET STATEMENT> ::=            SET <SET SYMBOL LIST>

<SET SYMBOL LIST> ::=          <SET SYMBOL>
                           / <SET SYMBOL LIST>
                             <SET SYMBOL>

<SET SYMBOL> ::=               <BOOLEAN SYMBOL>

<BOOLEAN SYMBOL> ::=           <LETTER>
                           / <BOOLEAN SYMBOL> <LETTER>
                           / <BOOLEAN SYMBOL> <DIGIT>

<RESET STATEMENT> ::=          RESET <RESET SYMBOL LIST>

<RESET SYMBOL LIST> ::=        <RESET SYMBOL>
                           / <RESET SYMBOL LIST>
                             <RESET SYMBOL>

<RESET SYMBOL> ::=             <BOOLEAN SYMBOL>

<PAGE STATEMENT> ::=           PAGE

<LIBRARY STATEMENT> ::=        LIBRARY <FILE NAME>


<FILE NAME>::=                 <MULTI-FILE IDENTIFIER>
```

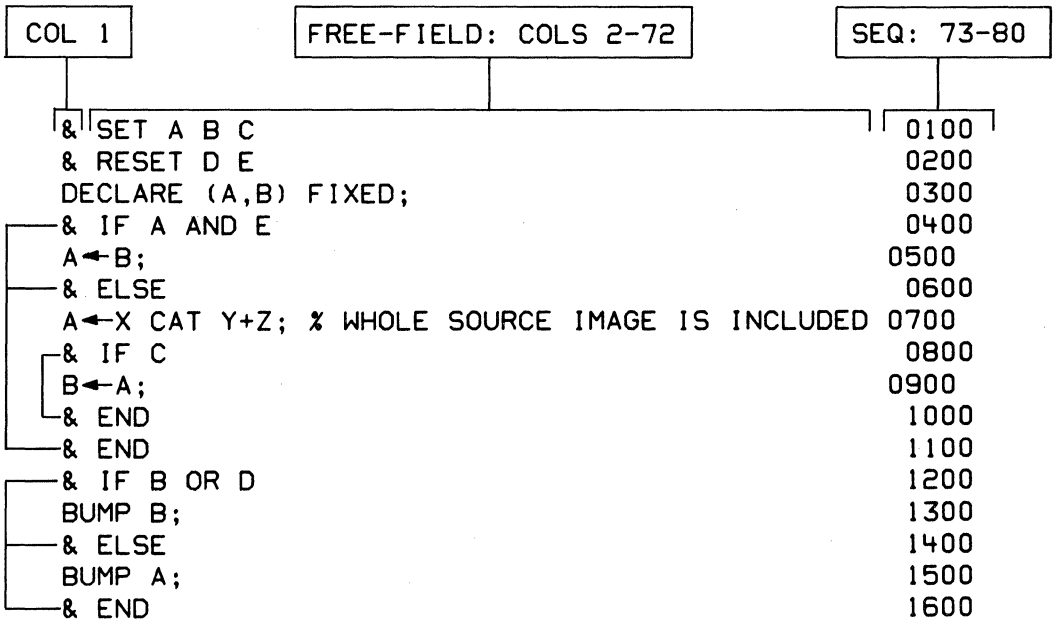
```
                                / <MULTI-FILE IDENTIFIER> <SLASH>
                                  <FILE IDENTIFIER>
                                / <PACK IDENTIFIER> <SLASH>
                                  <MULTI-FILE IDENTIFIER> <SLASH>
                                / <PACK IDENTIFIER> <SLASH>
                                  <MULTI-FILE IDENTIFIER> <SLASH>
                                  <FILE IDENTIFIER>

<PACK IDENTIFIER>::=              <IDENTIFIER>

<MULTI-FILE IDENTIFIER>::=        <IDENTIFIER>

<FILE IDENTIFIER>::=              <IDENTIFIER>

<SLASH>::=                        /

<IF BLOCK> ::=                    <IF STATEMENT>
                                  <INCLUSION BLOCK>
                                  <END STATEMENT>
                                / <IF STATEMENT>
                                  <TRUE PART>
                                  <INCLUSION BLOCK>
                                  <END STATEMENT>

<IF STATEMENT> ::=                IF <BOOLEAN EXPRESSION>

<BOOLEAN EXPRESSION> ::=          <BOOLEAN FACTOR>
                                / <BOOLEAN EXPRESSION> OR
                                  <BOOLEAN FACTOR>

<BOOLEAN FACTOR> ::=              <BOOLEAN SECONDARY>
                                / <BOOLEAN FACTOR> AND
                                  <BOOLEAN SECONDARY>

<BOOLEAN SECONDARY> ::=           <BOOLEAN PRIMARY>
                                / NOT <BOOLEAN PRIMARY>

<BOOLEAN PRIMARY> ::=             <SET SYMBOL>
                                / <RESET SYMBOL>

<INCLUSION BLOCK> ::=             <SDL SOURCE IMAGE BLOCK>
                                / <IF BLOCK>

<SDL SOURCE
IMAGE BLOCK> ::=                  <EMPTY>
                                / <1 OR MORE SDL SOURCE IMAGES>

<END STATEMENT> ::=               END

<TRUE PART> ::=                   <INCLUSION BLOCK> <ELSE STATEMENT>

<ELSE STATEMENT> ::=              ELSE
```

ALL RECORDS CONTAINING CONDITIONAL COMPILATION STATEMENTS MUST HAVE AN AMPERSAND (&) IN COLUMN 1 (EXCEPT THE <SDL SOURCE IMAGE BLOCK>). IN ADDITION, A COMPLETE CONDITIONAL INCLUSION STATEMENT MUST BE CONTAINED ON ONE AMPERSAND CARD. COLUMNS 2-72 ARE FREE-FIELD, AND COLUMNS 73-80 MAY CONTAIN SEQUENCE NUMBERS.

NOTE THAT <BOOLEAN EXPRESSION>S MAY CONTAIN THE LOGICAL OPERATORS (FROM LOWEST PRECEDENCE TO HIGHEST) "OR", "AND", AND "NOT".

THE <PAGE STATEMENT> WILL CAUSE A PAGE EJECT IF THE SOURCE FILE IS BEING LISTED. THE <LIBRARY STATEMENT> WILL CAUSE THE IMAGES FROM THE FILE SPECIFIED BY <FILE NAME> TO BE INCLUDED IN THE SOURCE PROGRAM.

AS AN EXAMPLE, CONSIDER THE FOLLOWING SDL SOURCE STATEMENTS ILLUSTRATING NESTED CONDITIONAL COMPILATION STATEMENTS AND <SDL SOURCE IMAGE BLOCK>S.




```
&SET A B C                                                  0100
& RESET D E                                                 0200
DECLARE (A,B) FIXED;                                        0300
& IF A AND E                                                0400
A←B;                                                       0500
& ELSE                                                      0600
A←X CAT Y+Z; % WHOLE SOURCE IMAGE IS INCLUDED              0700
& IF C                                                      0800
B←A;                                                       0900
& END                                                       1000
& END                                                       1100
& IF B OR D                                                 1200
BUMP B;                                                     1300
& ELSE                                                      1400
BUMP A;                                                     1500
& END                                                       1600
```

THE COMPILATION OF THE FOLLOWING STATEMENTS WOULD RESULT.

```
DECLARE (A,B) FIXED;                                        0300
A←X CAT Y+Z; % WHOLE SOURCE IMAGE IS INCLUDED              0700
B←A;                                                       0900
BUMP B;                                                     1300
```

NOTE THAT EVERY "IF" MUST BE PAIRED WITH EITHER AN "ELSE" OR AN "END". EVERY "ELSE" MUST HAVE AN "END" ASSOCIATED WITH IT.

APPENDIX VII: SDL PROGRAMMING TECHNIQUES

THIS SECTION CONTAINS CODING SUGGESTIONS AND EXAMPLES WHICH RESULT IN DECREASED SOURCE CODE AND/OR OBJECT CODE.

DECLARATIONS:

1. AS MANY NON-STRUCTURED DECLARATIONS AS POSSIBLE (UP TO A MAXIMUM OF 32) SHOULD BE DECLARED IN ONE <DECLARE STATEMENT>. EXAMPLE:

```
DECLARE A FIXED, (B,C) BIT(24);
```

GENERATES MORE EFFICIENT CODE THAN:

```
DECLARE A FIXED;
DECLARE (B,C) BIT(24);
```

2. A <DEFINE ACTUAL PARAMETER> (SEE "DEFINE INVOCATION") MAY BE A SERIES OF SDL STATEMENTS. FOR EXAMPLE:

```
DEFINE COMPARE(TS,S) AS#
     IF TOKEN.SYMBOL=TS
          THEN DO;
                 S;
                 UNDO THIS.ONE;
               END#;
```

MAY BE INVOKED AS:

```
DO THIS.ONE FOREVER;
     COMPARE ("SINGLE", SINGLE.SPACE  TRUE);
     COMPARE ("MERGE", IF LASTUSED ≠ 0
          THEN UNDO THIS.ONE;
          LASTUSED←2;
          OPEN SOURCE INPUT;
          READ SOURCE (TAPEWORK));
     COMPARE (...,...);
     .
     .
     .
END THIS.ONE;
```

PROCEDURES:

1. PROCEDURES FROM HIGHEST EFFICIENCY TO LOWEST ARE:

PARAMETERS	LOCAL DATA
NO	NO
NO	YES
YES	NO
YES	YES

STATEMENTS:

1. WHEN THE VALUE RETURNED BY A TYPED PROCEDURE IS TO BE IGNORED:

```
IF P(X-Y) THEN;
```

   IS MORE EFFICIENT THAN:

```
TEMP←P(X-Y);
```

2. THE FOLLOWING STATEMENTS:

```
<NULL STATEMENT>
<RETURN STATEMENT>
<GROUP TERMINATION STATEMENT>, AND
PARAMETER-LESS PROCEDURE CALLS
```

   SHOULD NEVER BE THE ONLY STATEMENT WITHIN A <DO GROUP> WHICH FOLLOWS A "THEN" OR "ELSE" OR WHICH IS AN ELEMENT OF A <CASE STATEMENT> (OR ANY "SUBORDINATE EXECUATBLE STATEMENT; SEE P. 4-2). EXAMPLE:

```
IF A THEN RETURN;
```

   IS MORE EFFICIENT THAN:

```
IF A THEN DO;
          RETURN;
        END;
```

3. USE "%" AT THE BEGINNING OF A COMMENT RATHER THAN "/*...*/" AS DELIMITERS. THE "%" STOPS THE SCANNING OF THAT RECORD. IF THE "/*...*/" FORM IS USED, SCANNING MUST CONTINUE TO

DETECT THE ENDING TERMINATOR. THUS COMPILE TIME IS INCREASED.

4. THE EXPRESSION:

```
SUBSTR("0123456789ABCDEF",N,1)
```

GENERATES MUCH LESS CODE THAN

```
CASE N OF ("0","1","2",..."E","F")
```

5. THE STATEMENT:

```
X←A>0;
```

IS MORE EFFICIENT THAN

```
X←IF A>0 THEN 1 ELSE 0;
```

THE RESULTS ARE THE SAME.

6. BUMP A←B; STORES B INTO A AND BUMPS B.
BUMP A:←B; STORES B INTO A AND BUMPS A.

7. REVERSE.STORE(IF <CONDITION> THEN A ELSE B, C); SELECTIVELY STORES C INTO A OR B.

8. CONSIDER THE FOLLOWING:

IN A COMPILER, FOR EXAMPLE, ASSUME THAT ALL CALLS ON THE ERROR ROUTINE FOLLOW A THEN/ELSE OR ARE IN A <CASE STATEMENT>. EXAMPLE:

```
1.  IF <CONDITION> THEN ERROR(E005);
2.  CASE N;
        .;
        .;
        .;
        ERROR(E137);
        .;
    END CASE;
```

IT IS SOMETIMES DESIRABLE TO PUT THESE CALLS INTO A SEPARATE SEGMENT, ESPECIALLY WHEN E005 AND E137 REPRESENT CHARACTER STRINGS (I.E., "IN-LINE" ERROR MESSAGES). FOR EXAMPLE:

```
DEFINE ERROR(N) AS #SEGMENT (ERROR.CALLS);
```

```
ERROR.ROUTINE (N)#;
```

BECAUSE OF THE TEMPORARY NATURE OF SEGMENTING SUBORDINATE EXECUTABLE STATEMENTS, ONLY THE CALLS WILL BE IN SEPARATE SEGMENTS.

9. WHEN TWO OR MORE ELEMENTS OF A <CASE STATEMENT> OR AN <IF STATEMENT> HAVE IDENTICAL CODE, MORE EFFICIENT CODE IS GENERATED IF THE CODE IS PUT INTO A SEPARATE PROCEDURE (WITH NO PARAMETERS OR DATA). IN BOTH CASES, EXECUTION TIME WILL BE IDENTICAL, BUT OBJECT CODE SAVINGS COULD BE SUBSTANTIAL.

10. USE CONDITIONAL COMPILATION STATEMENTS TO REMOVE DEBUGGING CODE, RATHER THAN PHYSICALLY REMOVING THE CODE. SEE APPENDIX VI.

APPENDIX VIII: THE SDL RECOMPILATION FACILITY

RECOMPILATION OF SDL PROGRAMS IS A TWO PART PROCESS: (A) COMPILATION OF THE FULL PROGRAM IN ORDER TO CREATE THE MASTER INFORMATION FILES, AND (B) RECOMPILATION OF PARTS OF THE PROGRAM USING THE MASTER INFORMATION FILES. THE "CREATE MASTER" OPERATION TAKES PLACE ONCE, AND IS THEN FOLLOWED BY SUCCESSIVE "RECOMPILE" OPERATIONS. NOTE THAT BOTH A "CREATE MASTER" AND A "RECOMPILE" MAY BE PERFORMED AT THE SAME TIME. IN ADDITION, IT IS POSSIBLE TO DO "REGULAR" COMPILATIONS WITHOUT INVOKING ANY OF THE RECOMPILATION FACILITY.

A. THE CREATE MASTER OPERATION

IN ORDER TO CREATE THE MASTER INFORMATION FILES, IT IS NECESSARY TO INCLUDE AS THE FIRST CARD IN THE COMPILATION DECK THE $-CARD:

```
$ CREATE.MASTER
```

THIS CAUSES THE COMPILER TO: (A) DUMP INFORMATION TO THE MASTER INFORMATION FILES, (B) CREATE A NEW SOURCE FILE (ABOUT WHICH, INFORMATION IS SAVED), AND (C) USE THESE TO CREATE A NEW CODE FILE.

THE INFORMATION THAT IS SAVED IS: THE INPUT SOURCE IMAGES, THE LEXIC LEVEL ONE PROCEDURE BOUNDARIES (BOTH SOURCE AND OBJECT), THE LEXIC LEVEL ZERO SYMBOL TABLES, A RECORD OF ALL CODE ADDRESSES THAT HAVE BEEN EMITTED, THE OBJECT CODE FROM WHICH CODE ADDRESSES HAVE BEEN EXCISED, AND THE FPB≥S AND SCRATCHPADS. THIS IS SAVED IN THE FILES:

NEWSOURCE,
NEW.INFO.FILE,
NEW.BLOCK.ADDRESS.FILE,
NEW.SECONDARY.FILE, AND
NEW.FPB.FILE.

THERE ARE SEVERAL RESTRICTIONS THAT APPLY TO THE CREATE MASTER OPERATION:

1. $ CREATE.MASTER MUST BE THE FIRST CARD IN THE DECK.

2. $ NEW NEED NOT BE USED WITH $ CREATE.MASTER.

3. $ SEQ SHOULD BE USED IF THERE ARE INPUT SOURCE IMAGES WITH EMPTY SEQUENCE FIELDS.

NOTE THAT (1) IMPLIES THAT THE &-CARDS AT THE FRONT OF THE DECK WILL BE INCLUDED ON THE NEW SOURCE FILE THAT IS CREATED. HENCE, THESE CARDS SHOULD BE SEQUENCED.

B. THE RECOMPILE OPERATION

RECOMPILATION IS DONE ON A LEXIC LEVEL ONE PROCEDURE BASIS: I.E., THE OUTERMOST PROCEDURE CONTAINING A PATCH IS THE PROCEDURE WHICH IS RECOMPILED. THE CODE PRODUCED BY THE RECOMPILE PROCESS WILL BE MERGED INTO (AND, IN FACT, REPLACE SOME OF) THE INFORMATION CREATED DURING THE CREATE MASTER PROCESS.

RECOMPILATION IS INVOKED BY INCLUDING AS THE FIRST CARD OF THE PATCH DECK:

```
$ RECOMPILE
```

THIS CAUSES THE COMPILER TO USE THE PATCHES AND THE MASTER INFORMATION FILES TO LOCATE THE LEXIC LEVEL ONE PROCEDURES AND TO GENERATE THE SAME INFORMATION FOR THESE AS WAS GENERATED FOR THE ENTIRE PROGRAM IN THE CREATE MASTER OPERATION. THIS INFORMATION IS THEN COMBINED, ON A PROCEDURE BY PROCEDURE BASIS, WITH THE MASTER INFORMATION FILES TO PRODUCE A FINAL FORM OF THE PROGRAM WHICH CAN BE TURNED INTO A CODE FILE.

THE INFORMATION THAT IS ACCESSED IS CONTAINED IN THE FILES:

```
SOURCE,
MASTER.INFO.FILE,
```

MASTER.BLOCK.ADDRESS.FILE,
MASTER.SECONDARY.FILE, AND
MASTER.FPB.FILE.

THERE ARE SEVERAL RESTRICTIONS THAT APPLY TO THE RECOMPILE OPERATION:

1. $ RECOMPILE MUST BE THE FIRST CARD IN THE DECK.

2. THE PATCH DECK MAY CONTAIN $-CARDS, & SET CARDS, AND & RESET CARDS, FOLLOWED BY PATCH CARDS.

3. NO LEXIC LEVEL ZERO CARDS MAY BE PATCHED: THIS INCLUDES GLOBAL DATA, LEXIC LEVEL ONE PROCEDURE HEADS, AND THE MAIN PROGRAM.

4. NEITHER $ SEQ NOR $ MERGE MAY APPEAR WITH $ RECOMPILE.

5. THE SOURCE FILE INPUT DURING RECOMPILATION MUST BE ON DISK (IN ORDER THAT IT MAY BE ACCESSED RANDOMLY).

C. THE CREATE MASTER AND RECOMPILE OPERATIONS PERFORMED TOGETHER

IT IS POSSIBLE TO CREATE NEW MASTER INFORMATION FILES WHILE RECOMPILING. IT IS NECESSARY ONLY TO ADHERE TO THE SET OF RESTRICTIONS LISTED IN BOTH A AND B, ABOVE. IN PARTICULAR,

```
$ RECOMPILE CREATE.MASTER
```

MUST BE THE THE FIRST CARD IN THE DECK. IT SHOULD BE NOTED, HOWEVER, THAT PERFORMING THESE OPERATIONS TOGETHER UPDATES SOME OF THE INFORMATION IN THE FILE MASTER.INFO.FILE. HENCE, IT WILL NOT BE POSSIBLE TO PERFORM ANOTHER RECOMPILE USING THIS MASTER.INFO.FILE UNLESS THE FILE HAS BEEN COPIED. IN THIS CASE, THE COPIED VERSION IS THE ONE WHICH MUST BE USED FOR SUBSEQUENT RECOMPILATIONS.

D. NOTES

1. DURING RECOMPILATION, THE ONLY INFORMATION WHICH MAY BE LISTED IS THAT WHICH IS ACTUALLY BEING RECOMPILED.

2. THE ORIGINAL SOURCE FILE USED WITH $ CREATE.MASTER MAY BE ON MAG TAPE. THE NEW SOURCE FILE CREATED BY $ CREATE.MASTER MAY BE ON MAG TAPE. HOWEVER, THIS NEW SOURCE FILE MUST THEN BE COPIED TO DISK BEFORE PERFORMING THE RECOMPILE OPERATION.

3. USE OF THE RECOMPILATION FACILITY REQUIRES A FAIR AMOUNT OF DISK SPACE, PARTICULARLY WHEN BOTH RECOMPILATION AND CREATE MASTER OPERATIONS ARE PERFORMED AT THE SAME TIME. IT IS ADVANTAGEOUS TO KEEP SOURCE FILES ON MAG TAPE AS MUCH AS POSSIBLE, COPY THEM TO DISK WHEN THEY ARE NEEDED, AND REMOVE THEM AS SOON AS THEY ARE RELEASED.

4. THE MARK III.1 MCP REQUIRES THAT ALL INTERNAL FILE NAMES BE LIMITED TO TEN CHARACTERS; THE MARK III.2 RESTRICTION IS 63 CHARACTERS, OF WHICH ONLY THE FIRST TEN ARE SIGNIFICANT.

5. THE SOURCE IMAGE FILE CREATED BY THE CREATE MASTER PROCESS CONTAINS NO INFORMATION OTHER THAN SOURCE IMAGES. IT MAY THEREFORE BE USED IN A REGULAR COMPILATION.

E. SAMPLE COMPILE DECKS

1. COMPILE, CREATE.MASTER:

```
? COMPILE SA SDL LIBRARY
? FILE SOURCE NAME SA0206/SOURCE TAPE;
? FILE NEWSOURCE NAME SA0410/SOURCE TAPE;
? FILE NEW.INFO.FILE NAME SA0410/INFO;
? FILE NEW.BLOCK.ADDRESS.FILE NAME SA0410/BLOCK.ADDRESS;
? FILE NEW.SECONDARY.FILE NAME SA0410/SECONDARY;
? FILE NEW.FPB.FILE NAME SA0410/FPB;
? DATA CARDS
$ CREATE.MASTER
$ MERGE LIST SINGLE SIZE SEQ
[PATCH CARDS]
[99999999 CARD]
? END
```

2. RECOMPILE (IT IS ASSUMED THAT THE TAPE LABELLED SA0410/SOURCE HAS BEEN COPIED TO A DISK FILE OF THE SAME NAME):

```
? COMPILE SA SDL LIBRARY
? FILE SOURCE NAME SA0410/SOURCE;
? FILE MASTER.INFO.FILE NAME SA0410/INFO;
? FILE MASTER.BLOCK.ADDRESS.FILE NAME SA0410/BLOCK.ADDRESS;
? FILE MASTER.SECONDARY.FILE NAME SA0410/SECONDARY;
```

```
? FILE MASTER.FPB.FILE NAME SA0410/FPB;
? DATA CARDS
$ RECOMPILE
$ LIST SINGLE SIZE
$ VSSIZE 10000 NSSIZE 100
[PATCH CARDS]
[99999999 CARD]
? END
```

3. RECOMPILE, CREATE.MASTER:

```
? COMPILE SA SDL LIBRARY
? FILE SOURCE NAME SA0410/SOURCE;
? FILE MASTER.INFO.FILE NAME SA0410/INFO;
? FILE MASTER.BLOCK.ADDRESS.FILE NAME SA0410/BLOCK.ADDRESS;
? FILE MASTER.SECONDARY.FILE NAME SA0410/SECONDARY;
? FILE MASTER.FPB.FILE NAME SA0410/FPB;
? FILE NEWSOURCE NAME SA0411/SOURCE TAPE;
? FILE NEW.INFO.FILE NAME SA0411/INFO;
? FILE NEW.BLOCK.ADDRESS.FILE NAME SA0411/BLOCK.ADDRESS;
? FILE NEW.SECONDARY.FILE NAME SA0411/SECONDARY;
? FILE NEW.FPB.FILE NAME SA0411/FPB;
? DATA CARDS
$ RECOMPILE CREATE.MASTER
$ VSSIZE 10000 NSSIZE 100
$ LIST SINGLE SIZE
[PATCH CARDS]
? END
```

APPENDIX IX: SDL MONITORING FACILITY

INTRODUCTION:

THE SDL MONITORING FACILITY PROVIDES A MEANS OF MONITORING PROCEDURE INVOCATIONS AND EXITS. THE MONITOR OUTPUT INCLUDES POINTS OF INVOCATION, PROCEDURE NAMES, PARAMETER VALUES, RETURNED VALUES, ETC.

SDL PROGRAM SYNTAX AND SEMANTIC CHANGES

A. "MONITOR.INPUT.FILE" AND "MONITOR.OUTPUT.FILE" ARE ADDED AS <FILE ATTRIBUTE> S TO THE <FILE DECLARATION STATEMENT>. THEY HAVE THE FOLLOWING SEMANTICS:

   1. BOTH CANNOT BE ATTRIBUTES OF THE SAME FILE.

   2. EACH MAY APPEAR ONLY ONCE IN A COMPILATION.

   3. "MONITOR.OUTPUT.FILE" MAY APPEAR ALONE.

B. "NO MONITOR" AND "MONITOR" ARE ADDED AS <CONTROL OPTION>S TO SDL. THEY HAVE THE FOLLOWING SEMANTICS:

   1. THE "MONITOR" <CONTROL OPTION> IS INITIALLY OFF.

   2. RECOGNIZING THE ("MONITOR"/"NO MONITOR") <CONTROL OPTION> TURNS THAT OPTION (ON/OFF).

GENERAL DISCUSSION

THE MONITORING SYSTEM WILL MONITOR PROCEDURE INVOCATIONS AND EXITS ON THE FILE WITH <FILE ATTRIBUTE> "MONITOR.OUTPUT.FILE". PROCEDURES WHICH ARE CANDIDATES TO BE MONITORED ARE THOSE WHOSE <FORWARD DECLARATION>S OR <PROCEDURE STATEMENT>S WERE RECOGNIZED WHILE THE "MONITOR" <CONTROL OPTION> WAS "ON".

THE SET OF PROCEDURES TO BE MONITORED FOR A GIVEN RUN IS

SELECTED VIA <RUN-TIME MONITOR STATEMENT>S WHICH ARE INPUT AT THE BEGINNING OF THE PROGRAM VIA THE FILE WITH THE "MONITOR.INPUT.FILE" ATTRIBUTE. IF NO FILE WITH THIS ATTRIBUTE IS DECLARED THE SPO IS USED BY DEFAULT.

A. FILE HANDLING

THE FILE WITH <FILE ATTRIBUTE> "MONITOR.INPUT.FILE" WILL BE OPENED, ACCESSED, AND CLOSED BEFORE THE USER CODE RECEIVES CONTROL.

THE FILE WITH <FILE ATTRIBUTE> "MONITOR.OUTPUT.FILE" WILL BE WRITTEN TO ONLY; NO EXPLICIT OPEN OR CLOSE WILL BE PERFORMED. THIS MEANS THAT (A) ONCE MONITOR OUTPUT IS STARTED THE USER SHOULD NOT PERFORM AN EXPLICIT OPEN ON THE FILE AND (B) IF MORE MONITOR OUTPUT IS EXPECTED THE USER SHOULD NOT PERFORM AN EXPLICIT CLOSE ON THE FILE.

RUN-TIME MONITOR STATEMENT

THE SYNTAX OF THE RUN-TIME MONITOR STATEMENT IS DESCRIBED BELOW.

<RUN-TIME MONITOR STATEMENT>::=	$ALL	1
	/ $NONE	2
	/ <EXPRESSION>	3
<EXPRESSION> ::=	<TERM>	4
	/ <EXPRESSION> <OR> <TERM>	5
<TERM> ::=	<FACTOR>	6
	/ <TERM> <AND> <FACTOR>	7
<FACTOR> ::=	<PRIME>	8
	/ <NOT> <PRIME>	9
<PRIME> ::=	<RANGE>	10
	/ <LIST>	11
	/ (<EXPRESSION>)	12
<OR> ::=	OR	13
	/ +	14
<AND> ::=	AND	15
	/ *	16
<NOT> ::=	NOT	17
	/ -	18

```
<RANGE> ::=                    <8 DIGIT SEQUENCE NUMBER> - 19
                               <8 DIGIT SEQUENCE NUMBER>   20

<LIST> ::=                     <IDENTIFIER>                21
                             / <LIST> <IDENTIFIER>         22
                             / <LIST>, <IDENTIFIER>        23
```

THE SEMANTICS OF A <RUN-TIME MONITOR STATEMENT> ARE DESCRIBED AS FOLLOWS. IF A PROCEDURE "P" SATISFIES THE CONDITIONS OF THE MONITOR STATEMENT THEN IT SHOULD BE MONITORED. NOTE THAT ALL PROCEDURES SATISFY (1) AND NO PROCEDURES SATISFY (2). A PROCEDURE SATISFIES AN <EXPRESSION> IF IT SATIFIES ANY <TERM> OF THAT <EXPRESSION>. A PROCEDURE SATISFIES A <TERM> IF IT SATISFIES ALL <FACTOR>S OF THAT <TERM>. A PROCEDURE SATISFIES A <FACTOR> IF IT SATISFIES THE <PRIME> COMPOSING IT (8) OR IT DOES NOT SATISFY THAT PRIME (9). A PROCEDURE SATISFIES A <PRIME> IF IT SATISFIES A RANGE CONDITION (10), A LIST CONDITION (11), OR A SUB-EXPRESSION (12). A RANGE CONDITION IS TWO 8 DIGIT SEQUENCE NUMBERS, THE FIRST STRICTLY LESS THAN THE SECOND. A PROCEDURE SATISFIES A RANGE CONDITION IF ITS PROCEDURE STATEMENT IS CONTAINED IN THAT RANGE. A PROCEDURE STATEMENT SATISFIES A LIST CONDITION IF ITS NAME IS CONTAINED IN THE LIST.

THE MONITOR FILE

WHEN MONITORING A PROGRAM, THE MONITORING INTRINSICS REFERENCE A RANDOM ACCESS FILE ASSOCIATED WITH THE COMPILATION OF THE PROGRAM. THE NAME OF THE CODE FILE AND THE NAME OF ITS MONITOR FILE ARE GIVEN BELOW:

CODE FILE	MONITOR FILE
A	$$A
A/B	A/$$B
A/B/C	A/B/$$C

APPENDIX X: BURROUGHS B1700 DATA COMMUNICATIONS SDL (DCSDL)

INTRODUCTION

B 1700 DATA COMMUNICATIONS SDL (DCSDL) IS AN EXTENSION OF B 1700 SDL, AND ANY CONSTRUCT USABLE IN B 1700 SDL IS PERMITED IN DCSDL.

DCSDL HAS BEEN PROVIDED SPECIFICALLY FOR THE IMPLEMENTATION OF THE NETWORK CONTROLLER FOR DATA COMMUNICATIONS NETWORK. IT SHOULD NOT BE USED FOR ANY OTHER PURPOSE. NOTE THAT THERE ARE SOME FEATURES OF DCSDL WHICH EITHER HAVE NOT BEEN FULLY IMPLEMENTED OR HAVE NOT BEEN IMPLEMENTED AT ALL. THESE FEATURES WILL BE MARKED AS SUCH.

FOR SIMPLICITY, ONLY THOSE CONSTRUCTS OF DCSDL THAT ARE NEW AND THAT HAVE NOT BEEN PREVIOUSLY DEFINED IN B1700 SDL WILL BE DESCRIBED IN THIS APPENDIX. THE SYNTACTIC DESCRIPTIONS WILL BE CONSISTENT WITH THOSE OF B1700 SDL, AND ANY METALINGUISTIC VARIABLE NOT DEFINED IN THIS MANUAL HAS THE SAME DEFINITION AS IN THE B1700 SDL INFORMATION MANUAL.

DCSDL EXTENSIONS

SYNTAX:

<DCSDL EXTENSION> ::= <DCSDL DECLARATION> / <DCSDL FUNCTION>/
<DCSDL STATEMENT>

SEMANTICS:

THE EXTENSIONS TO B1700 CONSIST OF TWO NEW DECLARATIONS, A NUMBER OF STATEMENTS, AND SOME SPECIALIZED FUNCTIONS THAT DEAL WITH THE NEW DECLARATIONS.

NOTE: THE WORDS "TRUE" AND "FALSE" IN THIS APPENDIX ARE SYNONOMOUS WITH ●(1)1● AND ●(1)0●, RESPECTIVELY.

DCSDL DECLARATIONS

SYNTAX:

<DCSDL DECLARATION> ::= <B1700 SDL DECLARATIONS> /
<QUEUE DECLARATION> / <MESSAGE DECLARATION>

SEMANTICS:

DCSDL DECLARATIONS MAY BE THE <B1700 SDL DECLARATIONS>, OR THE <QUEUE DECLARATION>, OR THE <MESSAGE DECLARATION>.

QUEUE DECLARATIONS

SYNTAX:

<QUEUE DECLARATION> ::= QUEUE <QUEUE LIST>

<QUEUE LIST> ::= <COMPLEX QUEUE IDENTIFIER> / <QUEUE LIST>,
<COMPLEX QUEUE IDENTIFIER>

<COMPLEX QUEUE IDENTIFIER> ::= <QUEUE IDENTIFIER> /
<QUEUE ARRAY IDENTIFIER> (<ARRAY BOUND>)

<QUEUE IDENTIFIER> ::= <IDENTIFIER>

<QUEUE ARRAY IDENTIFIER> ::= <IDENTIFIER>

<ARRAY BOUND> ::= <INTEGER>

EXAMPLES:

```
QUEUE   A, B, C (10);

QUEUE   QUEUE.ARRAY (15), PRIMARY (7);
```

SEMANTICS:
---

1. A <QUEUE DECLARATION> DEFINES ONE OR MORE IDENTIFIERS AS QUEUES OR QUEUE ARRAYS AND DEFINES THEIR BOUNDS OF SUBSCRIPT. QUEUE ARRAYS MUST BE ONE DIMENSIONAL.

2. THE SUBSCRIPT BOUND FOR EACH QUEUE ARRAY IS GIVEN IN THE <ARRAY BOUND> FOLLOWING THE QUEUE ARRAY IDENTIFIER(S) IN THE <QUEUE ARRAY LIST>.

3. EXPRESSIONS MAY BE USED AS QUEUE ARRAY SUBSCRIPTS.

4. QUEUE ARRAYS MUST BE DECLARED IN THE OUTERMOST BLOCK.

5. A DCSDL QUEUE IS A LINKED LIST OF MESSAGES.

6. A QUEUE ARRAY IS AN INDEXABLE ARRAY WHOSE ELEMENTS ARE QUEUES, NOT VALUES.

7. QUEUES AND QUEUE ARRAYS MAY NOT BE PASSED AS PARAMETERS TO PROCEDURES.

PRAGMATICS:

A DATA DICTIONARY ENTRY IS RESERVED FOR EACH QUEUE OR QUEUE ARRAY. IF THE QUEUE IS INACTIVE (A NULL QUEUE) THE ENTRY WILL BE ZERO. IF THE QUEUE IS ACTIVE, THE ENTRY WILL BE A DICTIONARY ENTRY WHICH POINTS TO A LOCATION IN THE DATACOM QUEUE STACK. WHEN A QUEUE IS ACTIVE, ITS ENTRY IN THE QUEUE STACK CONTAINS THE HEAD AND TAIL LINKS TO A LINKED LIST OF MESSAGES. WHEN THE QUEUE IS EMPTY, THE TAIL LINK POINTS TO THE HEAD LINK; HOWEVER, THE QUEUE IS STILL CONSIDERED ACTIVE.

QUEUES MAY BE REFERENCED BY THE USE OF THE NEW STATEMENTS AND FUNCTIONS DESCRIBED IN THIS DOCUMENT. A QUEUE MAY NOT BE ASSIGNED A VALUE OR USED AS AN INDEX.

A QUEUE MAY BE ACTIVATED WITH AN ENABLE STATEMENT AND DE-ACTIVATED WITH THE DISABLE STATEMENT.

A QUEUE BECOMES NULL (AND ITS MESSAGES, IF ANY, REMOVED) ONLY WHEN THERE ARE NO PROCESSES REFERRING TO IT.

MESSAGE DECLARATION

SYNTAX:

<MESSAGE DECLARATION> ::= MESSAGE <MESSAGE LIST>

<MESSAGE LIST> ::= <COMPLEX MESSAGE IDENTIFIER> / <MESSAGE LIST>, <COMPLEX MESSAGE IDENTIFIER>

<COMPLEX MESSAGE IDENTIFIER> ::= <MESSAGE IDENTIFIER> / <MESSAGE ARRAY IDENTIFIER> (<ARRAY BOUND>)

<MESSAGE IDENTIFIER> ::= <IDENTIFIER>

<MESSAGE ARRAY IDENTIFIER> ::= <IDENTIFIER>

<ARRAY BOUND> ::= <INTEGER>

EXAMPLES:

```
MESSAGE   A, B, C ;

MESSAGE   FROM.A (10), X, 4, ALPHA.1 (5);
```

SEMANTICS:

1. A <MESSAGE DECLARATION> DEFINES ONE OR MORE IDENTIFIERS AS MESSAGES OR MESSAGE ARRAYS AND DEFINES THEIR BOUNDS.

2. THE SUBSCRIPT BOUND FOR EACH MESSAGE ARRAY IS GIVEN IN THE <ARRAY BOUND> FOLLOWING THE MESSAGE ARRAY IDENTIFIER IN THE <MESSAGE ARRAY LIST>.

3. THE <ARRAY BOUND> GIVES UPPER BOUNDS OF ALL SUBSCRIPTS.

4. ESSENTIALLY, MESSAGES ARE SPECIAL WORK AREAS OUTSIDE THE BASE LIMIT OF INDIVIDUAL PROCESSES.

5. MESSAGE ARRAYS MUST BE ONE DIMENSIONAL. THE NUMBER OF BYTES IN THE MESSAGE IS SET BY THE ALLOCATE STATEMENT AND CAN BE DETERMINED BY THE SIZE OPTION IN THE QUEUE.INFO FUNCTION.

6. MESSAGE ARRAYS MUST BE DECLARED IN THE OUTERMOST BLOCK.

7. MESSAGES CANNOT BE PASSED AS PARAMETERS TO PROCEDURES.

MESSAGES ARE ORGANIZED AND MAINTAINED BY A SET OF PROCEDURES WITHIN THE MCP. THESE PROCEDURES MAINTAIN A LIST OF FREE SPACE WITHIN THE SAVE-MEMORY POOL AND ALLOCATE THIS SPACE TO PROCESSES THAT REQUEST IT, BY MAKING A DESCRIPTOR FOR AN AREA AND PASSING THAT. THE INTRINSICS ALSO ACCEPT SPACE FROM PROCESSES FINISHED WITH AN AREA AND ADD IT TO THE AVAILABLE LIST. IF AT ANY TIME THE INTRINSICS SEEM TO BE RUNNING OUT OF SPACE THEY GET A NEW SUBPOOL AND APPROPRIATELY SUBDIVIDE IT.

MESSAGE LINKAGE MECHANISM

DATA IN MESSAGES WILL BE ACCESSED ONLY THROUGH THE DATA DICTIONARY. A NEW SDL S-OPERATOR (TRANSFER.MESSAGE) IS DEFINED TO PERMIT THIS ACCESS TECHNIQUE. THESE OPERATORS WILL PERFORM THE FOLLOWING FUNCTIONS.

1. COMPUTE THE DATA DICTIONARY ENTRY NUMBER (I.E. INDEX)

2. FIND OUT THE LOCATION OF THE DATA DICTIONARY THROUGH THE FIELD RS.DATA.DIC IN THE RUN STRUCTURE NUCLEUS

3. COMPUTE THE ABSOLUTE ADDRESS OF THE DATA DICTIONARY ENTRY

4. DETERMINE THE ABSOLUTE LOCATION OF A MESSAGE AND ITS LENGTH

5. SET THE BASE-LIMIT OVER-RIDE FLIP FLOP CD(2) IF WRITE OPERATION ELSE CD(1)

6. TRANSFER INFORMATION FROM THE MESSAGE TO A LOCAL DATA AREA OR VICE VERSA. DESCRIPTORS ON THE EVALUATION STACK WILL DESCRIBE THE DATA TO BE TRANSFERRED. THE INTERPRETER WILL CHECK THE VALIDITY OF ANY ADDRESSES OUTSIDE BASE-LIMIT.

THE INTERPRETER WILL NOT CHANGE MESSAGE DESCRIPTORS IN THE DATA DICTIONARY. ONLY THE MCP WILL HAVE THIS PRIVILEGE. IF A MESSAGE IS NOT ASSIGNED TO A DATA DICTIONARY ENTRY AND IF THE PROGRAM TRIES TO READ OR WRITE TO IT AN INTERRUPT WILL BE CAUSED.

THIS MECHANISM IS SAFE. A USER PROGRAM (SDL) WILL NOT HAVE THE ABILITY TO WRITE INTO A MEMORY SPACE THAT HAS NOT BEEN ALLOCATED TO IT BECAUSE ALL ADDRESSING IS INDIRECT THROUGH THE DATA DICTIONARY.

DCSDL FUNCTIONS

SYNTAX:

<DCSDL FUNCTION> ::= <B1700 SDL FUNCTION> / <QUEUE FUNCTION>

<QUEUE FUNCTION> ::= <ALLOCATE FUNCTION>

SEMANTICS:

DCSDL FUNCTIONS MAY BE THE <B1700 SDL FUNCTION>, OR THE <QUEUE FUNCTION>.

ALLOCATE FUNCTION

SYNTAX:

<ALLOCATE FUNCTION> ::= ALLOCATE (<COMPLEX MESSAGE IDENTIFIER>, <SIZE>)

<SIZE> ::= <ARITHMETIC EXPRESSION>

EXAMPLES:

```
RESULT ::= ALLOCATE (MESSAGEID, SIZEINBITS);

RESULT ::= ALLOCATE (MESSAGEARRAYID(3), SIZEINBITS);
```

SEMANTICS:
---

1. THE <ALLOCATE FUNCTION> CAUSES AN AREA OF SAVE MEMORY, <SIZE> BITS IN LENGTH, TO BE RESERVED FOR A MESSAGE.

2. WHEN THE <SUB MESSAGE ARRAY> (OR <MESSAGE ARRAY IDENTIFIER>) FORM IS USED, REPEATED ALLOCATES ARE NEEDED FOR EACH ELEMENT OF THE SUB MESSAGE ARRAY (OR MESSAGE ARRAY), RESERVING <SIZE> BITS FOR EACH ELEMENT.

3. IF A MESSAGE AREA REFERENCED BY <COMPLEX MESSAGE IDENTIFIER> HAS ALREADY BEEN ALLOCATED, THE OLD MESSAGE AREA IS RETURNED TO THE SYSTEM BEFORE A NEW AREA IS ALLOCATED. THIS FEATURE HAS NOT BEEN IMPLEMENTED.

4. IF <SIZE> IS EQUAL TO ZERO, THE MESSAGE AREA, IF ANY, IS RETURNED TO THE SYSTEM, MAKING THE MESSAGE NULL. THIS FEATURE HAS NOT BEEN FULLY IMPLEMENTED.

5. THE VALUE RETURNED IS THE 24 BIT ADDRESS OF THE MESSAGE.

DCSDL STATEMENTS

SYNTAX:

<DCSDL STATEMENT> ::= <B1700 SDL STATEMENT> / <QUEUE STATEMENT>

<QUEUE STATEMENT> ::= <DC.WRITE STATEMENT> / <REMOVE STATEMENT>/
<QUEUE.INFO STATEMENT> / <MESSAGE.INFO STATEMENT> /
<DE.ALLOCATE STATEMENT> / <INSERT STATEMENT> /
<FLUSH STATEMENT> / <ENABLE.QUEUE STATEMENT> /
<DISABLE.QUEUE STATEMENT> / <TRANSFER.MESSAGE STATEMENT> /
<MCS.COMMUNICATE STATEMENT> /

SEMANTICS:

THE <DCSDL STATEMENT> MAY BE A <B1700 SDL STATEMENT>, OR AN <ALLOCATE STATEMENT>, OR A <QUEUE STATEMENT>.

DC.WRITE STATEMENT

SYNTAX:

<DC.WRITE STATEMENT> ::= DC.WRITE (<RESULT PART>, <MESSAGE
PART> <QUEUE PART PARAMETER>)

<RESULT PART> ::= <ADDRESS GENERATOR>

<MESSAGE PART> ::= <MESSAGE IDENTIFIER> / <MESSAGE ARRAY>
IDENTIFIER> (<SUBSCRIPT>)

<QUEUE PART PARAMETER> ::= , <QUEUE PART> / <EMPTY>

<QUEUE PART> ::= <QUEUE IDENTIFIER> / <QUEUE ARRAY IDENTIFIER>

(<SUBSCRIPT>)

<SUBSCRIPT> ::= <EXPRESSION>

EXAMPLES:

```
DC.WRITE (ERRORNO, MESSAGEID);
DC.WRITE (ERRORNO, MESSAGE.ARRAY.ID (3), QUEUEID);
```

SEMANTICS:
---

1. THE <DC.WRITE STATEMENT> CAUSES THE <MESSAGE PART> TO BE PASSED TO THE NETWORK CONTROLLER (NC).

2. THE ACTION TAKEN BY THE NC DEPENDS ON THE TYPE AND VARIANT FIELDS OF THE MESSAGE (AS SPECIFIED IN THE B1700 MCS MANUAL).

3. THE <QUEUE PART PARAMETER> IS REQUIRED FOR CERTAIN NC ACTIONS. (SEE THE B1700 MCS MANUAL).

4. THE VALUE RETURNED IS AN ERROR IDENTIFICATION NUMBER. A ZERO INDICATES NO ERROR; OTHER ERROR VALUES ARE GIVEN IN THE B1700 MCS MANUAL.

REMOVE STATEMENT

SYNTAX:

<REMOVE STATEMENT> ::= REMOVE (<ADDRESS GENERATOR>, <COMPLEX MESSAGE IDENTIFIER>, <QUEUE DESIGNATOR> <WAIT PART>)

<WAIT PART> ::= ,WAIT / <EMPTY>

EXAMPLES:

```
DECLARE
     01 RESULT
          ,02   MESSAGE.LENGTH   BIT(16)
          ,02   QUEUE.SUBSCRIPT  BIT(12)
          ,02   MESSAGE.ADDRESS  BIT(24)
          ;

MESSAGE M(3), MESG.X ;

QUEUE   Q(10), A, B ;

REMOVE (RESULT, M(2), A, WAIT);

REMOVE (RESULT, MESG.X, Q(3) );

REMOVE (RESULT, MSG.X, Q, WAIT);
```

THE LAST STATEMENT WILL SUSPEND THE PROCESS UNTIL A MESSAGE IS INSERTED IN ANY ELEMENT OF "Q". IF ANOTHER PROCESS INSERTS A MESSAGE OF 1000 BYTES IN Q(7) THEN THE FIRST PROCESS WILL BE RESUMED AND MESSAGE.LENGTH WILL CONTAIN 1000 AND QUEUE SUBSCRIPT WILL CONTAIN 7. MESSAGE.ADDRESS CONTAINS THE ABSOLUTE ADDRESS OF THE MESSAGE. A MESSAGE THAT BELONGS TO A PROGRAM WILL NOT BE OVERLAYED BY THE MCP.

SEMANTICS:
---

1. THE <REMOVE STATEMENT> DELINKS A MESSAGE FROM THE QUEUE SPECIFIED AND CAUSES IT TO BE REFERENCED BY THE <MESSAGE PART>.

2. IF THE <WAIT PART> IS NOT EMPTY THEN THE PROGRAM IS SUSPENDED UNTIL A MESSAGE IS INSERTED IN THE QUEUE OR QUEUE ARRAY. THE FIRST 16 BITS OF THE RETURNED VALUE CONTAIN THE SIZE OF THE MESSAGE AND THE NEXT 12 BITS CONTAIN A SUBSCRIPT INTO THE QUEUE ARRAY (WHERE APPLICABLE).

3. IF THE <WAIT PART> IS EMPTY THE <REMOVE STATEMENT> RETURNS A VALUE OF ZERO IF THERE ARE NO MESSAGE IN THE QUEUE. OTHERWISE, IT RETURNS, IN THE FIRST 16 BITS OF THE RECEIVING FIELD, THE LENGTH IN BITS OF THE MESSAGE REMOVED.

4. IF <MESSAGE PART> ALREADY REFERENCES A MESSAGE, THEN THAT MESSAGE AREA IS RETURNED TO THE SYSTEM BEFORE THE REMOVE. THIS FEATURE IS CURRENTLY NOT IMPLEMENTED.

5. IF THE <QUEUE PART> IS NOT ACTIVE, THE PROGRAM IS TERMINATED.

PRAGMATICS:

THE <REMOVE STATEMENT> TAKES THE MESSAGE THAT IS AT THE HEAD OF THE QUEUE, DELINKS IT FROM THAT QUEUE AND INSERTS A DESCRIPTOR POINTING TO THAT MESSAGE IN THE DATA DICTIONARY ENTRY REFERENCED BY THE <MESSAGE PART>.

CARE MUST BE TAKEN, WHEN REMOVING A MESSAGE FROM A QUEUE, THAT THE QUEUE IS ACTIVE. WHEN IN DOUBT, TEST USING QUEUE.INFO. (REMOVING A MESSAGE FROM AN INACTIVE QUEUE PRODUCES A RUN TIME ERROR AND TERMINATION BY "INACTIVEQUEUE").

QUEUE.INFO STATEMENT

SYNTAX:

<QUEUE.INFO STATEMENT> ::= QUEUE.INFO (<QUEUE.INFO RESULT>, <QUEUE PART> <INFO REQUEST>)

<QUEUE.INFO RESULT> ::= <ADDRESS GENERATOR>

<INFO REQUEST> ::= EMPTY /, NULL /, SIZE /, PROCESSES

<QUEUE PART> ::= <COMPLEX QUEUE IDENTIFIER> / <QUEUE ARRAY IDENTIFIER>

EXAMPLES:

```
QUEUE.INFO (ANSWER, QUEUEID, SIZE)

QUEUE.INFO (ANSWER, QUEUEARRAY (13), NULL);
```

SEMANTICS:

IF A <QUEUE ARRAY IDENTIFIER> IS USED (WITHOUT A SUBSCRIPT) THEN INFORMATION ABOUT ALL ELEMENTS OF THE QUEUE ARRAY IS RETURNED LEFT-JUSTIFIED IN <QUEUE.INFO RESULT>. OTHERWISE INFORMATION ABOUT THE SPECIFIED QUEUE IS RETURNED LEFT-JUSTIFIED IN <QUEUE.INFO RESULT>.

1. IF <INFO REQUEST> EQUALS "SIZE", THE NUMBER OF MESSAGES IN THE QUEUE IS RETURNED AS BIT(12).

2. IF <INFO REQUEST> EQUALS "PROCESSES", THE NUMBER OF PROCESSES ATTACHED TO THE QUEUE IS RETURNED AS BIT(12).

3. IF <INFO REQUEST> EQUALS "NULL", A BOOLEAN VALUE IS RETURNED AS BIT (1). "TRUE" IS RETURNED IF THE QUEUE IS ACTIVATED.

4. IF <INFO REQUEST> IS EQUAL TO "EMPTY" THEN "TRUE" IS RETURNED IF THE QUEUE ELEMENT HAS NO MESSAGES.

PRAGMATICS:

E.G.

```
     DECLARE
          A(20)            BIT(12),
          B REMAPS A       BIT(240);

     QUEUE   Q.I(20);

     QUEUE.INFO   (B,Q.I, SIZE);
```

PERFORMS THE SAME FUNCTION AS

```
     IO;

     DO   FOREVER;
```

```
        QUEUE.INFO (A (I), Q.I (I), SIZE).
        IF (BUMP I) = 20  THEN UNDO;
END;
```

MESSAGE.INFO STATEMENT

SYNTAX:

<MESSAGE.INFO FUNCTION::= MESSAGE.INFO (<MESSAGE.INFO RESULT>,
<MESSAGE PART>)

<MESSAGE.INFO RESULT> ::= <ADDRESS GENERATOR>

<MESSAGE PART> ::= <COMPLEX MESSAGE IDENTIFIER> /
<MESSAGE ARRAY IDENTIFIER>

EXAMPLES:

```
DECLARE
    01   MESSAGE.FIELD(5)       BIT(41)
         ,02  NULL.FIELD             BIT(1)
         ,02  SIZE.FIELD             BIT(16)
         ,02  ADDRESS.FIELD          BIT(24)
    ,B   REMAPS MESSAGE.FIELD  BIT(205)
    ;

MESSAGE   MSG (10) ;

MESSAGE.INFO (MESSAGE.FIELD (I), MSG (5) );

MESSAGE.INFO (B, MSG);
```

SEMANTICS:

THE <MESSAGE.INFO FUNCTION> RETURNS 3 VALUES. NULL.FIELD = 0 IMPLIES THAT THE MESSAGE IS NOT ALLOCATED. IF NULL.FIELD = 1 THEN THE SIZE (IN BITS) OF THE MESSAGE AND THE ABSOLUTE ADDRESS OF THE MESSAGE ARE RETURNED IN SIZE.FIELD AND ADDRESS.FIELD RESPECTIVELY. THE NAMES OF THE FIELDS USED ABOVE ARE ARBITRARY. THE FORMAT OF THE FIELDS MUST BE THE SAME AS INDICATED ABOVE.

IF <MESSAGE ARRAY IDENTIFIER> IS USED WITHOUT A SUBSCRIPT THEN THE RULES DEFINED FOR QUEUE.INFO WILL BE FOLLOWED; I.E., AN ARRAY OF RESULTS WILL BE RETURNED. THIS HAS NOT BEEN FULLY

IMPLEMENTED.

DE.ALLOCATE STATEMENT

SYNTAX:

<DE.ALLOCATE STATEMENT> ::= DE.ALLOCATE (<COMPLEX MESSAGE IDENTIFIER>)

EXAMPLES:

```
DE.ALLOCATE  (MSG (5) );

DE.ALLOCATE  (MESSAGE.ID);
```

SEMANTICS:

THE BUFFER ASSOCIATED WITH THE <COMPLEX MESSAGE IDENTIFIER> IS DE-ALLOCATED.

INSERT STATEMENT

SYNTAX:

<INSERT STATEMENT> ::= INSERT (<SOURCE PART>, <QUEUE PART> <PRIORITY>)

<SOURCE PART> ::= <COMPLEX MESSAGE IDENTIFIER>

<QUEUE PART> ::= <COMPLEX QUEUE IDENTIFIER>

<PRIORITY> ::= , TOP / <EMPTY>

EXAMPLES:

```
INSERT (MESSAGEID, QUEUEID);

INSERT (MESSAGEID, QUEUEID, TOP);

INSERT (MESSAGEARRAYID(1), QUEUEARRAYID(1));
```

SEMANTICS:
---

1. THE <INSERT STATEMENT> CAUSES THE MESSAGE REFERENCED BY <SOURCE PART> TO BE LINKED INTO THE QUEUE REFERENCED BY <QUEUE PART>.

2. IF <PRIORITY> EQUALS FALSE OR IS <EMPTY>, THE MESSAGE IS LINKED INTO THE TAIL OF THE QUEUE, OTHERWISE IT IS LINKED INTO THE HEAD OF THE QUEUE.

PRAGMATICS:

THE <INSERT STATEMENT> LINKS THE MESSAGE IDENTIFIED BY THE FIRST PARAMETER INTO THE QUEUE REFERENCED BY THE SECOND PARAMETER. IF <PRIORITY> IS NOT GIVEN, THE MESSAGE IS LINKED INTO THE END OF THE QUEUE, OTHERWISE IT IS LINKED INTO THE BEGINNING OF THE QUEUE. LINKING INTO THE END OF THE QUEUE WILL PRESERVE THE TIME ORDER OF THE MESSAGES IN THAT QUEUE; LINKING IT INTO THE BEGINNING INSURES THAT THAT MESSAGE WILL BE THE NEXT ONE REMOVED FROM THAT QUEUE, UNLESS ANOTHER MESSAGE IS LINKED INTO THE HEAD IN THE MEANTIME.

IF THE QUEUE REFERENCED IS INACTIVE (NULL), AN INVALID CONDITION IS CREATED AND THE PROCESS WILL BE TERMINATED.

FLUSH STATEMENT

SYNTAX:

<FLUSH STATEMENT> ::= FLUSH (<QUEUE PART>)

<QUEUE PART> ::= <COMPLEX QUEUE IDENTIFIER> / <QUEUE ARRAY IDENTIFIER>

EXAMPLES:

```
FLUSH (QUEUEID);

FLUSH (QUEUEARRAYID(3));
```

SEMANTICS:

1. THE <FLUSH STATEMENT> CAUSES ALL THE MESSAGES IN A QUEUE TO BE REMOVED FROM THE QUEUE.

2. IF THE <QUEUE PART> IS EMPTY, THE STATEMENT IS IGNORED.

3. IF THE QUEUE INDICATED BY <QUEUE PART> IS NOT ACTIVE, THE PROCESS IS TERMINATED.

PRAGMATICS:

THE <FLUSH STATEMENT> CAUSES ALL MESSAGES IN THE QUEUE TO BE DISCARDED. THE QUEUE REMAINS ACTIVE. MESSAGES WHICH HAVE BEEN FLUSHED FROM THE QUEUE NO LONGER EXIST IN THE SYSTEM.

ENABLE.QUEUE STATEMENT

SYNTAX:

<ENABLE.QUEUE STATEMENT> ::= ENABLE.QUEUE (<QUEUE DESIGNATOR>, <QUEUE NAME>)

<QUEUE DESIGNATOR> ::= <QUEUE IDENTIFIER> / <QUEUE ARRAY IDENTIFIER>

<QUEUE NAME> ::= <EXPRESSION>

EXAMPLES:

```
DECLARE     NAME.Q     CHARACTER (10);
QUEUE       A, Q(10);

ENABLE.QUEUE (A, "MY.QUEUE");

NAME.Q := "HIS.QUEUE" CAT "1";

ENABLE.QUEUE (Q, NAME.Q);
```

SEMANTICS:

ENABLE.QUEUE INITIALIZES A QUEUE/QUEUE ARRAY. IT IS NOT POSSIBLE TO INITIALIZE ONE ELEMENT OF A QUEUE ARRAY. <QUEUE NAME> DEFINES THE PHYSICAL NAME OF THE QUEUE AND CANNOT BE LONGER THAN 10 CHARACTERS.

QUEUES MUST BE INITIALIZED BEFORE THEY CAN BE USED IN <QUEUE FUNCTION>S AND <QUEUE STATEMENT>S (EXCEPT THE <QUEUE.INFO STATEMENT>).

IF THE QUEUE BEING ENABLED EXISTS (I.E. HAS BEEN ENABLED BY ANOTHER PROCESS) THEN THE CURRENT PROCESS IS ATTACHED TO IT ONLY IF THE NUMBER OF ELEMENTS IN THE QUEUE IS LESS THAN OR

EQUAL TO THE ELEMENTS IN THE EXISTING QUEUE. OTHERWISE THE PROGRAM IS TERMINATED.

DISABLE.QUEUE STATEMENT

SYNTAX:

<DISABLE.QUEUE STATEMENT> ::= DISABLE.QUEUE
(<QUEUE DESIGNATOR>)

<QUEUE DESIGNATOR> ::= <QUEUE IDENTIFIER /
<QUEUE ARRAY IDENTIFIER>

EXAMPLE

```
DISABLE.QUEUE (A);

DISABLE.QUEUE (Q);

DISABLE.QUEUE (Q(3)); IS ERRONEOUS.
```

SEMANTICS:

THE <DISABLE.QUEUE STATEMENT> DETACHES A PROCESS FROM A QUEUE. IF THE NUMBER OF PROCESSES ASSOCIATED WITH A QUEUE IS ZERO THEN THE QUEUE IS FLUSHED (IF REQUIRED) AND DE-ACTIVATED.

IT IS NOT POSSIBLE TO DISABLE AN ELEMENT OF A QUEUE ARRAY.

TRANSFER.MESSAGE STATEMENT

SYNTAX:

<TRANSFER.MESSAGE STATEMENT> ::= TRANSFER.MESSAGE
<SOURCE MESSAGE DESIGNATOR> TO <DESTINATION
MESSAGE DESIGNATOR>

<SOURCE MESSAGE DESIGNATOR>::= <EXPRESSION> /
<MESSAGE ADDRESS GENERATOR>

<MESSAGE ADDRESS GENERATOR> ::= <COMPLEX MESSAGE IDENTIFIER>/
<COMPLEX MESSAGE IDENTIFIER>, <ADDRESS GENERATOR>

<DESTINATION MESSAGE DESIGNATOR> ::= <ADDRESS GENERATOR> /
<MESSAGE ADDRESS GENERATOR>

EXAMPLES:

```
DECLARE
     01 WORK.AREA REMAPS BASE CHAR (1000)
          ,02 INBIT (100)
          ,02 MSG.TRAN.NUM CHAR(3)
          ;

MESSAGE M, FROM(3);

TRANSFER.MESSAGE "002" TO M,MSG.TRAN.NUM;

TRANSFER.MESSAGE  FROM(1)  TO M;
```

SEMANTICS:

TRANSFER.MESSAGE TRANSFERS DATA FROM <SOURCE MESSAGE DESIGNATOR>
TO <DESTINATION MESSAGE DESIGNATOR>. SOURCE AND DESTINATION
MESSAGE DESIGNATORS CAN BE DESCRIPTORS TO DATA INSIDE THE BASE
- LIMIT OF PROCESS OR THEY CAN REFERENCE SPECIFIC FIELDS IN

MESSAGES. A SPECIFIC FIELD WITHIN A MESSAGE IS REFERENCED BY SPECIFYING THE MESSAGE (WITH A <COMPLEX MESSAGE IDENTIFIER>) AND A TEMPLATE (REMAPS BASE DATA ITEM) WHICH SPECIFIES THE PART OF THE MESSAGE TO BE ACCESSED.

IF A MESSAGE IS TRANSFERRED TO ANOTHER MESSAGE THEN PHYSICAL MOVEMENT OF MESSAGE DOES TAKE PLACE. THE <ADDRESS GENERATOR> FIELDS DETERMINE THE TYPE OF MESSAGE TRANSFER, (I.E. LEFT JUSTIFIED BLANK FILLED OR RIGHT JUSTIFIED ZERO FILLED).

MCS COMMUNICATE STATEMENT

SYNTAX:

```
MCS COMMUNICATE ::= MCS.COMMUNICATE (<RESULT PART>,
          <MESSAGE PART>, <QUEUE PART>)

<RESULT PART> ::= <ADDRESS GENERATOR>

<MESSAGE PART> ::= <EXPRESSION>

<QUEUE PART> ::= <EMPTY> /, <QUEUE FILE DESIGNATOR>

<QUEUE FILE DESIGNATOR> ::= <FILE DESIGNATOR /
          <FILE DESIGNATOR> <KEY PART>

<FILE DESIGNATOR> ::= <FILE IDENTIFIER> /
          <SWITCH FILE IDENTIFIER> (<EXPRESSION>)

<KEY PART> ::= [<ADDRESS GENERATOR>]
```

SEMANTICS:

SEE THE B1700 MCS MANUAL FOR DETAILS.

PRAGMATICS:

THE PURPOSE OF THE MCS.COMMUNICATE STATEMENT IS TO PROVIDE COMMUNICATION BETWEEN THE MCS (MESSAGE CONTROL SYSTEM) AND THE NETWORK CONTROLLER.

APPENDIX XI: SDL CODING SUGGESTIONS

EACH PROGRAMMER HAS HIS OWN WAY OF DOING THINGS. WITHOUT STIFLING CREATIVITY, HOWEVER, IT IS DESIRABLE TO MAINTAIN SOME HOMOGENEITY IN CODING TECHNIQUES. THE ADVANTAGES ARE OBVIOUS: MORE UNDERSTANDABLE PROGRAMS AND EASIER MAINTENANCE (ESPECIALLY BY THOSE OTHER THAN THE PROGRAM-S AUTHOR).

ACCORDINGLY, THE FOLLOWING CODING GUIDELINES ARE SUGGESTED:

1. USE THE "TOP DOWN" PROGRAMMING METHOD. THINK IN TERMS OF FUNCTIONS AND AVOID A LOT OF INLINE CODE. THIS METHOD OF CODING IS MUCH EASIER TO FOLLOW BECAUSE IT KEEPS PROCEDURES SHORTER. THE ARGUMENT FOR SHORT PROCEDURES PARALLELS THE CASE AGAINST THE GO-TO: THE HUMAN SPAN OF ATTENTION IS SMALL--IF IT MUST RANGE OVER MANY PAGES OF CODING, THE READER MAY BECOME CONFUSED. YOU MIGHT GO SO FAR AS TO CRITICIZE ANY PROCEDURE THAT CONTAINS GREATER THEN N ("N" MIGHT BE AS LOW AS 3) LEVELS OF DO-ENDS OR IS LONGER THAN 1 PAGE. AFTER ALL, AREN-T WE PUSHING PROCEDURE-ORIENTED LANGUAGES?

2. MINIMIZE THE USE OF GLOBALS AND KEEP THE SCOPE OF A VARIABLE SMALL. OTHERWISE, YOU CAN RUN INTO SIDE EFFECTS THAT ARE DIFFICULT TO DEBUG. GLOBALS ARE SIMILAR TO GO TO≥S IN THAT THEY DEMAND A BROAD ATTENTION SPAN.

3. USE DEFINES INSTEAD OF INLINE LITERALS WHERE POSSIBLE. THIS MAKES CHANGES EASIER BECAUSE THEY ARE CONFINED TO THE DEFINE. LITERALS ARE NOT CROSS-REFERENCED.

4. DON-T DEFINE YOUR OWN NAMES AS RESERVED WORDS.

```
DEFINE F AS #FOREVER#;
DO F;
END;
```

   MAY BE CRYPTIC TO ANOTHER PROGRAMMER.

5. BEWARE OF OVERUSE OF THE DEFINE:

   A. NESTING DEFINES TOO DEEPLY CAN CAUSE CONFUSION.

   B. A DEFINE THAT CONTAINS A GREAT DEAL OF CODING MIGHT

BETTER BE A PROCEDURE.

C. CREATING YOUR OWN PRIVATE LANGUAGE BY USING DEFINES CAN DESTROY EASE OF UNDERSTANDING.

6. DON-T USE CONSTRUCTS THAT OBSCURE THE PROGRAM-S MEANING--ESPECIALLY BEWARE OF UNSAFE CONSTRUCTS SUCH AS DESCRIPTOR, SAVE AND RESTORE. AVOID TRICKY CODING TECHNIQUES--CODE IN A STRAIGHT FORWARD MANNER. COMPLICATED CODE MEANS COMPLICATED DEBUGGING AND MAINTENANCE.

7. SEPARATE PROCEDURES SO THEY ARE EASY TO LOOK AT--USE A FEW BLANK LINES OR A LINE OF ASTERISKS.

8. CODE CONSISTENTLY AND INDENT CONSISTENTLY. FOR EXAMPLE:

```
IF CONDITION THEN
    A←B;
ELSE
    B←A;
IF CONDITION THEN
    DO IT;
       A←B;
       C←D;
    END IT;
ELSE
    DO WORK;
       E←F;
       G←H;
    END WORK;
```

IFS AND ELSES SHOULD BE IN THE SAME COLUMN. SO SHOULD DOS AND ENDS.

START TYPE DECLARATIONS IN THE SAME COLUMN THROUGHOUT THE PROGRAM.

```
DECLARE 01 A,
02 BCD BIT(4),
02 NEXT.GUY BIT(16),
02 E BIT(1),
02 F,
03 GHIJK BIT(10);
```

IS HARDER TO READ THEN

```
DECLARE
     01 A,
          02 BCD          BIT(4),
          02 NEXT.GUY     BIT(16),
          02 E            BIT(1),
          02 F,
             03 GHIJK     BIT(10);
```

ANOTHER SUGGESTION IS TO INDENT PROCEDURES AS THE LEXIC LEVEL INCREASES.

9. A MOTHERHOOD AND APPLE PIE HINT: USE DESCRIPTIVE NAMES. DON-T USE CRYPTIC ABBREVIATIONS OR "ALMOST CORRECT" SPELLINGS: RECRD FOR RECORD, ETC.

10. NAME ALL DO GROUPS.

INDEX

cut along dotted line

BURROUGHS CORPORATION
DATA PROCESSING PUBLICATIONS
REMARKS FORM

TITLE: B 1700 SYSTEMS – SYSTEM SOFTWARE DEVELOPMENT LANGUAGE (SDL) Reference Manual (BNF Version)

FORM: 1081346
DATE: 12-74

CHECK TYPE OF SUGGESTION:

☐ ADDITION ☐ DELETION ☐ REVISION ☐ ERROR

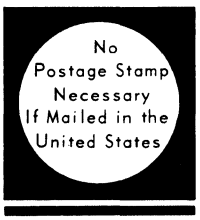
GENERAL COMMENTS AND/OR SUGGESTIONS FOR IMPROVEMENT OF PUBLICATION:

FROM: NAME _____ DATE _____
TITLE _____
COMPANY _____
ADDRESS _____
_____

cut along dotted line

STAPLE

FOLD DOWN SECOND FOLD DOWN



BUSINESS REPLY MAIL
First Class Permit No. 817, Detroit, Mich. 48232

Burroughs Corporation
Burroughs Place
Detroit, Michigan 48232

attn: Systems Documentation
Technical Information Organization, TIC-Central

FOLD UP FIRST FOLD UP

Burroughs

1081346 12-74 Printed in U. S. America