2210 0135

DISTRIBUTION LIST

B1800/B1700 SOFTWARE PRODUCT SPECIFICATIONS

Detroit

J. Cox - Prod. Mgmt.
F. Schoeman - International
H. R. Hayde - Interrational
D. Kosinski - Prod. Mgmt.
K. Stokes - International
J. Lambke - BMG
W. Varns - BMG
L. Atkins - BMG

Single Copy

D. Hill - TC, BM & SS
V. Morton - GPS, BM & SS
J. Shifman - CSG
C. Nash - International
J. G. Cleary - SSG
P. E. Pleming - Int'l F. E.
B. Dent - CSG
D. Dahm - Corp. Eng.

U.S. and Europe

K. Conry (Plymouth)
D. R. Bookwalter (Plymouth)
J. H. Pedersen (Plymouth)
J. Berta (Downingtown)
W. Minarcik (Paoli)
G. Smolnik (Paoli)
F. B. MacKenzie (Tredyffrin)
A. Kosla (McLean)
A. Lacaneta - F&SSG (McLean)
B. Bell (Wayne)
Mgr, WADC (Irvine)
R. Solt (Pasadena)
H. M. Townsend (Pasadena)
D. Prout - Pat. Atty. (Pasadena)
E. Sweaney (Mission Viejo)
E. D. Earnest (Mission Viejo)
J. J. Dowling (Westlake)

Single Copy

J. C. Allan (Glenrothes)
W. McKee (Cumbernauld)
I. J. Carradine (Cumbernauld)
Mgr, NPSGrp (Ruislip)
P. R. Evans (Middlesex)
J. Gerain (Pantin)
A. Isola (Gennevieliers)
P. Corril (Seneffe)
J. C. Kery (Liege)
R. Bouvier (Liege)
J. Cazanove (Villers)
B. Hammersley (Croydon)

Santa Barbara Plant

R. S. Bunker
J. Hale
R. Shobe
K. Meyers
R. Bauerle
A. van der Linden
E. Yardi
J. Carga
B. Ross-Smith
L. Thomas

Single/Multiple

E. Munsch	2
G. Hammond	2
J. Casey	1
K. King	2

Distribution current as of 9/21/77.

DOCUMENT TRANSMITTAL

PAGE 1 OF 1

CONTROL DOC. none

SYSTEM B1800/B7000

PL FILE NO. __________

PROJ. NO. __________ COORD. __________ CHKR. Hale TO RECORDS 9-21-77 DISTRIBUTION 9-21-77

DOC PFX	DOCUMENT NUMBER	REV	UNIT NUMBER	DL NO.	PL FILE NO.	ENG. RECORDS ONLY						REMARKS
						QTY REV PAGE	COPIES EACH PAGE	SEPIAS	MT REQ'D	CD DK REQ'D		
P.S.	2210 0135	D				84						

Burroughs Corporation

COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

2210 0135

B1800/~~B700~~B1700 BASIC S-LANGUAGE

PRODUCT SPECIFICATION

REV LTR	REVISION ISSUE DATE	APPROVED BY	REVISIONS
C	11/13/74	J Hale	Major Revision
D	9/20/77	J Hale	Major revision - MARK Level 7.0

84

TABLE OF CONTENTS

GENERAL DESCRIPTION

After a brief description of memory management for BASIC S-language programs, this product specification describes the operators generated by the B1800/B1700 BASIC Compiler. Operators are classified by type and indexed by their mnemonic abbreviations.

RELATED DOCUMENTATION

Name	Number
BASIC Compiler	P. S. 2212 5280
Burroughs Corporate Standard for BASIC	

```
BASE- - -!---------------------!- - - - - - - - - - -
         !                     !
         !                     !    VARIABLE SPACE
         !    STATIC MEMORY    !
         !                     !    NUMBER STACK
         !  (NOT OVERLAYABLE)  !         !
         !                     !         !
         !                     !         !
         !                     !    STRING STACK
         !---------------------!- - - - - - - - - MAY BE EXTENDED
         !                     !                  WITH "$" CARD
         !                     !                  AT COMPILE TIME
         !   VIRTUAL MEMORY    !    ARRAY SPACE
         !                     !    STRING SPACE
         !    (OVERLAYABLE)    !
         !                     !
         !                     !
LIMIT- - !---------------------!- - - - - - - - - EXTENDABLE AT BOJ.
         !                     !
         ! RUN STRUCTURE NUCLEUS !
         !                     !
         !---------------------!

         !---------------------!
         !                     !
         !        S-CODE       !
         !                     !
```

Figure 1.1 BASIC Object Program Memory Layout

VIRTUAL MEMORY

PAGE SIZE	PAGE CONTENTS	DESCRIPTION
64*48 Bits	Numeric array space	Numeric values.
64*48 Bits	String array space	String descriptors which point into the string space.
64*48 Bits	File parameter tables	Internal file tables.
512*8 Bits	Data statement elements	Data from data statements.
512*8 Bits	String constants	String constants from source.
512*8 Bits	String space	Space for generation of new strings.

Figure 1.3 Virtual Memory Layout Ordered According to the Data Dictionary Entries

RUN STRUCTURE NUCLEUS

The RS.NUCLEUS is located immediately above the data defined by the program's limit register. The RS.NUCLEUS contains information necessary for the MCP and the interpreter to execute the program. Consult current MCP documentation for further information.

The following 24 bit parameters are included in the RS.NUCLEUS. These parameters, in the order given, are placed in the scratchpad disk segment of the object program by the compiler anc are maintained in the RS.M.MACHINE area of the RS.NUCLEUS by the BASIC interpreter which updates them whenever control is transferred to the MCP.

Number stack address

Base-relative bit address of the start of the number stack.

Stack length remaining

One plus the length of the remaining stack space (in bits).

String stack address

Base-relative bit address of the start of the string stack.

FORMATS

The following is a description of data items and definition of terms which will be used throughout this manual. A list of terms used to describe items found in memory is given below.

- Static memory
    - Data table
        - Numeric variables
        - String variables
        - Numeric array descriptors
        - String array descriptors
    - Stack
        - Number stack
            - Numeric real value
            - Numeric integer value
            - Numeric base-relative address
            - Numeric virtual address
            - Numeric array pointer
            - Numeric array descriptor
            - Return control word
        - String stack
            - String non-self-relative descriptor
            - String self-relative descriptor
            - String base-relative address
            - String virtual address
            - String array pointer
            - String array descriptor
- Virtual memory
    - Numeric array elements
    - String array elements
    - File parameter tables
    - Data statement character strings and string constants
    - Free string space

DEFINITION OF TERMS

Base-relative address ::= Bit displacement from absolute bit address contained in base register

Virtual address ::= Page displacement

Page ::= Data dictionary entry number

Displacement ::= Word displacement from beginning of data

```
Internal string parameter address      bit (05)
                                 % Word displacement above the base-
                                 % relative address contained in the
                                 % internal string parameter register.

Internal string local address          bit (05)
                                 % Word displacement below the base-
                                 % relative address contained in the
                                 % internal string parameter register.
```

STACK FORMAT

The number stack is composed of two parts:

two 48 bit registers, (called A.REG and B.REG in the interpreter)

the number stack portion of static memory

If the A register is not empty, it is considered the top of the stack. If the A register is empty and the B register is not empty, the B register is considered the top of the stack. If they are both empty, the top of the stack is in static memory

The string stack is composed of two parts:

one 48 bit register (called STRING.TOP in the interpreter)

the string stack portion of static memory

The two stacks reside in static memory and grow toward each other. The length of these two stacks is a program parameter determined at runtime.

NUMBER STACK

The following items could be on the number stack.

01 NUMERIC VIRTUAL ADDRESS

```
   |--4---|-2-|-----18-----|-----24-----|
   --------------------------------------
   | 0011 |   |            |            |
   --------------------------------------
     TYPE         ADDRESS

   02 TYPE       BIT(04)   % 3
   02 FILLER     BIT(02)
   02 ADDRESS    BIT(18)   % VIRTUAL ADDRESS
      03 PAGE    BIT(12)   % DATA DICTIONARY ENTRY NUMBER
      03 DISP    BIT(06)   % WORD DISPLACEMENT FROM BEGINNING OF PAGE
   02 FILLER     BIT(24)
```

01 NUMERIC ARRAY POINTER

```
   |--4---|-2-|-----18-----|-----24-----|
   --------------------------------------
   | 0100 |   |            |            |
   --------------------------------------
     TYPE         ADDRESS

   02 TYPE       BIT(04)   % 4
   02 FILLER     BIT(02)   %
   02 ADDRESS    BIT(18)   % BASE-RELATIVE ADDRESS
   02 FILLER     BIT(24)
```

01 NUMERIC ARRAY DESCRIPTOR

```
   |--4---|-2-|---18---|-1-|---18---|---18---|---18---|---17---|
   -------------------------------------------------------------
   | 0101 |   |        |   |        |        |        |        |
   -------------------------------------------------------------
     TYPE        CCD     M     CRD    ADDRESS     MSV

   02 TYPE       BIT(04)   % 5
   02 FILLER     BIT(02)
   02 CCD        BIT(18)   % CURRENT COLUMN DIMENSION (FOR A(2,5)::=6)
   02 M          BIT(01)   % NOT A MATRIX SWITCH      (FOR A(2,5)::=0)
   02 CRD        BIT(18)   % CURRENT ROW DIMENSION    (FOR A(2,5)::=3)
   02 ADDRESS    BIT(18)   % VIRTUAL ADDRESS
      03 PAGE    BIT(12)   % DATA DICTIONARY ENTRY NUMBER
      03 DISP    BIT(06)   % WORD DISPLACEMENT FROM BEGINNING OF PAGE
   02 MSV        BIT(18)   % MAXIMUM SUBSCRIPT VALUE (FOR A(2,5)::=18)
```

STRING STACK

The following items could be on the string stack.

01 STRING NON-SELF-RELATIVE DESCRIPTOR

```
|--4---|--4---|-----16----|-3-|-------21------|
-----------------------------------------------
| 1000 |      |           |   |               |
-----------------------------------------------
  TYPE             STR.LEN          ADDRESS
```

```
02 TYPE       BIT(04)   % 8
02 FILLER     BIT(04)
02 STR.LEN    BIT(16)   % STRING LENGTH IN CHARACTERS
02 FILLER     BIT(03)
02 ADDRESS    BIT(21)   % VIRTUAL ADDRESS
  03 PAGE     BIT(12)   % DATA DICTIONARY ENTRY NUMBER
  03 DISP     BIT(09)   % CHARACTER DISPLACEMENT FROM BEGINNING
                        % OF PAGE.
```

01 STRING SELF-RELATIVE DESCRIPTOR

```
|--4---|--4---|---8---|------32-------|
---------------------------------------
| 1001 | 0001 |       |               |
---------------------------------------
  TYPE  TYPE.2  CHAR
```

```
02 TYPE         BIT(04)   % 9
02 TYPE.2       BIT(04)   % 1
02 CHAR         BIT(08)   % THE DESCRIBED SELF-RELATIVE STRING
02 FILLER       BIT(32)
```

01 STRING BASE-RELATIVE ADDRESS

```
|--4---|-2-|------18------|------24------|
------------------------------------------
| 1010 |   |              |              |
------------------------------------------
  TYPE        ADDRESS
```

```
02 TYPE         BIT(04)   % 10
02 FILLER       BIT(02)
02 ADDRESS      BIT(18)   % BASE-RELATIVE ADDRESS
02 FILLER       BIT(24)
```

S-INSTRUCTION FORMAT

Each BASIC S-instruction consists of a variable length S-operator followed by a variable number of arguments.

S-operators are 2, 5 or 8 bits in length with the most frequently used S-operators (on a static basis) being coded with the smallest number of bits. The length of the operator is determined by the operator itself as follows:

```
 --------        -----------         -------------
 !      !        !         !         !           !
 !  10  !        !  00000  !         !  01000000 !
 !  to  !        !   to    !         !    to     !
 !  11  !        !  00111  !         !  01111111 !
 !      !        !         !         !           !
 --------        -----------         -------------
  2 BIT             5 BIT                8 BIT
 OPERATOR         OPERATOR             OPERATOR
```

Each argument can consists of a variable number of bits. The format and interpretation of these arguments are described in detail with the description of the individual operators.

ADDRESS DECODING

ARRAY ADDRESS DECODING

Arrays may be referenced by address, value or by a pointer to an array descriptor. If the referencing is by address or value, the following is true:

IN: When referencing a global array, the argument to the opcode is a data table address. When referencing an external array the argument to the opcode is a stack address. In both cases, this cperand address is used to obtain an absolute bit address pointing to the array descriptor. The top of the number stack contains the subscript(s) of the element of the array being referenced.

OUT: The array descriptor and the subscript(s) to the array are used to obtain a virtual address. If the array is being referenced by address, this address is loaded to the top of the stack after type bits designating a virtual address are appended to the left. If the array is being referenced by value, the actual value of the array element (a numeric value of string descriptor) is loaded to the top of the appropriate stack. A fatal "INVALID SUBSCRIPT" error is generated if the subscript(s) are less than zero or greater than or equal to the array bounds value in the array descriptor.

If the reference is by a pointer to an array descriptor, the following is true:

IN: The argument to the opcode contains a data table address or a stack address. In either case, this operand address is used to obtain a base-relative bit address pointing to the array descriptor.

OUT: Type bits designating an array pointer (numeric or string) are appended to the left of the array descriptor address. This array pointer is loaded to the top of the appropriate stack.

INSTRUCTION SET

ARITHMETIC OPERATIONS

MNEMONIC	NAME	OPCODE	ARGUMENT
ADD	ADD	00011	
SUB	SUBTRACT	01000001	
MPY	MULTIPLY	01000010	
DIV	DIVIDE	01000011	
CSGN	COMPLEMENT SIGN	01000111	
PSGN	SET SIGN POSITIVE	01001000	
FLT	FLOAT	01001010	
FIX	FIX	01001001	
CONV	CONVERT TO DECIMAL	01010000	
RAT	ROUND AND TRUNCATE	01100101	

STACK OPERATIONS

MNEMONIC	NAME	OPCODE	ARGUMENT
EXCH	EXCHANGE	01001111	
PSHN	PUSH NUMERIC ZEROS	01010011	6 BIT OPERAND
PSHS	PUSH NULL STRINGS	01100010	6 BIT OPERAND

STORE OPERATIONS

MNEMONIC	NAME	OPCODE	ARGUMENT
STNX	STORE NUMERIC SPECIAL	01000101	1 BIT OPERAND
STB	STORE BITS	01001110	

LIT	LARGE LITERAL	01000000	40 BIT OPERAND
ZERO	LOAD ZERO	01101001	
ONE	LOAD ONE	01101010	
SLEN	LOAD STRING LENGTH	01011011	
ALAC	ARRAY LOAD ADDRESS FOR COMMUNICATE	01011100	9 BIT OPERAND

BRANCH OPERATIONS

MNEMONIC	NAME	OPCODE	ARGUMENT
FOR	FOR	01001011	THE ENTIRE "NEXT" INSTRUCTION IS THE OPCODE TO "FOR"
NEXT	NEXT	01000100	12 BIT OPERAND.1 12 BIT OPERAND.2 18 BIT OPERAND.3
ON	CASE	01000100	6 BIT OPERAND.1 18 BIT OPERAND.2 . . 18 BIT OPERAND.N
BUN	BRANCH UNCONDITIONAL	00110	1 BIT OPERAND.1 12 OR 18 BIT OPERAND.2
CALL	CALL	00000	2 BIT OPERAND.1 12 OR 18 BIT OPERAND.2
RTRN	RETURN	01100001	3 BIT OPERAND.1 5 BIT OPERAND.2 5 BIT OPERAND.3
BXXX	BRANCH CONDITIONAL	01101011	1 BIT OPERAND.1 12 OR 18 BIT OPERAND.2
CXXX	COMPARE NUMERIC AND BRANCH CONDITIONAL	00101	1 BIT OPERAND.1 12 OR 18 BIT OPERAND.2
COMM	COMMUNICATE	01010001	

S-OPERATIONS

ARITHMETIC OPERATIONS

ADD

SUBTRACT

MULTIPLY

DIVIDE

```
                                                         ***********************
                                                         * ADD, SUB, MPY, DIV *
                                                         ***********************

FORMAT:           ****************
         ADD      *       00011  *
         SUB      *    01000001  *
         MPY      *    01000010  *
         DIV      *    01000011  *
                  ****************
                       OPCODE

IN:      NUMBER STACK TOP        ::= ARGUMENT.1
         NUMBER STACK TOP-1      ::= ARGUMENT.2

         ARGUMENT.1              ::= NUMERIC REAL VALUE
         ARGUMENT.2              ::= NUMERIC REAL VALUE

OUT:     NUMBER STACK TOP        ::= RESULT

         RESULT                  ::= NUMERIC REAL VALUE
```

OPERATION:

Perform the indicated algebraic operation with ARGUMENT.1 and ARGUMENT.2 and replace them with the result. In subtraction, ARGUMENT.1 is subtracted from ARGUMENT.2. In division, ARGUMENT.1 is divided into ARGUMENT.2. If the exponent of the result overflows or underflows, or if during division ARGUMENT.1 is equal to zero, a runtime error is generated.

COMPLEMENT SIGN

```
********
* CSGN *
********
```

```
FORMAT:  *************
         * 01000111 *
         *************
            OPCODE
```

IN: NUMBER STACK TOP ::= NUMERIC REAL VALUE

OUT: NUMBER STACK TOP ::= NUMERIC REAL VALUE

OPERATION:

Complement the sign of the value on the top of the stack (i.e., change a positive value to a negative value and visa versa).

FLOAT

```
*******
* FLT *
*******
```

```
FORMAT:  *************
         * 01001010 *
         *************
            OPCODE
```

```
IN:      NUMBER STACK TOP ::= ARGUMENT

         ARGUMENT         ::= NUMERIC INTEGER VALUE

OUT:     NUMBER STACK TOP ::= RESULT

         RESULT           ::= NUMERIC REAL VALUE
```

OPERATION:

Replace the argument with its numeric real value equivalent. This is handled by shifting the integer Z places to the left; supplying Z trailing zero bits and supplying an exponent equal to 30-Z+256. Z equals the number of leading zero bits in the integer. If the exponent part of the argument is not equal to zero, then result = argument.

ROUND AND TRUNCATE

```
*******
* RAT *
*******
```

```
FORMAT:  ************
         * 01100101 *
         ************
            OPCODE

IN:      NUMBER STACK TOP ::= ARGUMENT

         ARGUMENT         ::= NUMERIC REAL VALUE

OUT      NUMBER STACK TOP ::= RESULT

         RESULT           ::= NUMERIC INTEGER VALUE
```

OPERATION:

Replace the argument with its integer value equivalent, rounced and truncated, using the following algorithm:

```
IF ARGUMENT < 0.5
  THEN IF ARGUMENT <= -0.5
     THEN RETURN A NUMERIC INTEGER VALUE OF MINUS INFINITY
     ELSE RETURN A NUMERIC INTEGER VALUE OF ZERO
IF ARGUMENT > 2**24-1
   THEN RETURN A NUMERIC INTEGER VALUE OF PLUS INFINITY
   ELSE RETURN A FIX(INT(STACK.TOP + 0.5))
```

PUSH NUMERIC ZEROS

```
********
* PSHN *
********
```

```
FORMAT:  |--- 8 ----|---- 6 ----|
         ************************
         * 01010011 *           *
         ************************
            OPCODE     OPERAND
```

```
OUT:     NUMBER STACK TOP   ::= NUMERIC VALUE OF ZERO

         NUMBER STACK TOP-1 ::= NUMERIC VALUE OF ZERO
                   .                        .
                   .                        .
```

OPERATION:

Push all occupied hardware number stack registers to memory. Then push as many numeric value zeros onto the top of the memory portion of the number stack as are indicated by the value of the operand.

STORE OPERATIONS

STORE BITS

```
                                                              *******
                                                              * STB *
                                                              *******

FORMAT:  !--- 8 ----!
         ************
         * 01001110 *
         ************
            OPCODE
```

```
IN:      NUMBER STACK TOP    ::= ADDRESS
         NUMBER STACK TOP-1  ::= OFFSET
         NUMBER STACK TOP-2  ::= LENGTH
         NUMBER STACK TOP-3  ::= DATA

         ADDRESS             ::=   NUMERIC VIRTUAL ADDRESS
                                 ! NUMERIC BASE-RELATIVE ADDRESS
         OFFSET              ::= NUMERIC REAL VALUE
         LENGTH              ::= NUMERIC REAL VALUE LEQ 48
         DATA                ::= BIT STRING

OUT:     ALL FOUR ITEMS ARE REMOVED FROM THE STACK
```

OPERATION:

Store a field of the bits found in DATA into the given ADDRESS. The OFFSET is integerized and added to the ADDRESS. This is used as the starting bit position of the requested bit field. The LENGTH is integerized and used as the length of the bit string. If LENGTH is less than the length of data, the rightmost bits of DATA are stored.

STORE NUMERIC SPECIAL

```
                                        ********
                                        * STNX *
                                        ********
```

```
FORMAT:  |--- 8 ----|--- 1 ---|
         **********************
         * 01000101 *         *
         **********************
            OPCODE    OPERAND
```

```
IN:      IF THE OPERAND = 1
           THEN NUMBER STACK TOP   ::= DESTINATION ADDRESS
                NUMBER STACK TOP-1 ::= SOURCE VALUE
         IF THE OPERAND = C
           THEN NUMBER STACK TOP   ::= SOURCE VALUE
                NUMBER STACK TOP-1 ::= DESTINATION ADDRESS

         DESTINATION ADDRESS       ::=   NUMERIC BASE-RELATIVE ACCRESS
                                       | NUMERIC VIRTUAL ADDRESS

         SOURCE VALUE              ::= NUMERIC REAL VALUE

CUT:     REMOVE NUMBER STACK TOP
         IF THE OPERAND = 1
           THEN REMOVE NUMBER STACK TOP-1
```

OPERATION:

Store the source value into the destination address.

LOAD OPERATIONS

ARRAY LOAD ADDRESS FOR COMMUNICATE

```
********
* ALAC *
********
```

```
FORMAT:  |--- 8 ----|--- 9 ---|
         **********************
         * 01011100 *         *
         **********************
            OPCODE    OPERAND
```

```
OPERAND ::= DATA TABLE ADDRESS POINTING TO THE ARRAY DESCRIPTOR

IN:      NUMBER STACK TOP   ::= SUBSCRIPT
         NUMBER STACK TOP-1 ::= [SUBSCRIPT]

OUT:     NUMBER STACK TOP   ::= BASE-RELATIVE ADDRESS
```

OPERATION:

Use the array descriptor pointed to by the operand and the subscript(s) to obtain a numeric virtual address of the array element being referenced. This numeric virtual address is used to make the appropriate page present. A base-relative address, pointing into the dynamic memory area, replaces the subscript on the number stack. A fatal "INVALID SUBSCRIPT" error is generated if the subscript(s) are less than zero or greater than or equal to the maximum subscript value in the array descriptor.

LOAD SIRING CONSIANI

```
*******
* LSC *
*******
```

```
FORMAT:  |--- 8 ----|---- 7 ----|---- 21 ----|
         *************************************
         * 01010101 *           *            *
         *************************************
            OPCODE    OPERAND.1   OPERAND.2

         OPERAND.1 ::= STRING LENGTH
         OPERAND.2 ::= STRING VIRTUAL ADDRESS
```

OPERATION:

Use OPERAND.1 and OPERAND.2 to create a string non-self-relative descriptor and load it to the top of the string stack.

LOAD EXTERNAL NUMERIC ARRAY

```
                                                              ********
                                                              * LENA *
                                                              ********

FORMAT:  |--- 8 ----|---- 3 ----|---- 9 ----|
         ************************************
         * 01011000 *           *           *
         ************************************
            OPCODE    OPERAND.1   OPERAND.2
```

OPERAND.1	OPERAND.2	TYPE OF LOAD
000	Data table address	Numeric array pointer
001	Data table address	Numeric array pointer
010	External numeric local address	Virtual address
011	External numeric parameter address	Virtual address
100	External numeric local address	Numeric real value
101	External numeric parameter address	Numeric real value
110	External numeric local address	Virtual address
111	External numeric parameter address	Virtual address

```
IN:      NUMBER STACK TOP    ::= SUBSCRIPT
         NUMBER STACK TOP-1  ::= [SUBSCRIPT]

OUT:     NUMBER STACK TOP    ::=    NUMERIC ARRAY POINTER
                                  | NUMERIC VIRTUAL ADDRESS
                                  | NUMERIC REAL VALUE
         THE SUBSCRIPT(S) ARE REMOVED FROM THE NUMBER STACK
```

OPERATION:

If bits 1 and 2 cf OPERAND.1 are not both 0 then convert OPERAND.2 to a base-relative address pointing to the numeric array descriptor. With this descriptor and the subscript(s) construct a numeric virtual address of the referenced array element. If OPERAND.1 specifies a load address then the numeric virtual address is loaded to the top of the number stack. If OPERAND.1 specifies a load value then the contents of the virtual address are loaded to the top of the number stack. A fatal "INVALID SUBSCRIPT" error is generated if the subscript(s) are less than zerc or greater than or equal to the maximum subscript value in the numeric array descriptor.

LOAD EXTERNAL STRING ARRAY

```
********
* LESA *
********
```

```
FORMAT:  !--- 8 ----!---- 3 ----!---- 9 ----!
         ************************************
         * 01011000 *           *           *
         ************************************
            OPCODE    OPERAND.1   OPERAND.2
```

OPERAND.1	OPERAND.2	TYPE OF LOAD
000	Data table address	String array pointer
001	Data table address	String array pointer
010	External string local address	String virtual address
011	External string parameter address	String virtual address
100	External string local address	String descriptor
101	External string parameter address	String descriptor
110	External string local address	String descriptor
111	External string parameter address	String descriptor

```
IN:      NUMBER STACK TOP    ::= SUBSCRIPT
         NUMBER STACK TOP-1  ::= [SUBSCRIPT]

OUT:     STRING STACK TOP    ::=   STRING ARRAY POINTER
                                 ! STRING VIRTUAL ADDRESS
                                 ! STRING DESCRIPTOR
         THE SUBSCRIPT(S) ARE REMOVED FROM THE NUMBER STACK
```

OPERATION:

If bits 1 and 2 of OPERAND.1 are not both 0 then convert OPERAND.2 to a base-relative address pointing to the string array descriptor. With this descriptor and the subscript(s) construct a string virtual address of the referenced array element. If OPERAND.1 specifies a load address then a string virtual address is loaded to the top of the string stack. A fatal "INVALID SUBSCRIPT" error is generated if the subscript(s) are less than zero or greater than or equal to the maximum subscript value in the numeric array descriptor.

If bits 1 and 2 of OPERAND.1 are both 0 then convert OPERAND.2 to a base-relative address. Append the type bits designating a string array pointer to the left of this base-relative address and load it onto the top of the string stack.

LOAD GLOBAL STRING ARRAY

```
                                                  ********
                                                  * LGSA *
                                                  ********
FORMAT:  !--- 8 ----!---- 3 ----!---- 9 ----!
         ************************************
         * 01011000 *           *           *
         ************************************
            OPCODE    OPERAND.1   OPERAND.2
```

OPERAND.1	OPERAND.2	TYPE OF LOAD
000	Data table address	String array pointer
001	Data table address	String array pointer
010	Data table address	String virtual address
011	Data table address	String virtual address
100	Data table address	String descriptor
101	Data table address	String descriptor
110	Data table address	String descriptor
111	Data table address	String descriptor

```
IN:      NUMBER STACK TOP   ::= SUBSCRIPT
         NUMBER STACK TOP-1 ::= [SUBSCRIPT]

OUT:     STRING STACK TOP   ::=   STRING ARRAY POINTER
                                ! STRING VIRTUAL ADDRESS
                                ! STRING DESCRIPTOR
         THE SUBSCRIPT(S) ARE REMOVED FROM THE NUMBER STACK
```

OPERATION:

If bits 1 and 2 of OPERAND.1 are not both 0 then convert OPERAND.2 to a base-relative address pointing to the string array descriptor. With this descriptor and the subscript(s) construct a string virtual address of the referenced array element. If OPERAND.1 specifies a load address then a string virtual address is loaded to the top of the string stack. If OPERAND.1 specifies a load value then the contents of the virtual address are loaded to the top of the string stack. A fatal "INVALID SUBSCRIPT" error is generated if the subscript(s) are less than zero or greater than or equal to the maximum subscript value in the numeric array descriptor.

If bits 1 and 2 of OPERAND.1 are both 0 then convert OPERAND.2 to a base-relative address. Append the type bits designating a string array pointer to the left of this base-relative address and load it onto the top of the string stack.

LARGE LITERAL

```
*******
* LIT *
*******
```

```
FORMAT:  |--- 8 ----|--- 40 ---|
         ***********************
         * 01000000 *          *
         ***********************
            OPCODE    OPERAND
```

OPERAND ::= 40 BIT LITERAL (1 SIGN; 9 EXPONENT; 30 FRACTION BITS)

CUT: NUMBER STACK TOP ::= NUMERIC REAL VALUE

OPERATION:

Use the operand to create a numeric real value and load it to the top of the number stack.

LOAD NUMERIC SCALAR

```
*******
* LNS *
*******
```

```
FORMAT:  |-- 2 --|--- 3 ---|--- 5 OR 9 ---|
         **********************************
         *  10   *         *              *
         **********************************
          CPCODE  OPERAND.1    OPERAND.2
```

OPERAND.1	OPERAND.2	TYPE OF LCAD
000	9-bit cata table address	Address
001	5-bit cata table address	Address
010	5-bit internal numeric local address	Address
011	5-bit internal numeric parameter address	Address
100	9-bit data table address	Value
101	5-bit data table address	Value
110	5-bit internal numeric local address	Value
111	5-bit internal numeric parameter address	Value

```
OUT:     NUMBER STACK TOP ::=   DATA TABLE ADDRESS
                              | INTERNAL NUMERIC LOCAL ADDRESS
                              | INTERNAL NUMERIC PARAMETER ACCRESS
                              | NUMERIC REAL VALUE
```

OPERATION:

Load an address or value to the top of the number stack.

LOAD SEQUENCE NUMBER

```
*******
* LSN *
*******
```

```
FORMAT:  *************
         * 01100111 *
         *************
            OPCODE
```

OPERATION:

Load the last stored sequence number to the top of the number stack as a numeric integer value.

LOAD STRING SCALAR

```
*******
* LSS *
*******
```

```
FORMAT:  |-- 5 --|---- 4 ----|--- 5 OR 9 ---|
         ************************************
         * 0011  *           *              *
         ************************************
          OPCODE   OPERAND.1    OPERAND.2
```

OPERAND.1	OPERAND.2	TYPE OF LOAD
0000	Data table address	Base-relative address
0001	Data table address	Base-relative address
0010	Internal string parameter address	Base-relative address
0011	Internal string local address	Base-relative address
0100	External string parameter address	Base-relative address
0101	External string local address	Base-relative address
1000	Data table address	String descriptor
1001	Data table address	String descriptor
1010	Internal string parameter address	String descriptor
1011	Internal string local address	String descriptor
1100	External string parameter address	String descriptor
1101	External string local address	String descriptor

```
OUT:     STRING STACK TOP ::=   BASE-RELATIVE ADDRESS
                               | STRING DESCRIPTOR
```

OPERATION:

Load the base-relative address of a string descriptor or the string descriptor to the top of the string stack.

LOAD STRING LENGTH

```
                                              ********
                                              * SLEN *
                                              ********

FORMAT:  1--- 8 ----1
         ************
         * 01011011 *
         ************
            OPCODE
```

```
IN:      STRING STACK TOP ::= STRING DESCRIPTOR

OUT:     NUMBER STACK TOP ::= NUMERIC REAL VALUE
         THE STRING DESCRIPTOR IS REMOVED FROM THE TOP OF THE
         STRING STACK
```

OPERATION:

Extract the string length from the string descriptor on the top of the string stack and load it as a positive numeric real value onto the top of the number stack.

LOAD ZERO

```
********
* ZERO *
********
```

```
FORMAT:  |--- 8 ----|
         ************
         * 01101001 *
         ************
            OPCODE
```

OPERATION:

Load a numeric real value of zero to the top of the number stack.

BRANCH CONDITIONAL

```
********
* BXXX *
********
```

```
FORMAT:  |--- 8 ----|---- 3 ----|---- 1 ----|-- 12 OR 18 --|
         ***************************************************
         * 01101011 *           *           *              *
         ***************************************************
            OPCODE     OPERAND.1   OPERAND.2     OPERAND.3

         OPERAND.1 ::= BRANCH.MASK
         OPERAND.2 ::= DISPLACEMENT.FLAG
         OPERAND.3 ::= BRANCH.DISPLACEMENT
```

CPERATION:

Compare the BRANCH.MASK to the current setting of the relational toggles. If any "ONE" bit in the BRANCH.MASK is matched by a corresponding "ONE" bit in the relational toggles, select the next instruction from the address determined by adding the unsigned binary integer BRANCH.DISPLACEMENT to the base address of the current ccde segment; otherwise terminate the instruction.

The following table gives the binary values of the BRANCH.MASK required to implement the given branch type.

BGTR = 001 = GREATER THAN
BLSS = 010 = LESS THAN
BNEQ = 011 = NOT EQUAL
BEQL = 10C = EQUAL
BGEQ = 101 = GREATER THAN OR EQUAL
BLEQ = 11C = LESS THAN OR EQUAL

If DISPLACEMENT.FLAG = 1, then the length of the BRANCH.DISPLACEMENT value is 18 bits; otherwise, it is 12 bits long.

parameter registers (ENPR and ESPR) point to the parameters of the routine where the function is defined.

When executing the return statement, the fields in the RCW containing the previous contents of the parameter registers are stored in the parameter registers. In this way, the operating environment is restored to its condition prior to the last call.

The format of the return control word is as follows:

```
|-- 4 --|-- 2 --|--- 25 ---|-- 17 --|-- 17 --|-- 17 --|-- 14 --|
****************************************************************
* 1111  * RCW.T * RTN.ADDR *  P.RCR *  P.NPR *  P.SPR *  0...0 *
****************************************************************
```

```
RCW.T    ::= 00  INTERNAL SUBROUTINE
             01  INTERNAL FUNCTION
             10  EXTERNAL SUBROUTINE
             11  EXTERNAL FUNCTION

RTN.ADDR ::= 7 BIT SEGMENT DICTIONARY ENTRY NUMBER
             18 BIT DISPLACEMENT FROM BEGINNING OF SEGMENT

P.RCR    ::= THE CONTENTS OF THE RETURN CONTROL REGISTER ASSOCIATED
             WITH THE PREVIOUS CALL

P.NPR    ::= IF THE CALL IS TO AN INTERNAL SUBROUTINE THEN P.NPR IS
                 NOT USED.
             IF THE CALL IS TO AN INTERNAL FUNCTION THEN P.NPR
                 CONTAINS THE CONTENTS OF THE INTERNAL NUMERIC
                 PARAMETER REGISTER (ENPR) ASSOCIATED WITH THE
                 PREVIOUS CALL.
             IF THE CALL IS TO AN EXTERNAL SUBROUTINE OR FUNCTION
                 THEN P.NPR CONTAINS THE CONTENTS OF THE EXTERNAL
                 STRING PARAMETER REGISTER (ESPR) ASSOCIATED WITH
                 THE PREVIOUS CALL.

P.SPR    ::= IF THE CALL IS TO AN INTERNAL SUBROUTINE THEN P.SPR IS
                 NOT USED.
             IF THE CALL IS TO AN INTERNAL FUNCTION THEN P.SPR
                 CONTAINS THE CONTENTS OF THE INTERNAL STRING
                 PARAMETER REGISTER (INSPR) ASSOCIATED WITH THE
                 PREVIOUS CALL.
             IF THE CALL IS TO AN EXTERNAL SUBROUTINE OR FUNCTION
                 THEN P.SPR CONTAINS THE CONTENTS OF THE EXTERNAL
                 STRING PARAMETER REGISTER (ESPR) ASSOCIATED WITH
                 THE PREVIOUS CALL.
```

COMPARE TWO STRINGS

```
                                                  ********
                                                  * CMPS *
                                                  ********
```

```
FORMAT:  ************
         * 01011111 *
         ************
            OPCODE
```

```
IN:      STRING STACK TOP   ::= STRING DESCRIPTOR
         STRING STACK TOP-1 ::= STRING DESCRIPTOR

OUT:     BOTH STRING DESCRIPTORS ARE REMOVED FROM THE TOP OF THE
         STRING STACK
```

OPERATION:

Compare two strings, as described by the two string descriptors on the top of the string stack, to each other until they compare unequal or until one of the strings is finished.

If they compare unequal, then the relational toggles are set to indicate whether the second string is less than or greater than the first string.

If they compare equal until one of the strings is finished, then the relational toggles are set to "LESS THAN" if the second string is shorter; to "GREATER THAN" if the second string is longer; or to "EQUAL" if the two strings have equal length.

COMMUNICATE

```
********
* COMM *
********
```

```
FORMAT:  *************
         * 01010001 *
         *************
            OPCODE
```

OPERATION:

Load the 48 bit RS.COMMUNICATE.MSG.PRT area of the RS.NUCLEUS with an SDL-type data descriptor indicating a length of 120 bits and containing the absolute address of the data table. This first 120 bits of the data-table is reserved for use as a communicate message area.

The format of this SDL-type data descriptor is as follows:

```
|---- 8 ----|----- 16 -----|------- 24 -------|
***********************************************
* TYPE      * LENGTH       * ABSOLUTE ADDRESS *
* 0100 0000 * 00..01111000 * OF DATA-TABLE    *
***********************************************
```

Store the status of the M-machine in the appropriate parts of this program's RS.NUCLEUS.

Instate the program (GISMO) whose RS.NUCLEUS address is given in the RS.COMMUNICATE.LR field of this program's RS.NUCLEUS.

FOR

```
                                                                    *******
                                                                    * FOR *
                                                                    *******

FORMAT:  I--- 8 ----I---- 8 ----I--- 12 ---I--- 12 ---I--- 18 ---I
         *********************************************************
         * 01001011 * 01001100  *          *          *          *
         *********************************************************
           OPCODE    OPCODE(NEXT) OPERAND.1  OPERAND.2  OPERAND.3

         OPERAND.1                ::= LOOP INDEX ADDRESS
         OPERAND.2                ::= LOOP PARAMETER ADDRESS
         OPERAND.3                ::= LOOP ENDING ADDRESS

         LOOP INDEX ADDRESS       ::= BASE-RELATIVE ADDRESS
         LOOP PARAMETERS ADDRESS  ::= BASE-RELATIVE ADDRESS
         LOOP ENDING ADDRESS      ::= CODE LOCATION

IN:      NUMBER STACK TOP         ::= STEP.SIZE
         NUMBER STACK TOP-1       ::= ENDING INDEX VALUE
         NUMBER STACK TOP-2       ::= BEGINNING INDEX VALUE

OUT:     ALL THREE ITEMS ARE REMOVED FROM THE TOP OF THE NUMBER
         STACK.

OPERATION:
```

Store the STEP.SIZE and then the ENDING.INDEX.VALUE into the data table at the location specified by the LOOP.PARAMETERS.ADDRESS. Store the BEGINNING.INDEX.VALUE in the data table at the location specified by the LOOP.INDEX.ADDRESS. Compare the BEGINNING.INDEX.VALUE with the ENDING.INDEX.VALUE.

The loop is satisfied if, for a positive STEP.SIZE, the BEGINNING.INDEX.VALUE is greater than the ENDING.INDEX.VALUE; or for a negative STEP.SIZE, the BEGINNING.INDEX.VALUE is less than the ENDING.INDEX.VALUE.

If the loop is not satisfied, the instruction following the "FOR" instruction is then selected. If the loop is satisfied, the next instruction is selected from the code location specified by the LOOP.ENDING.ADDRESS.

CASE

```
                                                                  ******
                                                                  * ON *
                                                                  ******
```

```
FORMAT:  |--- 8 ----|--- 6 ---|--- 18 ---|      |--- 18 ---|
         **********************************     ************
         * 01000100 *         *          *.....*          *
         **********************************     ************
            OPCODE   OPERAND.1  OPERAND.2         OPERAND.N

         OPERAND.1               ::= NUMBER.OF.ADDRESSES
         OPERAND.2               ::= BRANCH.DISPLACEMENT.ONE
         .
         .
         OPERAND.N               ::= BRANCH.DISPLACEMENT.N

         NUMBER.OF.ADDRESSES     ::= 6 BIT BINARY INTEGER
         BRANCH.DISPLACEMENT.X   ::= 18 BIT BINARY INTEGER

IN:      NUMBER STACK TOP        ::= NUMERIC REAL VALUE

OUT:     THE NUMERIC REAL VALUE IS REMOVED FROM THE NUMBER STACK.
```

OPERATION:

Select the next instruction from the location determined by adding the branch displacement to the base address of the current code segment.

The particular BRANCH.DISPLACEMENT to be used is determined by the rounded and truncated value of the numeric real value found on the top of the number stack, i.e., STACK TOP ::= TRC(STACK.TOP + 0.5).

If this value is less than one or greater than OPERAND.1, a runtime error is generated.

EXIRACI A SIRING

```
********
* EXTS *
********
```

```
FORMAT:  *************
         * 01011001 *
         *************
            OPCODE

IN:      STRING STACK TOP               ::= STRING NON-SELF-RELATIVE
                                            DESCRIPTOR
         NUMBER STACK TOP               ::= BEGINNING CHARACTER POSITION
         NUMBER STACK TOP-1             ::= ENDING CHARACTER POSITICN

         BEGINNING CHARACTER POSITION ::= NUMERIC REAL VALUE
         ENDING CHARACTER POSITION    ::= NUMERIC REAL VALUE

CUT:     STRING STACK TOP               ::= STRING NON-SELF-RELATIVE
                                            DESCRIPTOR
         THE NUMBER STACK IS RETURNED TO IIS CONDITICN PRICR TC
         THIS S-OP.
```

OPERATION:

Convert the string descriptor on the top of the string stack to a sub-string of itself, as indicated by the beginning and ending character position values found on the top of the number stack.

If either the beginning character position value initially incicates a character position preceding the first character in the input string, or the ending character position value initially incicates a character position following the last character in the input string, it is first adjusted to point just inside the string. If, after any necessary adjustments, they overlap, then the top of the string stack is converted to a null string descriptor.

MAKE A STRING DESCRIPTOR

```
********
* MSDS *
********
```

```
FORMAT:  *************
         * 0101101C *
         *************
            OPCODE
```

```
IN:      NUMBER STACK TOP    ::= ARRAY ADDRESS
         NUMBER STACK TOP-1  ::= OFFSET
         NUMBER STACK TOP-2  ::= STRING LENGTH

         ARRAY ADDRESS       ::= NUMERIC VIRTUAL ADDRESS
         OFFSET              ::= NUMERIC REAL VALUE
         STRING LENGTH       ::=  NUMERIC REAL VALUE
```

OPERATION:

This S-op was designed to be used when compiling an intrinsic. One of its uses is to convert the buffer area (an array) to a string descriptor.

Convert the array address, the offset and the string length to a string descriptor which is left on the top of the string stack. Remove the three items from the top of the number stack.

MOVE STRING CHARACTERS

```
********
* MVSC *
********
```

```
FORMAT:  *************
         * 01011101 *
         *************
            OPCODE

IN:      STRING STACK TOP    ::= DESTINATION
         STRING STACK TOP-1  ::= SOURCE

         NUMBER STACK TOP    ::= OFFSET

         DESTINATION         ::= STRING NON-SELF-RELATIVE DESCRIPTOR
         SOURCE              ::= STRING NON-SELF-RELATIVE DESCRIPTOR
         OFFSET              ::= NUMERIC REAL VALUE WHICH IS THE
                                 CHARACTER OFFSET INTO THE DESTINATION
                                 FIELD WHERE THE FIRST SOURCE CHARACTER
                                 IS TO BE MOVED.

OUT:     THE TWO DESCRIPTORS ARE REMOVED FROM THE TOP OF THE
         STRING STACK.  THE NUMBER STACK CONTAINS THE UPDATED
         OFFSET, EQUAL TO THE INPUT OFFSET PLUS THE NUMBER OF
         CHARACTERS MOVED.

OPERATION:
```

Move the characters from the source string to the area of the destination string.

SCAN STRING

```
********
* SCAN *
********
```

```
FORMAT:  |--- 8 ----|--- 2 ---|
         **********************
         * 01100110 *         *
         **********************
           OPCODE     OPERAND
```

OPERATION:

The source statement which causes creation of this S-op looks like this:

```
SCAN (A$, I, FWD, EQ, D$, R$, J)
A$  ::= STRING TO BE SCANNED
I   ::= CHARACTER POSITION AT WHICH TO BEGIN
FWD ::= FORWARD OR REVERSE
EQ  ::= EQL OR NEQ
D$  ::= CHARACTER FOR WHICH TO SCAN
R$  ::= IF D$ FOUND THEN D$ ELSE ""
J   ::= IF D$ FOUND THEN CHARACTER POSITION OF D$ ELSE 0
```

The stacks look like this prior to execution of the S-op:

```
STRING STACK                  NUMBER STACK

TOP   ::= A$                  TOP ::= I
TOP-1 ::= D$
```

After execution:

```
TOP ::= D$ OR ""              TOP ::= CHAR POSITION OF D$
```

```
OPERAND (FIRST BIT)  ::= 1 FORWARD; 0 REVERSE
       (SECOND BIT) ::= 1 EQ; 0 NEQ
```

TRACE AND SAVE SEQUENCE NUMBER

```
********
* TSSN *
********
```

```
FORMAT:  1--- 8 ----1--- 17 ---1
         ***********************
         * 01101111 *          *
         ***********************
            OPCODE    OPERAND
```

OPERATION:

Save the sequence number stored in the 17 bit operand in static memory word 27. Set up for a call to the ERR intrinsic with a parameter of 52 which tells ERR to print the sequence number and continue.

INDEX