#2201 6729

Burroughs Corporation

BUSINESS MACHINES GROUP
SMALL SYSTEMS PLANT

B1700 COBOL S-LANGUAGE

PRODUCT SPECIFICATION

REVISIONS

REV LTR	REVISION ISSUE DATE	PAGES REVISED ADDED DELETED OR CHANGE OF CLASSIFICATION	PREPARED BY	APPROVED BY
E	2/3/76	Major revision. Name changed to B1700 COBOL S-LANGUAGE. Translated to upper case and lower case.	K.M.K.	[illegible]

GOL 1955 [illegible]

Burroughs Corporation

SYSTEMS M & E GROUP
SANTA BARBARA PLANT

#2201 6729

COBOL S-LANGUAGE

PRODUCT SPECIFICATION

REVISIONS

REV LTR	REVISION ISSUE DATE	PAGES REVISED ADDED DELETED OR CHANGE OF CLASSIFICATION	PREPARED BY	APPROVED BY
A	12-XX-70	Transferred from P.S. #1912 3553 Sec. 1.1 Changed some names of program parameters 1.2 Moved reinstate info to above limit register 2.1.3 Changed L TYPE BIT assignment 2.1.4 Changed method of Address calculation 2.2.1 Added Segment # 2.2.4 Changed DATA TYPE BIT Assignment 2.2.8 Changed Indexing BIT Assignment 2.2.10 Added ASCII flag description 3.0 Deleted CNZ (Compare for N Zero) instruction 3.0 through 3.4.6 Added ASCII code sensitivity changes where necessary (see section 2.2.10) Deleted CONVERT SIGN Instruction. 3.1.5 Restricted MUL result field to 4-bit format 3.1.5 Required COPX2 data length be equal to the sum of the lengths of the operands 3.1.6 Restricted DIV result field to 4-bit format. 3.1.6 Required COPX1 data length be equal to the difference of the lengths of the operands. 3.2.11 Added MVT (Move Translate) instruction 3.2.12 Changed order of OPND2 and COPX2. 3.2.13 Deleted SKIP Forward Destination operator. 3.4 and 3.4.6 Reversed BRANCH Taken-Not Taken Condition Changed Relational Condition Bit Assignments 3.4.3 Generalized ZRO to full relational test. 3.4.4 Generalized SPA to full relational test.		
B	2-26-71	Sec. 1 Changed typical program memory layout. 1.1 Major change of program parameters. 2.0 Changed OP from 8 to 9 bits. 2.1.5 Added In-Line-COP Information 2.2 Deleted Edit Mask Address. Added Table Bound. 2.2.2 Changed BASE REGISTER to Base of Data Segment.		

#2201 6729

Burroughs Corporation

SYSTEMS M & E GROUP
SANTA BARBARA PLANT

COBOL S-LANGUAGE

PRODUCT SPECIFICATION

REVISIONS

REV LTR	REVISION ISSUE DATE	PAGES REVISED ADDED DELETED OR CHANGE OF CLASSIFICATION	PREPARED BY	APPROVED BY
B		Cont. 2.2.6 & 2.2.7 & 2.2.8 Changed method of indicating number of subscripts/indexes. 2.2.7& 2.2.8 Added out of range-condition on subscripting and indexing 2.2.9 Added description of Table Bound. Deleted Edit Mask Address description. 3.0 Added DADDR in Edit. Moved N variant and changed BADDR to BDISP in GTD. Added BOF, OFY, CRPT, COMM, FCMP, CNV and LDS operators. 3.1.6 Added: Division by zero results in overflow toggle being set. Dividend not quotient field must be 4-bit. 3.2.1& .3 & .11 Added statement on overlap of fields. 3.10 SMVN- COPX1 changed to OPND1. 3.2.13 Restricted destination field of Edit to 8-bit format 3.2.13.1 Added DADDR to edit instruction. 3.2.13.2 Corrected bit type from 10 to 01. 3.2.13.3 S=0 Changed to S=1 throughout added S=0, T=8 and S=1, T=9 to Insert on Minus. 3.3 Major change to branch types. 3.3.2 & .3 Added BOF and OFL. 3.3.2.8 GTD-Moved N variant. 3.3.4 Major change in branch types. 3.4.7 Added CRPT. 3.5.1, 3.5.2, 3.5.3, 3.5.4 Added COMM, CNV, LDS		
C	5-17-72	Sec 1 Deleted address store and added alter table to table. 1.1 Changed BDISPB to BDISP1 2.2 Changed min size of seq # container from 1 to 0. Specified max size of LENB as 13 and 14 for 8 & 4 bit data resp. 2.2.7 Specified subscript value of ≤ 0 results in error comm. Added overflow is ignored if sum of subscript values exceed 24 bits. 2.2.8 Added sign position to index register & detection when it is negative.	WFK	[illegible] 6/29/72 WFK 6-29-72 [illegible] 6/30/72

#2201 6729

Burroughs Corporation

SYSTEMS M & E GROUP
SANTA BARBARA PLANT

COBOL S-LANGUAGE

PRODUCT SPECIFICATION

REVISIONS

REV LTR	REVISION ISSUE DATE	PAGES REVISED ADDED DELETED OR CHANGE OF CLASSIFICATION	PREPARED BY	APPROVED BY
C	Cont'd.	Sec. 3.0 Added OP code and changed some nmemonics 3.1 Changed no restriction as to type to restriction as to type are specified under description of various operations. 3.1.6 Changed dividend from OPND to COPX operand 3.2.10 Scale factor length must be $\leq$ dest. field length if V=0 and $\leq$ source field length if V=1. 3.2.11 Added COPX1 to describe translation table. 3.2.12 Examine: deleted 3.2.13 Deleted S=0, T=8, and S=1, T=9 entries 3.2.14 & 3.2.15 Added MCF & MCE operators 3.3 Deleted self-relative branches. Changed format of BADDR 3.3.4 PERF: Changed format of stack entry 3.3.8 GOTD: Changed name of BDISP to DBADDR and changed format 3.4 Specified order of operands 3.4.3 CMPX: Changed binary compare to algebraic. Specified order of operands 3.4.4 CMPS: Specified order of operands. 3.4.5 CMPC: Restricted to UA. 3.4.7 Deleted req't that OPND1 size be larger than 2. 3.5.1 Address stored as absolute instead of relative. 3.5.2 LDCR name changed from FCMP 3.5.3 CNV: Changed operand from OPND to COPX. 3.5.4 MAKP name changed from LDSN. DSEGN argument changed to COPX1.		
D	11-28-72	Sec 2.2.10, 3.0, 3.2.5, 3.2.6, 3.2.7, 3.2.8, 3.4.6: MMVA, MMVS, MMVN, MMVZ and CMPM operators deleted. 3.2.9 Thru 3.2.15 renumbered to 3.2.5 thru 3.2.11. 3.2.8 Added Examine operator. 3.2.11 Corrected Tally register to COPX3. 3.4.5 Specified sign position is ignored for Signal 8-bit format in the numeric test. 3.4.7 renumbered to 3.4.6	WFK	12/7/72 12-8-72 12/12/72 12/26/72

TABLE OF CONTENTS:

GENERAL

B1700 COBOL S-Language provides the virtual machine interface between the code generated by the COBOL Compiler and the COBOL Interpreter. This specification describes the format of COBOL S-instructions and then explains each operator as a member of one of the following classes:

- ARITHMETIC
- DATA MOVEMENT
- BRANCHING
- CONDITIONAL BRANCHING
- MISCELLANEOUS

RELATED PUBLICATIONS

TITLE	NUMBER
B1700 Systems COBOL Reference Manual	1057197
B1700 COBOL Compiler	(P.S.) 2212 5314
B1700 COBOL Compiler Logic	(P.S.) 2212 5397

S-LANGUAGE PROGRAMS

All COBOL S-language programs have associated with them, a base register and a limit register. The area between the base and the limit is to be used as data space only. All program code, organized in segment form, is stored at any available location in memory, according to the memory management algorithms used by the B1700 operating system.

The data space includes a non-overlayable area which contains the COP table and various other parameters such as Edit Masks and Record Areas.

Various parameters, necessary for the running of the S-Language object code and maintained by the MCP, are stored beyond the Limit Register in the Run Structure Nucleus (RSN).

A typical COBOL program layout in memory is as follows:

```
BASE REGISTER--->--------------------------------
              * !     EDIT TABLE (8 CHRS)       !
             *  !-------------------------------!<--1ST ENTRY (COPX=1)
            *   !         COP TABLE             !
           *    !-------------------------------!<--DATA SEGMENT ZERO
          *     !      SPECIAL REGISTERS        ! *        BASE
         *      !-------------------------------!  *
        *       !       DATANAME MONITOR        !   *
       *        !          SYMBOLS              !    *
       *        !-------------------------------!     *
  STATIC MEMORY !         EDIT MASKS            !      *
       *        !-------------------------------!       *
       *        !   NONOVERLAYABLE USER'S       !   DATA SEGMENT ZERO
        *       !          DATA AREA            !       *
         *      !-------------------------------!      *
          *     ! TRASH(INTERMEDIATE RESULTS)   !     *
           *    !-------------------------------!    *
            *   !    ALTER TABLE (IF ANY)       !   *
             *  !-------------------------------!<-*STACK BASE
              * !           STACK               ! *
               *!-------------------------------!<--STACK LIMIT
              * !                               !
DYNAMIC MEMORY<* !          OVERLAYABLE          !
              * !        USER DATA AREA         !
LIMIT REGISTER<-*!-------------------------------!<--EXTENDABLE LIMIT REG.
                !       REINSTATE INFO          !   (BOJ ATTRIBUTE)
                !       & RUN STRUCTURE         !
                !-------------------------------!
                !       DATA DICTIONARY         !
                !-------------------------------!
                !           S-CODE              !
                !                               !
```

FIGURE 1-1: COBOL PROGRAM LAYOUT

ADR	NAME	PIC
0	SW1	9 CMP
:	:	:
7	SW8	9 CMP
8	TALLY	9(5) CMP
13	DATE (JULIAN) (YYDDD)	9(5) CMP
18	TIME (HHMMSST)	9(7) CMP
25	TODAYS-DATE (MMDDYY)	9(6) CMP
31	TODAYS-NAME	X(9)

FIGURE 1-2: SPECIAL REGISTERS

PROGRAM PARAMETERS

The parameters pertaining to a particular program are listed below. The number of bits used to contain the parameter appears in parentheses following the parameter name.

BDISPB1 (5)	BRANCH DISPLACEMENT CONTAINER SIZE + 1
DSEGZ (24)	BASE OF DATA SEGMENT ZERO
STACK-POINTER (24)	BASE ADDRESS OF STACK
STACK-SIZE (5)	SIZE OF THE STACK
COP-BASE (24)	BASE ADDRESS OF COP TABLE
COPB (12)	COP ENTRY CONTAINER SIZE
SEGB (5)	DATA SEGMENT NUMBER CONTAINER SIZE
DISPB (5)	DATA DISPLACEMENT CONTAINER SIZE
LENB (5)	DATA LENGTH CONTAINER SIZE
COPXB (5)	COP INDEX CONTAINER SIZE

CONTAINER SIZE

Container size is a field size (in number of bits) necessary to contain the maximum value required for that field. For example: A container size of five bits allows a field value to house 32 bit addresses (0-31).

S-INSTRUCTION FORMAT

Each COBOL S-Instruction consists of an S-operator followed by arguments consisting of a variable number of bits. The format and interpretation of these arguments is specified by the S-operator and is described in detail by the specification of the individual operators. An example of one such instruction format is illustrated below.

```
-------------------------------------
 OP1  OP2  OPND         COPX
 (3)  (6)  (VARIABLE)   (COPXB)
-------------+-------------+--------
             I             I
             I             I--INDEX INTO COP TABLE
             I
             I--INDEX INTO COP TABLE OR LITERAL
```

S-OPERATORS

The most frequently used S-operators are encoded in a three bit S-operator denoted as OP1. If OP1 is equal to seven, the operator is encoded in the next six bits denoted as OP2. If OP1 is less than seven, OP2 is omitted.

OPND

An operand is normally referenced indirectly through a table containing the attributes of the operand. An argument which references an operand in this manner is denoted as COPX. An operand is either contained in the instruction as a literal or is referenced indirectly through the table. An argument of this type is denoted as OPND. The first bit of OPND is denoted as LITFLG and is used to indicate a literal string or COPX as follows:

```
              +---+---------------------------+
        !-->! 1 ! LITERAL STRING            !
        !     +---+---------------------------+
LITFLG---
        !     +---+---------------------------+
        !-->! 0 ! COPX                      !
              +---+---------------------------+
```

LITERAL STRING

When LITFLG specifies a literal, the literal string, which includes the literal type (LTYPE), the literal length (LLGTH), and the literal (LSYMB) itself in that order, is included in the code stream immediately following the LITFLG. The format is as follows:

LTYPE	LLGTH1	LLGTH2	LSYMB
(2)	(3)	(8)	(variable)

Note: LLGTH2 present if LLGTH1 equal zero

LTYPE

00=Unsigned 4-BIT
01=Unsigned 8-BIT
10=Signed 4-BIT (sign is MSD)
11=Reserved

The length of the literal expressed in binary is encoded in LLGTH1 and LLGTH2. If the length of the literal is less than eight digits or characters, its length is encoded in LLGTH1; and LLGTH2 is omitted. If the length of the literal is greater than or equal to eight digits or characters, its length is encoded in LLGTH2 and LLGTH1 is set to zero. The maximum literal length is 255 digits or characters excluding the sign.

CURRENT OPERAND INDEX (COPX)

The argument COPX is an index value used to index into the current operand table (COP table). The number of bits (COPXB) used to index into the COP table is a function of the maximum number of COP table entries required for the source program. For example, a COP table consisting of between 512 and 1023 entries would require ten bits.

The address of an entry is calculated by multiplying the value "COPX" by the value "COPB" and then adding the result to the base address of the COP table.

A COPX value of zero specifies that the COP table information is contained in-line in the S-Instruction itself rather than in the COP table. (See next section.)

Note: The base address of the COP table points to an unused entry.

IN-LINE COP INFORMATION

The format for in-line COP information differs from its COP table format (See "CURRENT OPERAND TABLE") when subscripting or indexing is required.

The format for in-line COP information is as follows:

DATA SEG.# (SEGB)	ADDRESS DISPL. (DISPB)	DATA LENGTH (LENB)	SUBSCRIPT-OR-INDEX-FLAG (1)	DATA TYPE (2)	ASCII FLAG (1)

NO. OF SUBSCRIPTS OR INDEXES (2)	SUBSCRIPT-FLAG (1)	COPX1 (COPXB)	SUBSCRIPT FACTOR 1 (LENB)	COPX2 (COPXB)	SUBSCRIPT FACTOR 2 (LENB)

COPX3 (COPXB)	SUBSCRIPT FACTOR 3 (LENB)	TABLE BOUND (DISPB)

Notes:

1. None of the subscripting/indexing information (all entries following the ASCII flag) is present unless the subscript-or-index-flag equals one.

2. A COPX for each index value, or a COPX/subscript factor pair for each subscript value, must be present as indicated by the value of number of subscripts or indexes:
   00 = One
   01 = Two
   10 = Three
   11 = Reserved

3. COPX1, COPX2, and COPX3 may be in-line entries but must not be subscripted or indexed.

CURRENT OPERAND TABLE (COP)

The COP table consists of a set of entries, each of which contains the attributes of a variable. The width of one entry is a function of the source program and is determined by the number of bits required to express its attributes (segment number, displacement, length, subscript-or-index-flag, data type and ASCII flag).

When the attributes exceed one entry, muliple entries are used to accommodate the additional information. Any reference to a multiple entry attribute points to the first of its entries.

The format of an entry in the COP table is as follows:

```
|<-----------------------COPB-------------------------->|
---------------------------------------------------------
 DATA     ADDRESS  DATA    SUBSCRIPT-OR-        DATA  ASCII
 SEG#     DISPL.   LENGTH  INDEX-FLAG           TYPE  FLAG
 (SEGB)   (DISPB)  (LENB)  (1)                  (2)   (1)
---------------------------------------------------------
```

```
----------------------------------------------------------------------------
 NO. OF       SUBSCRIPT-FLAG  SUBSCRIPT  SUBSCRIPT  SUBSCRIPT  TABLE
 SUBSCRIPTS                   FACTOR 1   FACTOR 2   FACTOR 3   BOUND
 OR INDEXES
 (2)          (1)             (LENB)     (LENB)     (LENB)     (DISPB)
-+------+--------+------------+----------+----------+----------+-------
 |      |        |            |          |          |          |
 |      |        |            |          |          |          |
 |      ---------             |          |          |          |
 |               |            |          |          |          |
 |        PRESENT IF          |   PRESENT IF        |      PRESENT IF
 |        SUBSCRIPT-OR-       |   NUMBER OF         |      SUBSCRIPT-OR-
 |        INDEX-FLAG          |   SUBSCRIPTS        |      INDEX-FLAG
 |        =1                  |   = 01 OR 10        |      =1
 00 = 1            PRESENT IF SUBSCRIPT-            |
 01 = 2            OR-INDEX-FLAG = 1                PRESENT IF NUMBER
 10 = 3                                             OF SUBSCRIPTS = 10
 11 = RESERVED
```

SEGMENT NUMBER

Segment number is expressed in binary and specifies the data segment number of the operand. The container size (SEGB) is a function of the maximum number of data segments specified in the source program. The range of the segment number container size (SEGB) is 0 through 18. If it is zero, then there is no segment number container for that particular program; that program has no segmented (overlayable) data and all data references are to data segment zero, the non-overlayable data segment.

DISPLACEMENT

Displacement is expressed in binary and specifies the digit displacement of the data from the base of the data segment. All data is stored beginning at an address which modulo 4-BIT must equal zero. The container size (DISPB) is a function of the maximum data displacement specified in the source program. The range of the displacement container size (DISPB) is 1 through 21.

DATA LENGTH

Data length is expressed in binary and specifies the number of digits or characters in the data item, excluding the sign. The container size (LENB) is a function of the maximum length specified in the source program. The range of the data length container size (LENB) is 1 through 14; however, the largest data item allowed is 8,191 8-BIT units or 16,383 4-BIT units.

DATA TYPE

Data type specifies the type of data as follows:

00 = Unsigned 4-BIT
01 = Unsigned 8-BIT
10 = Signed 4-BIT (sign is MSD)
11 = Signed 8-BIT (sign over MSD)

SUBSCRIPT-OR-INDEX-FLAG

The subscript-or-index-flag bit is true to indicate subscripting or indexing and false otherwise. When true the next entry(s) contains the necessary subscripting or indexing information.

NUMBER OF SUBSCRIPTS OR INDEXES

When indexing or subscripting is indicated by the subscript-or-index-flag, the number of subscripts or indexes required for the variable is specified as follows:

00 = One
01 = Two
10 = Three
11 = Reserved

The bit immediately following this field indicates the appropriate operation: indexing or subscripting.

0 = Index
1 = Subscript

SUBSCRIPT FACTORS

Subscripting requires one to three fields, LENB bits in length, containing the binary factor by which each subscript value is to be multiplied to obtain the proper digit address. The factor is the digit displacement between elements of the table. The value one is subtracted from the subcript value prior to multiplying by the factor. The subscript value may be signed.

If the subscript value is zero or negative, or if the final sum of the multiplied subscript values exceeds the table bound, an error communicate will be issued.

If the binary equivalent of the multiplied subscript value or the sum of the multiplied subscript values exceeds 24 bits, overflow is ignored.

A COPX for each subscript value immediately follows the primary COPX in the S-Instruction. A subscript variable must not itself be subscripted or indexed.

Note: Literal subscript values are optimized by the compiler by building a new descriptor in-line in the S-Instruction.

INDEXING

When indexing is indicated, a COPX for each index value (up to three) immediately follows the primary COPX in the S-Instruction. An index variable must not itself be indexed or subscripted.

An index value is contained in a 28 BIT field. The value consists of a 4-BIT sign followed by six 4-BIT decimal digits. The value is converted to binary and combined with the binary data address at execution time.

If any index value is less than zero or if the sum of the index values exceeds the table bound, an error communicate will be issued.

TABLE BOUND

Table bound is a binary value used to specify the maximum permissible digit displacement from a table base for subscripting and indexing. Its container size is DISPB.

ASCII FLAG

The ASCII flag bit of the destination field influences the execution of certain code sensitive S-language instructions. These instructions are:

ADD	MVA	CAT	CMPA
SUB	MVS	SMVN	CMPS
INC	MVN	MVT	
DEC	MVZ		
INC1			
DEC1			
DIV			
DIVS			

The ASCII flag bit does not influence the execution of the following code sensitive instructions in which EBCDIC is assumed:

EDIT	MICF
EDTE	MICE
CMPC	

INSTRUCTION SET

ARITHMETIC

NAME	MNEMONIC	OP	ARGUMENTS
INCREMENT	INC	02	OPND1, COPX1
ADD	ADD	08	OPND1, COPX1, COPX2
DECREMENT	DEC	09	OPND1, COPX1
SUBTRACT	SUB	10	OPND1, OPND2, COPX1
MULTIPLY	MULT	11	OPND1, COPX1, COPX2
DIVIDE	DIV	12	OPND1, COPX1, COPX2
DIVIDE SPECIAL	DIVS	16	OPND1, COPX1, COPX2
INCREMENT BY ONE	INC1	13	COPX1
DECREMENT BY ONE	DEC1	14	COPX1

DATA MOVEMENT

NAME	MNEMONIC	OP	ARGUMENTS
MOVE ALPHANUMERIC	MVA	00	OPND1, COPX1
MOVE SPACES	MVS	15	COPX1
MOVE NUMERIC	MVN	01	OPND1, COPX1
MOVE ZEROS	MVZ	22	COPX1

CONCATENATE	CAT	32	N, COPX1, OPND0,...., OPNDN
SCALED MOVE NUMERIC	SMVN	28	OPND1, COPX1, V, SCL
EXAMINE	EXAM	44	M, T, COPX1, OPND1, COPX2, OPND2
MOVE TRANSLATE	MVT	47	OPND1, COPX1, COPX2
EDIT	EDIT	17	OPND1, COPX1, DADDR
EDIT WITH EXPLICIT MASK	EDTE	21	OPND1, COPX1, MASK
MICR FORMAT	MICF	48	COPX1, COPX2
MICR EDIT	MICE	49	COPX1, COPX2, COPX3

BRANCHING

NAME	MNEMONIC	OP	ARGUMENTS
BRANCH ON OVERFLOW	BOFL	23	V, BADDR
SET OVERFLOW	SOFL	07	V
BRANCH UNCONDITIONALLY	BUN	03	BADDR
PERFORM ENTER	PERF	06	K, BADDR
PERFORM EXIT	PXIT	34	K
ENTER	NTR	18	BADDR
EXIT	XIT	19	
GO TO DEPENDING	GOTD	39	COPX1, L, DBADDR0,...., DBADDRL

ALTERED GO TO PARAGRAPH	GPAR	35	DADDR
ALTER	ALTR	36	DADDR, ACON

CONDITIONAL BRANCHING

NAME	MNEMONIC	OP	ARGUMENTS
COMPARE ALPHANUMERIC	CMPA	04	OPND1, COPX1, R, BADDR
COMPARE NUMERIC	CMPN	05	OPND1, COPX1, R, BADDR
COMPARE FOR ZEROS	CMPZ	27	COPX1, R, BADDR
COMPARE FOR SPACES	CMPS	37	COPX1, R, BADDR
COMPARE FOR CLASS	CMPC	38	COPX1, C, BADDR
COMPARE REPEAT	CMPR	45	OPND1, COPX1, R, BADDR

MISCELLANEOUS

NAME	MNEMONIC	OP	ARGUMENTS
COMMUNICATE	COMM	33	COPX1
LOAD COMMUNICATE REPLY	LDCR	41	DADDR

CONVERT	CONV	40	COPX1, DADDR
MAKE PRESENT	MAKP	42	COPX1, DADDR
HARDWARE MONITOR	HMON	43	OPND1

ARITHMETIC OPERANDS AND INSTRUCTIONS

In general, arithmetic operands can have any of the following formats:

1. Unsigned 4-BIT
2. Unsigned 8-BIT
3. Signed 4-BIT (sign is MSD)
4. Signed 8-BIT (sign over MSD)

Any restrictions concerning the types of operands permitted in an operation are specified under the description of the particular operation.

All fields are addressed by pointing to the most significant bit of the most significant unit, which in the case of a signed field is the sign.

All fields are considered to be comprised of decimal integers.

The absolute value is stored if the receiving field is unsigned.

Unsigned fields are considered positive.

When signed format is specified for the receiving field for any arithmetic operation, the sign position is set to 1100 for a positive result and to 1101 for a negative result.

4-BIT operands are interpreted in units of four bits. When a signed operand is specified, the sign is interpreted as a separate and leading (leftmost) 4-BIT unit which is not included in the statement of length.

8-BIT operands are interpreted in units of eight bits. When a signed operand is specified, the sign is interpreted as being contained in the leftmost four bits of the leftmost 8-BIT unit.

The length of the operand field specifies the number of 4-BIT or 8-BIT units.

When 8-BIT units are specified for the receiving field of an arithmetic operation, the leftmost four bits of each 8-BIT unit, except the unit carrying a sign, is set to 1111 if EBCDIC or to 0011 if ASCII.

The value of an 8-BIT unit is carried in the rightmost four bits of the unit. Its value is as defined below for the 4-BIT unit. The leftmost four bits, except for a sign, are ignored. The value and sign interpretation of a 4-BIT unit is as follows:

UNIT	VALUE	SIGN
0000	0	+
0001	1	+
0010	2	+
0011	3	+
0100	4	+
0101	5	+
0110	6	+
0111	7	+
1000	8	+
1001	9	+
1010	UNDEFINED	+
1011	UNDEFINED	+
1100	UNDEFINED	+
1101	UNDEFINED	-
1110	UNDEFINED	+
1111	UNDEFINED	+

In addition and subtraction, results generated when the size of the result field is not sufficient to contain the result are not specified. When the result field is longer than the length of the result, leading zero units are stored.

In three address add, three address subtract and in multiply, total or partial overlap of the first two operands is permitted. Results generated when the result field totally or partially overlaps either of the operand fields are not specified.

In two address add and subtract, total overlap is permitted. Results generated when the result field partially overlaps the first operand field are not specified. Note that total overlap implies that the two fields are identical.

No overlap of operands or result fields is permitted in divide. Results generated under any condition of overlap are not specified.

ADD THREE ADDRESS

```
*******
* ADD *
*******
```

OP: 08

Format:

```
****************************
* ADD OPND1, COPX1, COPX2 *
****************************
```

Function:

Algebraically add an addend denoted by OPND1 to an augend denoted by COPX1 and store the sum in the field denoted by COPX2.

SUBTRACT THREE ADDRESS

```
*******
* SUB *
*******
```

OP: 10

Format:

```
****************************
* SUB OPND1, OPND2, COPX1 *
****************************
```

Function:

Algebraically subtract a subtrahend denoted by OPND1 from a minuend denoted by OPND2 and store the difference in the field denoted by COPX1.

ADD TWO ADDRESS

```
*******
* INC *
*******
```

OP: 02

Format:

```
*********************
* INC OPND1, COPX1 *
*********************
```

Function:

Algebraically add an addend denoted by OPND1 to an augend denoted by COPX1 and store the sum in the field denoted by COPX1.

SUBTRACT TWO ADDRESS

```
*******
* DEC *
*******
```

OP: 09

```
*********************
* DEC OPND1, COPX1 *
*********************
```

Function:

Algebraically subtract a subtrahend denoted by OPND1 from a minuend denoted by COPX1 and store the difference in the field denoted by COPX1.

MULTIPLY

```
********
* MULT *
********
```

OP: 11

Format:

```
*****************************
* MULT OPND1, COPX1, COPX2 *
*****************************
```

Function:

Algebraically multiply a multiplicand denoted by COPX1 by a multiplier denoted by OPND1 and store the product in the field denoted by COPX2.

The result field length is the sum of the lengths of the two operands and must be denoted by COPX2.

The result field will always be either signed 4-BIT format or unsigned 4-BIT format.

DIVIDE

```
*******
* DIV *
*******
```

OP: 12

Format:

```
****************************
* DIV OPND1, COPX1, COPX2 *
****************************
```

Function:

Algebraically divide a dividend denoted by COPX1 by a divisor denoted by OPND1 and store the quotient in the field denoted by COPX2. Store the remainder in the field denoted by COPX1.

The result field length is the difference of the lengths of the two operands and must be denoted by COPX2.

Results are not specified if the length of the dividend is not greater than the lenth of the divisor.

If the absolute value of the divisor is not greater than the absolute value of an equivalent number of leading digits of the dividend, the result is undefined.

Division by zero results in a fatal error communicate to the MCP.

The sign of the remainder is that of the original dividend.

The dividend field will always be either signed 4-BIT format or unsigned 4-BIT format.

DIVIDE SPECIAL

```
********
* DIVS *
********
```

OP: 16

Format:

```
*****************************
* DIVS OPND1, COPX1, COPX2 *
*****************************
```

Function:

This operation is performed in exactly the same manner as the standard divide (DIV) operator; except that when a divisor equal to zero is encountered, an overflow toggle is set and processing is allowed to continue. The overflow toggle can be manipulated by the "SOFL" and "BOFL" S-operators.

INCREMENT BY ONE

```
********
* INC1 *
********
```

OP: 13

Format:

```
***************
* INC1 COPX1 *
***************
```

Function:

Algebraically add the positive integer one to an augend denoted by COPX1 and store the sum in the field specified by COPX1.

DECREMENT BY ONE

```
********
* DEC1 *
********
```

OP: 14

Format:

```
***************
* DEC1 COPX1 *
***************
```

Function:

Algebraically subtract the positive integer one from a minuend denoted by COPX1 and store the difference in the field specified by COPX1.

DATA MOVEMENT OPERANDS AND INSTRUCTIONS

In general, fields involved in data movement operations can have any of the following formats:

1. Unsigned 4-BIT
2. Unsigned 8-BIT
3. Signed 4-BIT (sign is MSD)
4. Signed 8-BIT (sign over MSD)

Any restrictions as to the type of fields permitted in an operatior are specified under the description of the particular operation.

See arithmetic operands and instructions for a description of th four types of fields.

Totally or partially overlapped fields are not permitted, unless specifically specified by the description of the individual instruction.

MOVE ALPHANUMERIC

```
*******
* MVA *
*******
```

OP: 00

Format:

```
*********************
* MVA OPND1, COPX1 *
*********************
```

Function:

Move 8-BIT or 4-BIT units from the source field denoted by OPND1 to the 8-BIT or 4-BIT destination field denoted by COPX1.

If the destination field is signed, it receives either the sign of the source if the source is signed, or 1100 if the source is unsigned.

If the data type of the source field is 4-BIT and the data type of the destination field is 8-BIT, each 4-BIT unit is moved to the destination with 1111 if EBCDIC or 0011 if ASCII moved to the leftmost four bits of each 8-BIT unit.

If the data type of the source field is 8-BIT and the data type of the destination is 4-BIT, the rightmost four bits are moved.

If the data type of the source field is the same as the data type of the destination field, each unit is moved unchanged to the destination.

If the destination length is greater in size than the source length, the destination field is filled in on the right with trailing spaces (0100 0000 if EBCDIC or 0010 0000 if ASCII) if the destination type is 8-BIT; otherwise it is filled in on the right with zeros (0000).

If the destination length is lesser in size than the source length, the source data is truncated on the right.

Overlapping operand fields are permitted if the data type of both fields is the same. It can be assumed that the source is moved 24 bits (six digits or three characters) at a time into the destination field and that the move is from left to right.

MOVE SPACES

```
*******
* MVS *
*******
```

OP: 15

Format:

```
*************
* MVS COPX1 *
*************
```

Function:

Fill the destination field denoted by COPX1 with spaces (0100 0000 if EBCDIC or 0010 0000 if ASCII).

The data type of the destination field is ignored and is assumed to be unsigned 8-BIT.

MOVE NUMERIC

```
*******
* MVN *
*******
```

OP: 01

Format:

```
*********************
* MVN OPND1, COPX1 *
*********************
```

Function:

Move 8-BIT or 4-BIT units from the source field denoted by OPND1 to the 8-BIT or 4-BIT destination field denoted by COPX1.

If the destination field is signed, it receives either the sign of the source if the source is signed, or 1100 if the source is unsigned.

If the destination field is unsigned, the sign of the source is ignored.

If the data type of the destination field is 8-BIT, the leftmost four bits of each 8-BIT unit, except for the sign position, if signed, are set to 1111 if EBCDIC or to 0011 if ASCII, regardless of the data type of the source field.

If the data type of the destination field is 4-BIT, the leftmost four bits of each source 8-BIT unit are ignored and only the rightmost four bits are moved; if the source field is a 4-BIT field, each 4-BIT unit is moved unchanged.

left zero fill

If the destination length is greater in size than the source length, the destination field is filled in on the left with leading zeros of appropriate type (1111 0000 if EBCDIC, 0011 0000 if ASCII or 0000 if 4-BIT).

left truncate

If the destination length is lesser in size than the source length, the source data is truncated on the left.

sign in left digit

Note that a sign is placed in the leftmost four bits of a field, whether 4-BIT or 8-BIT.

Overlapping operand fields are permitted if the data type of both fields is the same. It can be assumed that the source is moved 24 bits (six digits or three characters) at a time into the destination field and that the move is from left to right.

MOVE ZEROS

```
*******
* MVZ *
*******
```

OP: 22

Format:

```
*************
* MVZ COPX1 *
*************
```

Function:

Fill the destination field denoted by COPX1 with zeros of the appropriate type (1111 0000 if EBCDIC, 0011 0000 if ASCII or 0000 if 4-BIT).

If the destination field is signed, 1100 is placed into the sign position.

CONCATENATE

```
*******
* CAT *
*******
```

OP: 32

Format:

```
************************************
* CAT N, COPX1, OPNDO, ..., OPNDN *
************************************
```

Function:

Move each of the N+1 fields denoted by OPNDO through OPNDN, in the order specified, into an output string starting at the field denoted by COPX1.

The number of source fields is specified by the 4-BIT binary value N. The value N ranging from 0000 to 1111 is used to indicated 1 to 16 source fields.

Each field is moved according to the rules specified for MOVE ALPHANUMERIC.

If the destination length is greater in size than the combined source length, the destination field is filled in on the right with trailing spaces (0100 0000 if EBCDIC or 0010 0000 if ASCII).

If the destination length is lesser in size than the combined source lengths, the source data is truncated on the right.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

COMPANY CONFIDENTIAL
B1700 COBOL S-LANGUAGE
P.S. 2201 6729

Add scaled = move scaled + add
SUB scaled
Compare scaled
Move scaled

SCALED MOVE NUMERIC

```
********
* SMVN *
********
```

OP: 28

Format:

```
*******************************
* SMVN OPND1, COPX1, V, SCL *
*******************************
```

Function:

Move Scl Zeroes to right of dest, Then ~~perform~~ Move Numeric

If V equals 0, perform a MOVE NUMERIC operation after first adding the scale factor to the field length of the source field and assuming that the added portion of the field is zeros on the right. The scale factor must not be greater than the destination field length.

?. If V equals one, perform a MOVE NUMERIC operation after first subtracting the scale factor from the field length of the source field. The scale factor must not be greater than the source field length.

All rules specified for MOVE NUMERIC are applicable after adjustment by the scale factor.

The container size for the scale factor is the same as the container size for the length of an operand (LENB). The length of V is one bit.

MOVE TRANSLATE

```
*******
* MVT *
*******
```

OP: 47

Format:

```
****************************
* MVT OPND1, COPX1, COPX2 *
****************************
```

Function:

Move 8-BIT units from the source field denoted by OPND1 to the destination field denoted by COPX2, translating enroute.

Translation is accomplished by using each 8-BIT source character, multiplied by eight, as an index into the translation table, denoted by COPX1, to obtain the translated character.

The data type of the source and table fields are ignored and are assumed to be unsigned 8-BIT. The destination field is also assumed to be unsigned, but may be 4-BIT or 8-BIT.

If the destination length is greater in size than the source length, the destination field is filled in on the right with trailing spaces (0100 0000 if EBCDIC, 0010 0000 if ASCII or 0000 if 4-BIT).

If the destination length is lesser is size than the source length, the source data is truncated on the right.

Total overlap of operand fields is permitted to allow inplace translation.

EXAMINE

```
********
* EXAM *
********
```

OP: 44

Format:

```
******************************************
* EXAM M, T, COPX1, OPND1, COPX2, OPND2 *
******************************************
```

Function:

Examine the operand defined by COPX1, tallying and/or replacing a variable number of 8-BIT characters. The particular 8-BIT character to be tallied and/or replaced is specified by OPND1. The character to be used as the replacement character is specified by OPND2. The field into which the tally is stored is specified by COPX2.

The type of operation is specified by the 4-BIT parameter, T. If these four bits are identified, left to right, as T1, T2, T3 and T4, then T is interpreted as follows:

```
T1T2 = 00 undefined
       01 tally T3T4 occurrences of the character
          specified by OPND1
       10 replace T3T4 occurrences of the character
          specified by OPND1
       11 tally and replace T3T4 occurrences of the
          character specified by OPND1

T3T4 = 00 all
       01 (all) leading
```

10 until first
11 first

NOTE: T1T2T3T4 = 0111 and 1111 not specified and results are undefined.

The OPND2 argument is not present when T1T2 = 01.

The COPX2 argument is not present when T1T2 = 10.

The data type of the examined operand (COPX1) is assumed to be signed or unsigned 8-BIT. If it is signed, then the original sign will be preserved by this operation.

The data type of the examining operand, defined by OPND1, must be unsigned. Its length is assumed to be one. When 4-BIT format is specified, the operand is assumed to have the four bits 1111 if EBCDIC or 0011 if ASCII appended to the left.

The data type of the replacing operand, defined by OPND2, must be unsigned. Its length is assumed to be one. When 4-BIT format is specified, the leftmost four bits of the position replaced are set to 1111 if EBCDIC or 0011 if ASCII, and the rightmost four bits receive the four bits from the replacing source. When 8-BIT format is specified, the position replaced receives all eight bits from the replacing source.

The data type of the tally field defined by COPX2 is assumed to be unsigned 4-BIT. Its length is assumed to be five.

If the one bit parameter M equals zero, it denotes numeric items, and only the rightmost four bits of a character are used in the comparison; the leftmost four bits are ignored. If M equals one, alphanumeric items are denoted, and all eight bits of a character are used in comparing.

EDIT INSTRUCTIONS AND EDIT MICRO-OPERATORS

No restrictions are placed on the data type of the source field of an edit operation.

The data type of the destination field of an edit operation must be unsigned 8-BIT.

If the destination length is greater in size than the source length, the source data is assumed to have leading zero fill on the left.

If the destination length is lesser in size than the source length, the source data is truncated on the left.

The operation is terminated by an edit micro-operator and not by exhaustion of either the source or destination fields.

EDIT

```
********
* EDIT *
********
```

OP: 17

Format:

```
****************************
* EDIT OPND1, COPX1, DADDR *
****************************
```

Function:

Move data from the source field, denoted by OPND1, to the destination field, denoted by COPX1, under the control of the micro-operator string contained at the location denoted by the DADDR.

The argument DADDR is an unsigned binary value which specifies the digit displacement of the micro-operator string relative to the data segment zero base. The container size of DADDR is DISPB.

EDIT WITH EXPLICIT MASK

```
********
* EDTE *
********
```

OP: 21

Format:

```
******************************
* EDTE OPND1, COPX1, MASK *
******************************
```

Function:

Move data from the source field denoted by OPND1 to the destination field denoted by COPX1 under the control of the micro-operator string immediately following COPX1. The format of the explicit micro-operator string is the same as a literal and is as follows:

```
------------------------------------------------
 LTYPE  LLGTH1  LLGTH2  MICRO-OPERATOR STRING
 (2)    (3)     (8)     (variable)
-+------+-------+-------------------------------
 !      !       !
 !      !       !
 !      !      Present if LLGTH equals zero
 !      !
 !  Length of the micro-operator string in
 !  8-BIT units.  If length is greater than or
 !  equal to eight units, the length is encoded
 !  in LLGTH2 and LLGTH 1 is set to zero.
 !
01: Unsigned 8-BIT format
```

EDIT MICRO-OPERATORS

The edit micro-operators used in an edit instruction are:

OPERATOR	MNEMONIC	OPERATION
0000 R	MVD	MOVE DIGITS
0001 R	MVC	MOVE CHARACTERS
0010 R	MVS	MOVE SUPPRESS
0011 R	FIL	FILL SUPPRESS
0100 N	SRD	SKIP REVERSE DESTINATION
0101 T	INU	INSERT UNCONDITIONALLY
0110 T	INM	INSERT ON MINUS
0111 T	INS	INSERT SUPPRESS
1000 T	INF	INSERT FLOAT
1001 T	EFM	END FLOAT MODE
1010 0000	ENZ	END NON-ZERO
1010 0001	EOM	END OF MASK
1010 0010	SZS	START ZERO SUPPRESS
1010 0011	CCP	COMPLEMENT CHECK PROTECT
OTHERS		UNDEFINED

"R" indicates a 4-BIT binary value used as a repeat count. The value 0000 represents no repeat, do it once.

"N" indicates a 4-BIT binary value used to skip over a number of destination 8-BIT units. The value 0000 represents no skip.

"T" indicates a 4-BIT binary value which is:

1) used to index into a table of editing constants
2) used to indicate a conditional selection between two table constants
3) used to indicate an editing constant in line with the edit-operator string.

The next edit-operator follows the constant.

The following table indicates the normal table editing constants as well as the conditional and unconditional selection of constants associated with the value "T".

EDITING CONSTANTS

T	TABLE ENTRY EBCDIC	MNEMONIC	UNCONDITIONAL OR CONDITIONAL CONSTANT
0000	"+"	PLU	
0001	"-"	MIN	
0010	"*"	AST	
0011	"."	DPT	
0100	","	CMA	
0101	"$"	CUR	
0110	"0"	ZRO	
0111	" "	BLK	
1000		SPM	EITHER ENTRY 0 OR 1
1001		SBM	EITHER ENTRY 7 OR 1
1010		LIT	IN-LINE 8-BIT CONSTANT

Associated with the edit instructions are three toggles denoted as "S" for sign, "Z" for zero suppress and "P" for check protect. Initially the "Z" and the "P" toggles are assumed to be set to the zero state. They are set and reset as specified by the description of the individual micro-operators. The "S" toggle is set to zero if the source field sign is positive and to one otherwise. Unsigned fields are considered positive.

The EDIT MICRO-OPERATORS are explained individually in the following section.

MOVE DIGIT

Set "Z" to "1", ending the zero suppress state. Move an appropriate unit (4-BIT digit or 8-BIT character) from the source field to the destination field. If a 4-BIT unit is moved, append the four bits 1111 to the left before storing in the destination. If an 8-BIT unit is moved, the four bits 1111 are substituted for the leftmost four bits of the 8-BIT unit.

MOVE CHARACTER

Set "Z" to "1", ending the zero suppress state. Move an appropriate unit (4-BIT digit or 8-BIT character) from the source field to the destination field. If a 4-BIT unit is moved, append the four bits 1111 to the left before storing in the destination. If an 8-BIT unit is moved, it is moved unchanged.

MOVE SUPPRESS

The micro-operator "MOVE DIGIT" is performed if the 4-BIT unit, or the rightmost four bits of the 8-BIT unit, of the source field is not equal to 0000.

If the appropriate four bits of the source field unit are equal to 0000, the suppress toggle "Z" is inspected. If "Z" equals "1", indicating non-suppress mode, the micro-operator "MOVE DIGIT" is performed. If the suppress toggle "Z" equals "0", the check protect toggle "P" is inspected. If "P" = "0", indicating non-check protect mode, move the table entry containing the 8-BIT code for blank to the destination field. If "P" = "1", move the table entry containing the 8-BIT code for asterisk to the destination field.

SUMMARY

```
        SOURCE NOT = 0  MOVE DIGIT
Z=1     SOURCE     = 0  MOVE DIGIT
Z=0 P=0 SOURCE     = 0  MOVE TABLE ENTRY 7 (BLANK)
Z=0 P=1 SOURCE     = 0  MOVE TABLE ENTRY 2 (ASTERISK)
```

FILL SUPPRESS

If "P" = "0", indicating non-check protect mode, move the table entry containing the 8-BIT code for blank to the destination field. If "P" = "1", move the table entry containing the 8-BIT code for asterisk to the destination field.

SUMMARY

P = 0	MOVE TABLE ENTRY 7 (BLANK)
P = 1	MOVE TABLE ENTRY 2 (ASTERISK)

SKIP REVERSE DESTINATION

Adjust the address pointer of the destination field to skip backward (lower address) "N" 8-BIT units.

INSERT UNCONDITIONALLY

Move the table entry "T" as indicated below to the destination field.

	T=0...7	MOVE TABLE ENTRY T
S=0	T=8	MOVE TABLE ENTRY 0 (PLUS)
S=1	T=8	MOVE TABLE ENTRY 1 (MINUS)
S=0	T=9	MOVE TABLE ENTRY 7 (BLANK)
S=1	T=9	MOVE TABLE ENTRY 1 (MINUS)
	T=10	MOVE IN-LINE TABLE ENTRY

INSERT ON MINUS

Move the table entry "T" as indicated below to the destination field.

S=1		T=0...7	MOVE TABLE ENTRY T
*	P=0		MOVE TABLE ENTRY 7 (BLANK)
*	P=1		MOVE TABLE ENTRY 2 (ASTERISK)
S=1		T=8	MOVE TABLE ENTRY 1 (MINUS)
S=1		T=9	MOVE TABLE ENTRY 1 (MINUS)
S=1		T=10	MOVE IN-LINE TABLE ENTRY

*: S = 0 or only source digits/characters equal to zero (minus zero) have been moved.

INSERT SUPPRESS

Move the table entry "T" as indicated below to the destination field.

Z=1			T=0...7	MOVE TABLE ENTRY T
Z=0	P=0			MOVE TABLE ENTRY 7 (BLANK)
Z=0	P=1			MOVE TABLE ENTRY 2 (ASTERISK)
Z=1		S=0	T=8	MOVE TABLE ENTRY 0 (PLUS)
Z=1		S=1	T=8	MOVE TABLE ENTRY 1 (MINUS)
Z=1		S=0	T=9	MOVE TABLE ENTRY 7 (BLANK)
Z=1		S=1	T=9	MOVE TABLE ENTRY 1 (MINUS)
Z=1			T=10	MOVE IN-LINE TABLE ENTRY

INSERT FLOAT

Move the table entry "T" and/or perform the micro-operator "MOVE DIGIT" as indicated below.

Z=1					MOVE DIGIT
Z=0	SOURCE	=0	P=0		MOVE TABLE ENTRY 7 (BLANK)
Z=0	SOURCE	=0	P=1		MOVE TABLE ENTRY 2 (ASTERISK)
Z=0	SOURCE	NOT=0	T=0..7		MOVE TABLE ENTRY T, THEN MOVE DIGIT
Z=0	SOURCE	NOT=0	T=8	S=0	MOVE TABLE ENTRY 0 (PLUS) THEN MOVE DIGIT
Z=0	SOURCE	NOT=0	T=8	S=1	MOVE TABLE ENTRY 1 (MINUS) THEN MOVE DIGIT
Z=0	SOURCE	NOT=0	T=9	S=0	MOVE TABLE ENTRY 7 (BLANK) THEN MOVE DIGIT
Z=0	SOURCE	NOT=0	T=9	S=1	MOVE TABLE ENTRY 1 (MINUS) THEN MOVE DIGIT
Z=0	SOURCE	NOT=0	T=10		MOVE IN-LINE TABLE ENTRY, THEN MOVE DIGIT

END FLOAT MODE

Move the table entry "T" as indicated below to the destination field.

Z=0		T=0...7	MOVE TABLE ENTRY T
Z=0	S=0	T=8	MOVE TABLE ENTRY 0 (PLUS)
Z=0	S=1	T=8	MOVE TABLE ENTRY 1 (MINUS)
Z=0	S=0	T=9	MOVE TABLE ENTRY 7 (BLANK)
Z=0	S=1	T=9	MOVE TABLE ENTRY 1 (MINUS)
Z=0		T=10	MOVE IN-LINE TABLE ENTRY
Z=1	NO OPERATION		

END NON-ZERO

Terminate the micro-operator operations if any non-zero source character/digit has been moved; otherwise continue with the next in-line operator.

END OF MASK

Terminate the micro-operator operations.

START ZERO SUPPRESS

Set "Z" to the "0" state.

COMPLEMENT CHECK PROTECT

Complement the state of "P".

MICR FORMAT

```
********
* MICF *
********
```

OP: 48

Format:

```
**********************
* MICF COPX1, COPX2 *
**********************
```

Function:

Format the data from the source field denoted by COPX1 into the destination field denoted by COPX2.

The data type of both the source and the destination fields is assumed to be unsigned 8-BIT.

The field length of the destination MODULO 20 must equal zero. The destination field is considered to be composed of a number of 20 character subfields.

Data movement is right to left beginning with the rightmost character of the source field and beginning with the rightmost character position of the destination field.

In the discussion that follows, the following definitions apply:

1. Transfer characters are characters that are automatically transferred from the source field into the current destination subfield. They never

occupy the rightmost control character position of a destination subfield. They are the numeric "0" through "9" and the HYPHEN "-". The HYPHEN is not expected to occur for OCR input.

2. Defined control characters are characters that cause some specific action to be taken, depending on the character. They are: the END-OF-DOCUMENT "!", the MICR CANT-READ "*" and the OCR CANT-READ @3F@.

3. Default control characters are characters other than transfer and defined control characters. They are expected to be, but will not be limited to, the MICR AMOUNT "#", TRANSIT "@" and ON-US ":" and the OCR HOOK "<", FORK "=", CHAIR ">", VERTICAL BAR "|", BLANK @40@ and PLUS "+".

Operation is as follows:

1. Begin formatting into a subfield by fetching a source field character, unless the source field is exhausted, and then proceeding to step 1A.

   A. If the source field is exhausted, assume an END-OF-DOCUMENT (!) character and proceed to step 1B.

   B. If the source field character is an END-OF-DOCUMENT character, move it to the rightmost position of the current subfield, blank-fill the rest of the destination field and then terminate the operation.

   C. If the source field character is other than a default control or END-OF-DOCUMENT character, move a blank to the rightmost position of the current subfield, then move the source character and proceed to step 2A.

   D. If the source field character is a default control character, move it to the rightmost position of the current subfield and then proceed to step 2A.

2. Continue formatting into the current subfield by fetching a new source character and then proceeding to step 2A.

   A. If the source field is exhausted, blank-fill the rest of the current subfield, assume an END-OF-DOCUMENT source character and proceed to step 1B.

   B. If the source field character is an END-OF-DOCUMENT character, blank-fill the rest of the current subfield, save the source field character and proceed to step 1B.

   C. If the source field character is other than a default control or END-OF-DOCUMENT character, store the character in the destination and proceed to step 2A.

   D. If the source field character is a default control character that is equal to the character in the rightmost position of the current subfield, move it to the next position of the current subfield, blank-fill the rest of the current subfield and then proceed to step 1A.

   E. If the source field character is a default control character, but it is not equal to the character in the rightmost position of the current subfield, the rest of the current subfield is blank-filled and the control character is used in step 1D, to which we now proceed.

NOTES:
------

1. If any attempt is made to exceed the size of any subfield or of the entire destination field, the overflow toggle is set to one, the operation is terminated and the contents of the destination field are undefined.

2. If any individual subfield contains a CANT-READ ("*" or "@3F@) character, then the high order (leftmost) position of the subfield will be set to 1101 0001; otherwise, it will be set to a blank (0100 0000).

MICR EDIT

```
********
* MICE *
********
```

OP: 49

Format:

```
****************************
* MICE COPX1, COPX2, COPX3 *
****************************
```

Function:

Move data from the source field denoted by COPX1 to the destination field denoted by COPX2 deleting all characters except numeric characters ("0" through "9") and CANT-READ characters ("*" and @3F@).

The moved characters are right justified in the destination field and zero filled on the left, if necessary, to fill the remaining destination area. If the destination field is lesser in size than the moved data, the source data is truncated on the left.

A decimal count of all numeric characters moved is provided in the special COBOL register "TALLY" denoted by COPX3.

The data type of the source field must be unsigned 8-BIT. The data type of the destination field must be unsigned 4-BIT or 8-BIT. The data type of the "TALLY" field must be unsigned 4-BIT and its length is assumed to be five.

BRANCHING OPERANDS AND INSTRUCTIONS

A branch address argument "BADDR" has the following format:

```
---------------------------------------
 DISPLACEMENT  BTYPE  SEGMENT NUMBER
 (BDISPB)      (1)    (7)
--------------+------+--------------
              |      |
              |    present if BTYPE = 1
              |
              |
            0: Relative to the current code
               segment base (intrasegment branch)
            1: Relative to a new code segment base
               (intersegment branch)
```

Displacement is an unsigned binary value which specifies the bit displacement of an instruction relative to a segment base. The container size of the displacement and BTYPE combined is a program parameter (BDISPB1).

BRANCH UNCONDITIONALLY

```
*******
* BUN *
*******
```

OP: BUN

Format:

```
**************
* BUN BADDR *
**************
```

Function:

Obtain the next instruction from the location specified by BADDR.

BRANCH ON OVERFLOW

```
********
* BOFL *
********
```

OP: 23

Format:

```
*****************
* BOFL V, BADDR *
*****************
```

Function:

If the overflow toggle equals V, a transfer to the address (BADDR) given in the instruction occurs, otherwise control is passed to the next sequential instruction.

The overflow toggle is unchanged. The length of V is one bit.

SET OVERFLOW TOGGLE

```
********
* SOFL *
********
```

OP: 07

Format:

```
**********
* SOFL V *
**********
```

Function:

Set the overflow toggle to V.

The length of V is one bit.

NOTE: The overflow toggle is set to one if a "DIVIDE BY ZERO" is encountered in the DIVIDE SPECIAL S-operator or if a field overflow is attempted in the MICR format S-operator.

PERFORM ENTER

```
********
* PERF *
********
```

OP: 06

Format:

```
*****************
* PERF K, BADDR *
*****************
```

Function:

Create a stack entry with the following format:

DISPLACEMENT	SEGMENT NO.	K
(24)	(7)	(8)

Insert a displacement value, relative to the active code segment base and pointing to the next sequential S-instruction, into the stack.

Insert the current code segment number into the stack. Insert the value of K from the instruction into the stack.

Adjust the stack pointer to point to the next possible entry.

Obtain the next instruction from the location specified by BADDR.

PERFORM EXIT

```
*********
* PXIT *
*********
```

OP: 34

Format:

```
***********
* PXIT K *
***********
```

Function:

Compare the K contained in the instruction to the K in the current stack entry and if unequal proceed to the next in-line S-instruction. If equal, adjust the stack pointer to point to the previous entry and obtain the next S-instruction from the information contained in the removed stack entry.

ENTER

```
*******
* NTR *
*******
```

OP: 18

Format:

```
**************
* NTR BADDR *
**************
```

Function:

Same function as "PERF". K is assumed equal to zero.

EXIT

```
*******
* XIT *
*******
```

OP: 19

Format:

```
*******
* XIT *
*******
```

Function:

Same function as "PXIT". K is assumed equal to zero.

GO TO DEPENDING

```
********
* GOTD *
********
```

OP: 39

Format:

```
*****************************************
* GOTD COPX1, L, DBADDR0, ..., DBADDRL *
*****************************************
```

Function:

Compare the ten bit binary value L with the variable specified by COPX1. The variable is first converted to a binary value, MODULO 2 to the 24th power.

If the binary value of the variable is less than zero or greater than L, the next instruction is obtained from the location specified by DBADDR0. Note that the variable can be signed.

If the binary value of the variable is in the range zero through L, it is used as an index to select from the list of DBADDR's the appropriate DBADDR to be used to obtain the next instruction.

DBADDR and BADDR have the same format with the exception that DBADDR will always contain the segment number. Although segment number is unneccessary for those DBADDR's with BTYPE equal to zero, in order to index into the list of DBADDR's, all of the DBADDR's must be of equal length. The container size of DBADDR is BDISPB1 + 7.

ALTERED GO TO PARAGRAPH

```
********
* GPAR *
********
```

OP: 35

Format:

```
***************
* GPAR DADDR *
***************
```

Function:

Obtain the next instruction from the location specified by the address "ACON".

The address constant "ACON" has the same format as a BADDR.

The argument DADDR is an unsigned binary value which specifies the digit displacement of the "ACON" relative to the data segment zero base.

The container size of DADDR is DISPB.

ALTER

```
********
* ALTR *
********
```

OP: 36

Format:

```
*********************
* ALTR DADDR, ACON *
*********************
```

Function:

Copy the address constant "ACON" into the data area specified by the argument DADDR.

The address constant "ACON" has the same format as a BADDR.

The argument DADDR is an unsigned binary value which specifies the digit displacement of the "ACON" relative to the data segment zero base.

The container size of DADDR is DISPB.

CONDITIONAL BRANCH OPERANDS AND INSTRUCTIONS

If the condition "A (R) B" is true a transfer to the address (BADDR) given in the instruction occurs, otherwise control is passed to the next sequential instruction. The relation (R) is defined as follows:

000	UNDEFINED
001	GTR
010	LSS
011	NEQ
100	EQL
101	GEQ
110	LEQ
111	UNDEFINED

Overlap of fields is permitted. "A" is the first operand denoted in the instruction. If an instruction has only one operand, then the assumed field is the "A" field.

COMPARE ALPHANUMERIC

```
********
* CMPA *
********
```

OP: 04

Format:

```
*******************************
* CMPA OPND1, COPX1, R, BADDR *
*******************************
```

Function:

Compare the two operand fields according to their binary values.

The comparison is performed left to right with any shorter operand assumed to be right-filled with blank characters (0100 0000 if EBCDIC or 0010 0000 if ASCII).

The fields are considered equal when the equal size portions are equal and the longer (if one is longer) field has trailing blanks.

8-BIT data format is assumed for both fields with no checking to verify otherwise. Signed fields have their most significant four bits, i.e., their sign, modified to the appropriate numeric zone (1111 for EBCDIC, 0011 for ASCII) before being compared. This modification is not permanent and is done so that sign will not affect the result of an alphanumeric comparison.

COMPARE NUMERIC

```
********
* CMPN *
********
```

OP: 05

Format:

```
********************************
* CMPN OPND1, COPX1, R, BADDR *
********************************
```

Function:

Compare the two operand fields according to the algebraic values, considering the two fields to be comprised of decimal integers.

When the field sizes are different, the longer is tested for leading zeros (0000). There is no restriction as to data type. In comparing an 8-BIT character only the rightmost four bits of the character are considered; the other bits are ignored.

Two fields of all zeros are equal regardless of sign.

Unsigned fields are considered positive. Sign conventions are the same as for arithmetic operands.

Results generated by invalid digit values are undefined.

COMPARE FOR ZEROS

```
********
* CMPZ *
********
```

OP: 27

Format:

```
*************************
* CMPZ COPX1, R, BADDR *
*************************
```

Function:

Compare two operand fields according to their algebraic values, assuming the first field to be comprised of all zeros (0000).

There is no restriction as to data type. In comparing an 8-BIT character only the rightmost four bits of the character are considered. The other bits are ignored.

Two fields of all zeros are equal regardless of sign.

Unsigned fields are considered positive. Sign conventions are the same as for arithmetic operands.

Results generated by invalid digit values are undefined.

COMPARE FOR SPACES

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

OP: 37

Format:

```
*************************
* CMPS COPX1, R, BADDR *
*************************
```

Function:

Compare two operand fields according to their binary values, assuming the first field to be comprised of all spaces (0100 0000 if EBCDIC or 0010 0000 if ASCII).

The comparison is performed left to right.

Unsigned 8-BIT format is assumed with no checking to verify otherwise.

COMPARE FOR CLASS

```
********
* CMPC *
********
```

OP: 38

Format:

```
*************************
* CMPC COPX1, C, BADDR *
*************************
```

Function:

Compare the operand field and determine whether the field is:

```
C=00 COMPLETELY ALPHABETIC
  01 COMPLETELY NUMERIC
  10 NOT COMPLETELY ALPHABETIC
  11 NOT COMPLETELY NUMERIC
```

If the condition being tested is true, a transfer to the address BADDR given in the instruction occurs, otherwise control is passed to the next sequential instruction.

In the alphabetic test, each character is range-checked for 1100 0001 through 1100 1001, 1101 0001 through 1101 1001, 1110 0010 through 1110 1001 and for 0100 0000. Unsigned 8-BIT format is assumed with no checking to verify otherwise.

In the numeric test each character is range-checked for 1111 0000 through 1111 1001. Signed or unsigned 8-BIT format is permitted. The four bits in the sign position of a signed 8-BIT field are ignored. The sign position is the leftmost four bits of the most significant character.

COMPARE REPEAT

```
********
* CMPR *
********
```

OP: 45

Format:

```
*********************************
* CMPR OPND1, COPX1, R, BADDR *
*********************************
```

Function:

Compare the two operand fields according to their binary value.

Comparison proceeds from left to right.

The field lengths are considered equal by repeating OPND1.

Both fields are assumed to have unsigned 8-BIT data type.

The size of OPND1 must divide evenly into the size of COPX1; otherwise, the results of the compare may be erroneous.

MISCELLANEOUS INSTRUCTION

COMMUNICATE

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

OP: 33

Format:

```
**************
* COMM COPX1 *
**************
```

Function:

Move the length and address fields from the COPX1 entry to the RS.COMMUNICATE.MSG.PTR field located in this program's RS.NUCLEUS, converting them enroute. The origin field is unchanged.

The length is converted from a digit or character length to a bit length. The address is stored as an absolute bit address.

LOAD COMMUNICATE REPLY

```
********
* LDCR *
********
```

OP: 41

Format:

```
**************
* LDCR DADDR *
**************
```

Function:

Move the last 24 bits of information from the RS.REPLY area of the RS.NUCLEUS to the location specified by DADDR.

See 'MAKE PRESENT' for definition of DADDR.

CONVERT

```
********
* CONV *
********
```

OP: 40

Format:

```
*********************
* CONV COPX1 DADDR *
*********************
```

Function:

Convert the operand denoted by COPX1 from a decimal value to an unsigned 24 bit binary value, truncating or zero filling on the left if necessary. Place the result at the location specified by DADDR.

The operand must be either unsigned 4-BIT or unsigned 8-BIT units.

See 'MAKE PRESENT' for definition of DADDR.

MAKE PRESENT

```
********
* MAKP *
********
```

OP: 42

Format:

```
**********************
* MAKP COPX1, DADDR *
**********************
```

Function:

Load the data segment specified by COPX1 and place the base relative address of the data area specified by COPX1 into the 24 bit location specified by DADDR.

DADDR is an unsigned binary value which specifies a digit displacement from the data segment zero base.

The container size of DADDR is DISPB.

HARDWARE MONITOR

```
********
* HMON *
********
```

OP: 43

Format:

```
**************
* HMON OPND1 *
**************
```

Function:

The low order eight bits of the field described by OPND1 are used as the input to the monitor micro-operator described in the following product specifications:

M-Memory Processor	#1913 1747
S-Memory Processor	#2201 6760

The length of the field described by OPND1 must be greater than or equal to eight bits.

ALPHABETIC INDEX: