DOCUMENT TRANSMITTAL

Product Specifications

PAGE 1 OF 1

CONTROL DOC. None

SYSTEM B1800/B1700

PL FILE NO. ______

PROJ. NO. ______ COORD. ______ CHKR. J Nalz TO RECORDS 1-15-79 DISTRIBUTION 1-15-79

DOC PFX	DOCUMENT NUMBER	REV	UNIT NUMBER	DL NO.	PL FILE NO.	ENG. RECORDS ONLY: QTY REV PAGE	COPIES EACH PAGE	SEPIAS	MT REQ'D	CD DK REQ'D		REMARKS
P.S.	2212 5306	C				106	64					

RECEIVED
JAN 17 1979
GENERAL MANAGER
SANTA BARBARA PLANT

DOCUMENT TRANSMITTAL

Burroughs Corporation

COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

B1800/B1700 RPG S-LANGUAGE
P.S. 2212 5306

PRODUCT SPECIFICATION

REV LTR	REVISION ISSUE DATE	APPROVED BY	REVISIONS
C	1/10/79		Changes for MARK VIII.0 1-3 Updated Note. 2-5 Added COPX1 to TEST NUMERIC. 3-37 Updated "Overlap of fields" paragraph. 3-52 Added "The boolean mask has a length of 144 bits" sentence. 3-59 Changed (OPND1 NEQ 9) to (OPND1 NEQ 0) for IND1. 3-60 Updated test A. 3-61 Updates tests B and C. 3-64 Changed TSTZ to TST2. 3-72 Updated "the argument (OPND1)" paragraph. 3-74 Updated "If no match is found" sentence.

TABLE OF CONTENTS

GENERAL 1-1
RELATED DOCUMENTATION 1-1
S-LANGUAGE PROGRAMS 1-1
Program Parameters 1-5
Container Size 1-5
S-Instruction Format 1-6
S-Operators 1-6
Operand (OPND) 1-6
Literal String 1-7
Current Operand Index (COPX) 1-7
In-Line COP Information 1-9
Current Operand Table (COP) 1-11
DISPLACEMENT 1-12
DATA LENGTH 1-12
SUBSCRIPT-OR-INDEX-FLAG 1-12
DATA TYPE 1-12
NUMBER OF SUBSCRIPTS OR INDEXES 1-13
SUBSCRIPT FACTORS 1-13
INDEXING 1-14
TABLE BOUND 1-14
INSTRUCTION SET 2-1
ARITHMETIC 2-1
DATA MOVEMENT 2-2
BRANCHING 2-3
NON-BOOLEAN COMPARISON 2-4
BOOLEAN MANIPULATION 2-5
BOOLEAN COMPARISON 2-6
MISCELLANEOUS 2-7
S-INSTRUCTIONS 3-1
ARITHMETIC OPERANDS AND INSTRUCTIONS 3-1
INCREMENT 3-4
ADD 3-5
DECREMENT 3-6
SUBTRACT 3-7
MULTIPLY 3-8
DIVIDE 3-9
INCREMENT BY ONE 3-10
DECREMENT BY ONE 3-11
DATA MOVEMENT OPERANDS AND INSTRUCTIONS 3-12
MOVE ALPHANUMERIC 3-13
MOVE NUMERIC 3-15
MOVE SPACES 3-17
EDIT INSTRUCTIONS 3-18
EDIT WITH EXPLICIT MASK 3-19

EDIT MICRO-OPERATORS 3-20
MOVE ZEROS 3-28
SCALED MOVE NUMERIC 3-29
MOVE NUMERIC TO FILE INDEX 3-30
MOVE NUMERIC TO OUTPUT SWITCH 3-31
MOVE ARRAY 3-32
BRANCHING OPERANDS AND INSTRUCTIONS 3-33
BRANCH UNCONDITIONALLY 3-34
ENTER 3-35
EXIT 3-36
NON-BOOLEAN COMPARISON AND INSTRUCTIONS 3-37
COMPARE ALPHANUMERIC 3-38
COMPARE NUMERIC 3-39
COMPARE FOR ZEROS 3-40
COMPARE FOR SPACES 3-41
COMPARE ZERO TO OUTPUT SWITCH 3-42
COMPARE ZERO TO FILE INDEX 3-43
COMPARE NUMERIC TO OUTPUT SWITCH 3-44
COMPARE NUMERIC TO FILE INDEX 3-45
COMPARE ALPHA, MARK RESULT 3-46
COMPARE NUMERIC, MARK RESULT 3-47
COMPARE FOR SPACES, MARK RESULT 3-48
COMPARE FOR ZEROS, MARK RESULT 3-49
BOOLEAN MANIPULATION 3-50
SET ONE BOOLEAN 3-50
CLEAR ONE BOOLEAN 3-51
CLEAR BOOLEANS USING MASK 3-52
CLEAR TWO BOOLEANS 3-53
CLEAR THREE BOOLEANS 3-54
SET TWO BOOLEANS 3-55
SET THREE BOOLEANS 3-56
BIT ON 3-57
BIT OFF 3-58
TEST BIT 3-59
TEST NUMERIC 3-60
BOOLEAN COMPARISON 3-62
BRANCH IF BOOLEAN FALSE 3-62
BRANCH IF BOOLEAN TRUE 3-63
TEST 2 BOOLEANS 3-64
TEST 3 BOOLEANS 3-65
MISCELLANEOUS INSTRUCTIONS 3-66
CONVERT TO BINARY 3-66
LOAD COMMUNICATE REPLY 3-67
MAKE PRESENT 3-68
COMMUNICATE 3-69
HARDWARE MONITOR 3-70
BEGINNING OF JOB 3-71
UNORDERED TABLE SEARCH 3-72
ORDERED TABLE SEARCH 3-73
CONVERT FIELD 3-76

GENERAL

The B1800/B1700 RPG S-Language provides the virtual machine interface between the code generated by the RPG compiler and the RPG interpreter. This specification describes the format of RPG S-instructions and then explains each operator as a member of one of the following classes:

ARITHMETIC
DATA MOVEMENT
BRANCHING
NON-BOOLEAN COMPARISON
BOOLEAN MANIPULATION
BOOLEAN COMPARISON
MISCELLANEOUS

RELATED DOCUMENTATION

NAME	NUMBER
B1800/B1700 Software Operational Guide	1068731
B1800/B1700 COBOL Reference Manual	1057197
B1800/B1700 RPG Reference Manual	1057189
B1800/B1700 RPG Compiler Specification	P.S. 2205 1155

S-LANGUAGE PROGRAMS

All RPG S-language programs have a base register and a limit register associated with them. The area between the base register and the limit register 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 B1800/B1700 operating system.

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

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

A typical RPG program layout in memory is as follows:

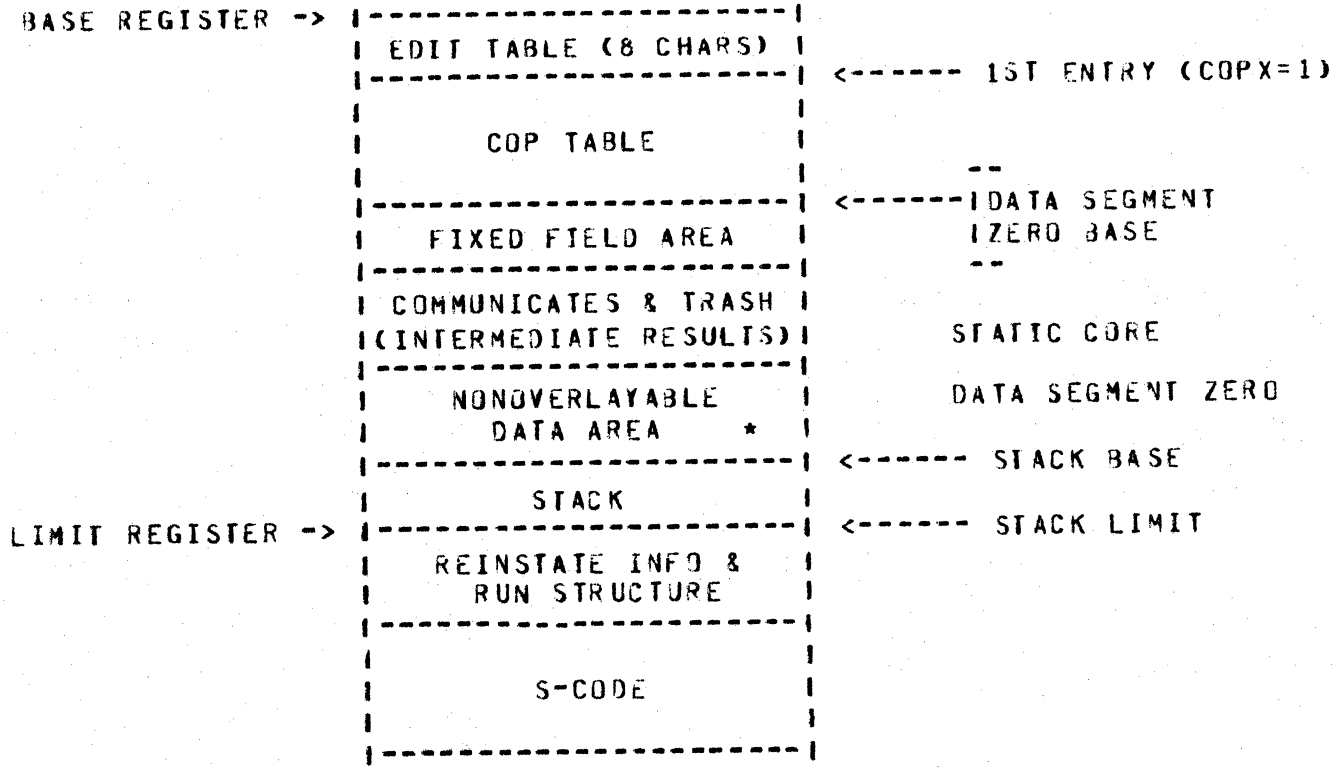


Figure 1-1: RPG Program Layout

* THIS AREA IS ACCESSED BY BOTH THE USER AND THE COMPILER FOR DEFINED DATA.

*		SIZE (BITS)
0 - 255	256 BOOLEAN INDICATORS	1 EA
256	FX	8
264	SK	12
276	SP	12
288	FN	8
296	OS	4

* Address relative to Data Segment Zero Base

Figure 1-2: Fixed Field Area for Programs Prior to 8.0.

Note: Variables peculiar to RPG operation are contained in the Fixed Field Area. The 256 Boolean Indicators set conditions for execution of various parts of the program. FX [File Index] conditions the file to be processed. SK controls printer skipping, SP controls printer spacing. FN designates a file by number and OS [Output Switch] denotes what I/O operation is to be performed.

*		SIZE (BITS)
0 - 511	512 BOOLEAN INDICATORS	1 EA
512	FX	12
524	SK	12
536	SP	12
548	FN	8
556	OS	4

* Address relative to Data Segment Zero Base

Figure 1-3: Fixed Field Area for Programs 8.0 and After.

Note: The total size of the Fixed Field Area for programs prior to 8.0, 8.0, or after 8.0 may vary depending on the actual user program. Note that 256 Boolean Indicators means the container size for a boolean address index is 8-bits wide, while 512 Boolean Indicators means a container size of 9 bits.

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.

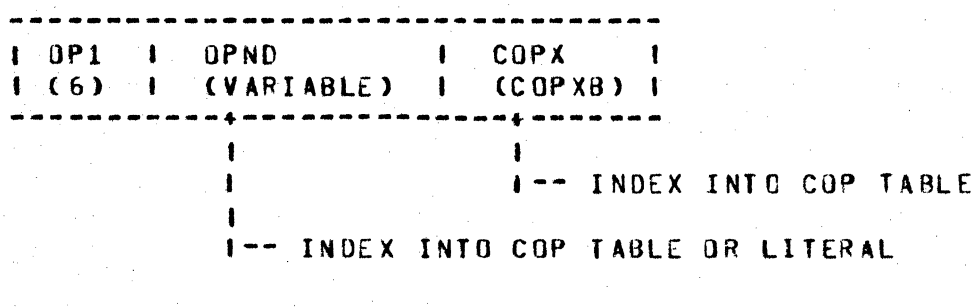
BDISPBI (5)	Branch displacement container size + 1
DSEGZ (24)	Base of data segment zero
STACK-POINTER (24)	Base address of stack
STACK-SIZE (5)	Size of the stack
COP-BASE (24)	Base address of COP table
COPB (12)	COP entry container size
DISPB (5)	Data displacement container size
LENB (5)	Data length 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 addresses (0-31).

S-Instruction Format

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

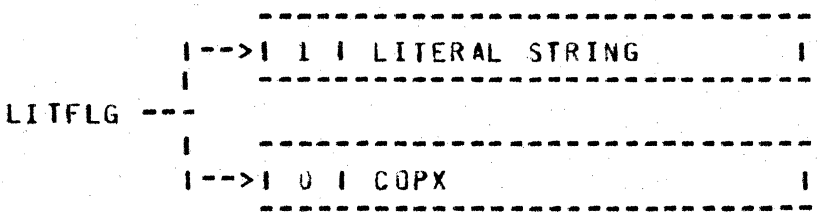


S-Operators

All S-operators are encoded in six-bit fields denoted as OP1.

Operand (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:



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 (2)	LLGTH1 (3)	LLGTH2 (8)	LSYMB (VARIABLE)

Note: LLGTH2 present if LLGTH1 equals zero.

LTYPE

00 = Unsigned 4-bit
01 = Unsigned 8-bit
10 = Signed 4-bit (sign is Most-Significant-Digit)
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 LLGHT2 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:

ADDRESS DISPL. (DISPB)	DATA LENGTH (LENB)	SUBSCRIPT OR INDEX FLAG (1)	DATA TYPE (2)	UNUSED (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 data type) 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

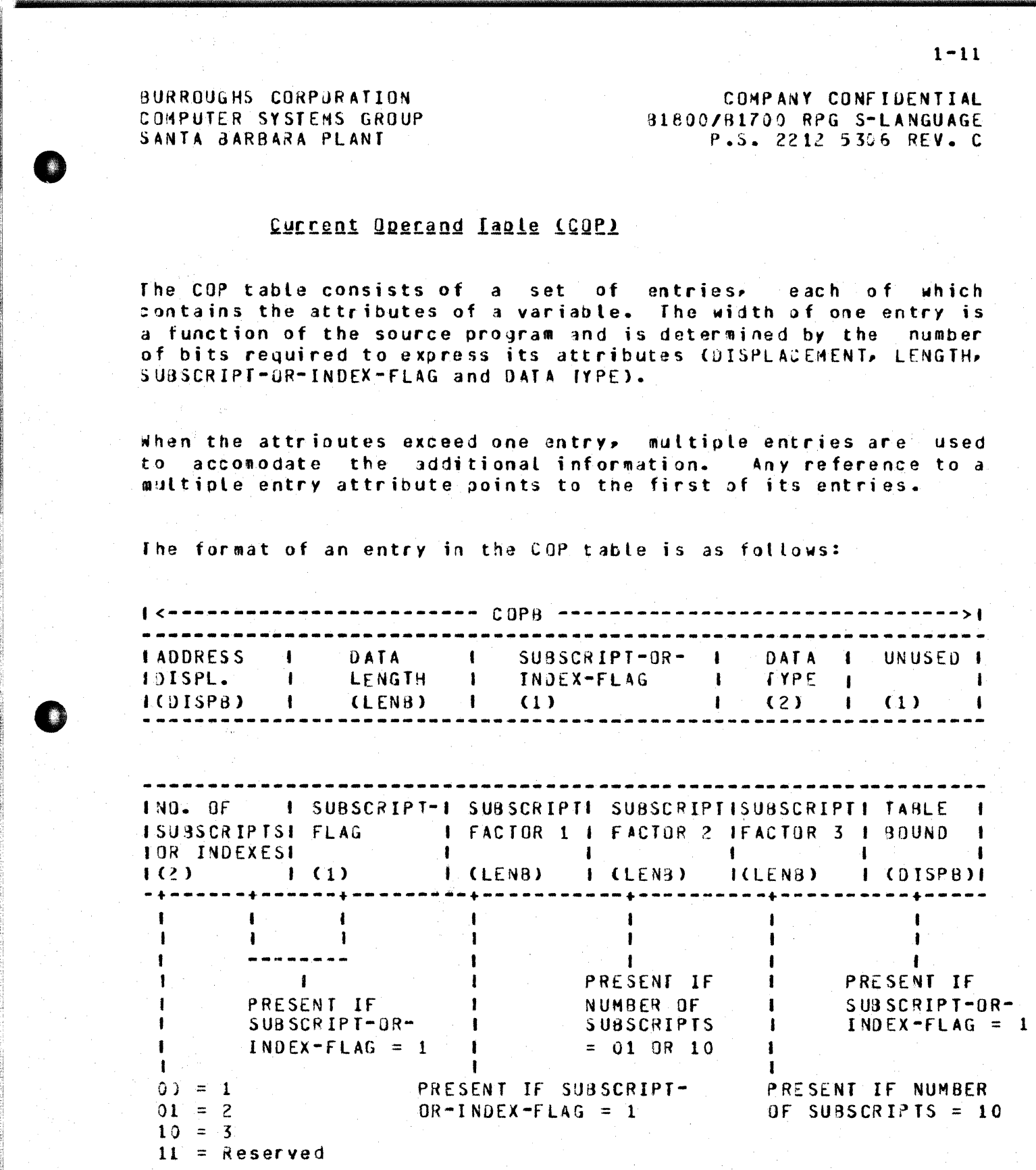
3. COPX1, COPX2, and COPX3 may be in-line entries but must not be subscripted or indexed.

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 (DISPLACEMENT, LENGTH, SUBSCRIPT-OR-INDEX-FLAG and DATA TYPE).

When the attributes exceed one entry, multiple entries are used to accomodate 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:



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 or 16,383 4-bit units.

SUBSCRIPT-OR-INDEX-FLAG

The SUBSCRIPT-OR-INDEX-FLAG bit is true to indicate subscripting or indexing; otherwise, it is false. When true, the next entry(s) contain the necessary subscripting or indexing information.

Note: None of the subscripting/indexing information (all entries following the Data Type) is present unless the SUBSCRIPT-OR-INDEX-FLAG equals one.

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 Most-Significant-Digit)
11 = Signed 8-bit (sign over Most-Significant-Digit)

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 the SUBSCRIPT-FLAG field indicates the appropriate operation: indexing or subscripting.

0 = Indexing
1 = Subscripting

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 a digit value indicating the displacement between elements of the table. The value one is subtracted from the subscript 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 in-line descriptor in the S-instructions.

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.

INSTRUCTION SET

ARITHMETIC

NAME	MNEMONIC	OP	ARGUMENTS
INCREMENT	INC	05	OPND1, COPX1
ADD	ADD	10	OPND1, COPX1, COPX2
DECREMENT	DEC	11	OPND1, COPX1
SUBTRACT	SUB	12	OPND1, OPND2, COPX1
MULTIPLY	MULT	13	OPND1, COPX1, COPX2
DIVIDE	DIV	14	OPND1, COPX1, COPX2
INCREMENT BY ONE	INC1	15	COPX1
DECREMENT BY ONE	DEC1	16	COPX1

DATA MOVEMENT

NAME	MNEMONIC	OP	ARGUMENTS
MOVE ALPHANUMERIC	MVA	03	OPND1, COPX1
MOVE SPACES	MVS	17	COPX1
MOVE NUMERIC	MVN	04	OPND1, COPX1
EDIT WITH EXPLICIT MASK	EDTE	20	OPND1, COPX1, MASK
MOVE ZEROS	MVZ	21	COPX1
SCALED MOVE NUMERIC	SMVN	23	OPND1, COPX1, V, SCL
MOVE NUMERIC TO FILE INDEX	MNFX	30	OPND1
MOVE NUMERIC TO OUTPUT SWITCH	MNOS	31	OPND1
MOVE ARRAY	MVAR	47	IND1, OPND1, COPX1, COPX2, COPX3

BRANCHING

NAME	MNEMONIC	OP	ARGUMENTS
BRANCH UNCONDITIONALLY	BUN	06	BADDR
ENTER	NTR	18	
EXIT	XIT	19	BADDR

NON-BOOLEAN COMPARISON

NAME	MNEMONIC	OP	ARGUMENTS
COMPARE ALPHANUMERIC	CMPA	07	OPND1, COPX1, R, BADDR
COMPARE NUMERIC	CMPN	08	OPND1, COPX1, R, BADDR
COMPARE FOR ZEROS	CMPZ	22	COPX1, R, BADDR
COMPARE FOR SPACES	CMPS	25	COPX1, R, BADDR
COMPARE ZERO TO OUTPUT SWITCH	CZOS	26	R, BADDR
COMPARE ZERO TO FILE INDEX	CZFX	27	R, BADDR
COMPARE NUMERIC TO OUTPUT SWITCH	CNOS	28	OPND1, R, BADDR
COMPARE NUMERIC TO FILE INDEX	CNFX	29	OPND1, R, BADDR
COMPARE ALPHA, MARK RESULT	CAMR	43	OPND1, COPX1, R, BI
COMPARE NUMERIC, MARK RESULT	CNMR	44	OPND1, COPX1, R, BI
COMPARE FOR SPACES, MARK RESULT	CSMR	45	COPX1, R, BI
COMPARE FOR ZEROS, MARK RESULT	CZMR	46	COPX1, R, BI

BOOLEAN MANIPULATION

NAME	MNEMONIC	OP	ARGUMENTS
SET ONE BOOLEAN	SET1	32	BI
CLEAR ONE BOOLEAN	CLR1	33	BI
CLEAR BOOLEANS USING MASK	CBUM	36	MASK
CLEAR TWO BOOLEANS	CLR2	37	BI1, BI2
CLEAR THREE BOOLEANS	CLR3	38	BI1, BI2, BI3
SET TWO BOOLEANS	SET2	39	BI1, BI2
SET THREE BOOLEANS	SET3	40	BI1, BI2, BI3
BIT ON	BTON	48	OPND1, COPX1
BIT OFF	BTOF	49	OPND1, COPX1
TEST BIT	TSTB	50	OPND1, COPX1, IND1, IND2, IND3
TEST NUMERIC	TSTN	55	BI1, BI2, BI3, COPX1

BOOLEAN COMPARISON

NAME	MNEMONIC	OP	ARGUMENTS
BRANCH IF BOOLEAN FALSE	BIBF	09	BI, BADDR
BRANCH IF BOOLEAN TRUE	BIBT	34	BI, BADDR
TEST 2 BOOLEANS	TST2	41	BTV1, BI1, BTV2, BI2, BADDR
TEST 3 BOOLEANS	TST3	42	BTV1, BI1, BTV2, BI2, BTV3, BI3, BADDR

MISCELLANEOUS

NAME	MNEMONIC	OP	ARGUMENTS
CONVERT TO BINARY	CONV	00	COPX1, BADDR
LOAD COMMUNICATE REPLY	LDCR	01	BADDR
COMMUNICATE	COMM	24	COPX1
MAKE PRESENT	MAKP	02	COPX1, BADDR
HARDWARE MONITOR	HMON	35	OPND1
BEGINNING OF JOB	BOJ	51	COPX1
UNORDERED TABLE SEARCH	UTSR	52	OPND1, COPX1, COPX2, COPX3, BI
ORDERED TABLE SEARCH	OTSR	53	OPND1, COPX1, COPX2, COPX3, IND1, KEYS, BI1, BI2
CONVERT FIELD	CNV	54	COPX1, COPX2

S-INSTRUCTIONS

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 Most-Significant-Digit)
4. Signed 8-bit (sign over Most-Signficant-Digit)

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.

The 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.

The 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 for EBCDIC.

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.

INCREMENT

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

OP: 05

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.

ADD

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

OP: 10

Format:

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

Function:

Algebraically add an addend denoted by OPND1 to an augend denoted by COP(1 and store the sum in the COPX1 and store that sum in the field denoted by COPX2.

DECREMENT

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

OP: 11

Format:

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

Function:

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

SUBIBACI

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

OP: 12

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.

MULTIPLY

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

OP: 13

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: 14

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 length 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.

INCREMENT BY ONE

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

OP: 15

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: 16

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 Most-Significant-Digit)
4. Signed 8-bit (sign over Most-Significant-Digit)

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

See ARITHMETIC OPERANDS AND INSTRUCTIONS for a description of the 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: 03

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 or 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 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 the destination type is 8-bit; otherwise, it is filled in on the right with nulls (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 NUMERIC

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

OP: 04

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, 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.

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

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

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 SPACES

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

OP: 17

Format:

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

Function:

Fill the destination field denoted by COPX1 with spaces (0100 0000).

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

EDIT INSTRUCTIONS

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 the exhaustion of either the source or destination fields.

EDIT WITH EXPLICIT MASK

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

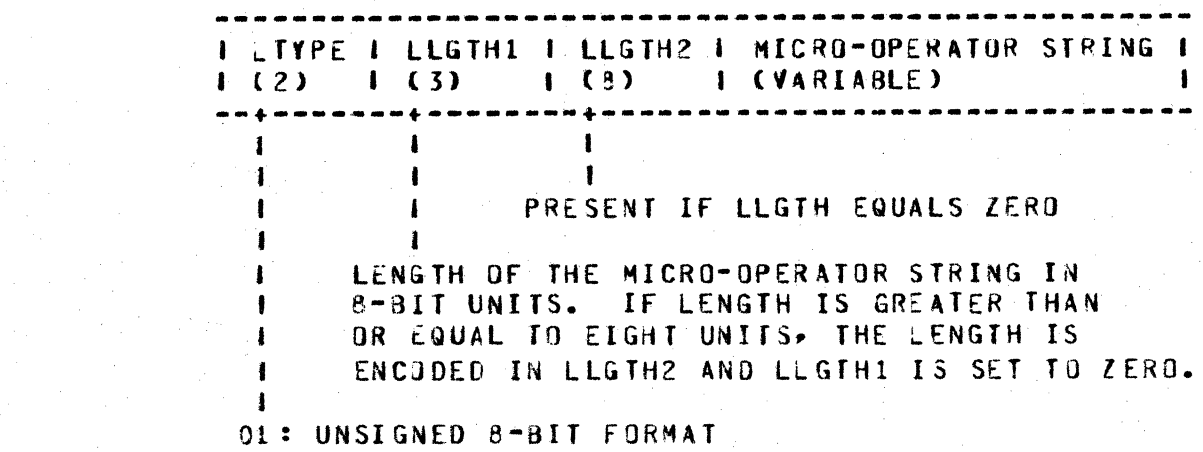
OP: 20

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 within the MASK field immediately following COPX1. The format of the MASK is the same as the literal and is as follows:



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
1010 0100	ISP	INSERT SUPPRESS PREVIOUS
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 table below 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 he 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" equals "0", indicating non-check protect mode, move the table entry containing the 8-bit code for blank to the destination field. If "P" equals "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)
```

FULL SUPPRESS

If "P" equals "0", indicating non-check protect mode, move the table entry containing the 8-bit code for blank to the destination field. If "P" equals "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
*	S=0		MOVE TABLE ENTRY 7 (BLANK)
	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".

INSERT SUPPRESS PREVIOUS

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

Z=0	P=0	MOVE TABLE ENTRY 7 (BLANK)
Z=0	P=1	MOVE TABLE ENTRY 2 (ASTERISK)
Z=1		MOVE PREVIOUS DESTINATION FIELD CHARACTER, MOVE IN LINE MASK CONSTANT INTO WHERE THE PREVIOUS DESTINATION CHARACTER CAME FROM, THEN UPDATE DESTINATION FIELD POINTER

MOVE ZEROS

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

OP: 21

Format:

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

Function:

Fill the destination field denoted by COPX1 with zeros of the appropriate type (1111 0000 if 8-bit or 0000 if 4-bit).

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

SCALED MOVE NUMERIC

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

OP: 23

Format:

```
*********************************
* SMVN OPND1, COPZ1, V, SCL *
*********************************
```

Function:

If V equals zero, 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 NUMERIC IO FILE INDEX

```
********
* MNFX *
********
```

OP: 30

Format:

```
***************
* MNFX OPND1 *
***************
```

Function:

Performs a normal MOVE NUMERIC after building the destination descriptor (OPND1) for the implicitly declared file index field.

BURROUGHS CORPORATION
COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

MOVE NUMERIC TO OUTPUT SWITCH

```
********
* MNOS *
********
```

OP: 31

Format:

```
***************
* MNOS OPND1 *
***************
```

Function:

Performs a normal MOVE NUMERIC after building the destination descriptor (OPND1) for the implicitly declared output switch field.

MOVE ARRAY

```
********
* MVAR *
********
```

OP: 47

Format:

```
*******************************************
* MVAR IND1, OPND1, COPX1, COPX2, COPX3 *
*******************************************
```

Function:

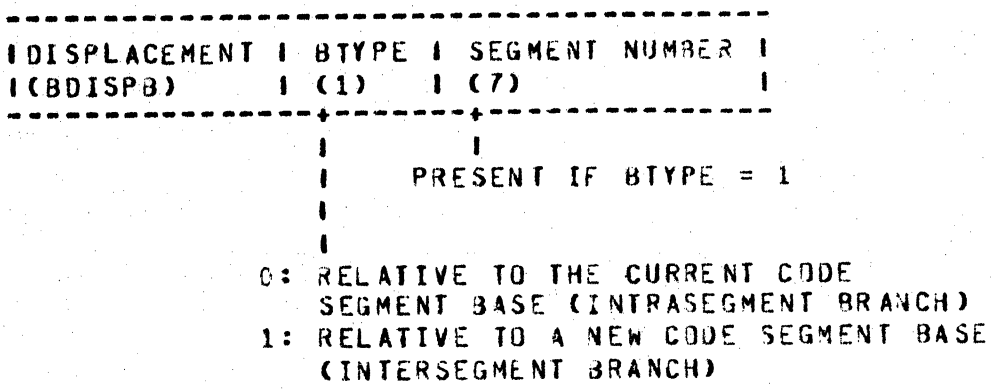
First adjusts the field lengths of the source (OPND1) and destination (COPX2) descriptors so that they each describe their arrays from the actual element addressed to the end of the array and by adjusting their field lengths so that no "Destination Longer" fill will occur.

If the source field is not an array (IND1), then COPX1 is not present. If the destination field is not an array (IND1), then COPX3 is not present. These entries describe their entire respective arrays.

All fields are assumed to be "unsigned 8-bit".

BRANCHING OPERANDS AND INSTRUCTIONS

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



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 UNCONDIIIONALLY

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

OP: 06

Format:

```
*************
* BJN BADDR *
*************
```

Function:

Obtain the next instruction from the location specified by BADDR.

ENTER

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

OP: 18

Format:

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

Function:

Places an entry on the top of the program pointer stack, bumps up the stack pointer to point to the next available stack entry and branches to (or enters at) the location specified by the BADDR entry.

The stack entry consists of the displacement from the current code segment base to the S-instruction following the perform (subroutine call) and the current segment number.

EXII

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

OP: 19

Format:

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

Function:

Performs by removing and saving off the current entry on the top of the program pointer stack, downdating the stack pointer, and obtaining the address of the next S-instruction from the removed stack entry.

NON-BOOLEAN COMPARISON 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; i.e., for CZDS with R and BADDR, the assumed field "A" is a field comprised of zeros (0000).

COMPARE ALPHANUMERIC

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

OP: 07

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 8-bit).

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

The 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 8-bit) before being compared. This modification is not permanent and is done so that the sign will not affect the result of an alphanumeric comparison.

COMPARE NUMERIC

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

OP: 08

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 the 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: 22

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: 25

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 for 8-bit).

The comparison is performed left to right.

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

COMPARE ZERO TO OUTPUT SWITCH

```
********
* CZOS *
********
```

OP: 26

Format:

```
******************
* CZOS R, BADDR *
******************
```

Function:

Performs a normal COMPARE FOR ZEROS after building the descriptor for the implicitly declared output switch field.

COMPARE ZERO IO FILE INDEX

```
********
* CZFX *
********
```

OP: 27

Format:

```
******************
* CZFX R, BADDR *
******************
```

Function:

Performs a normal COMPARE FOR ZEROS after building the descriptor for the implicitly declared file index field.

COMPARE NUMERIC IO OUTPUT SWITCH

```
********
* CNOS *
********
```

OP: 28

Format:

```
**************************
* CNOS OPND1, R, BADDR *
**************************
```

Function:

Performs a normal COMPARE NUMERIC after building the descriptor for the implicitly declared output switch field.

COMPARE NUMERIC TO FILE INDEX

```
********
* CNFX *
********
```

OP: 29

Format:

```
**************************
* CNFX OPND1, F, BADDR *
**************************
```

Function:

Performs a normal COMPARE NUMERIC after building the descriptor for the implicitly declared file index field.

COMPARE ALPHA, MARK RESULT

```
********
* CAMR *
********
```

OP: 43

Format:

```
*******************************
* CAMR OPND1, COPX1, R, BI *
*******************************
```

Function:

Performs a normal COMPARE ALPHA, except that if the relation (R) is met, the boolean indicator specified by the boolean index (BI) is set to zero.

If the relation is not met, the boolean is set to one.

This S-instruction thus marks the result of the comparison, rather than branching.

COMPARE NUMERIC, MARK RESULT

```
********
* CNMR *
********
```

OP: 44

Format:

```
*******************************
* CNMR OPND1, COPX1, R, BI *
*******************************
```

Function:

Performs a normal COMPARE NUMERIC, except that if the relation (R) is met, the boolean indicator specified by the boolean index (BI) is set to zero.

If the relation is not met, the boolean is set to one.

This S-instruction thus marks the result of the comparison, rather than branching.

COMPARE FOR SPACES, MARK RESULT

```
********
* CSMR *
********
```

OP: 45

Format:

```
************************
* CSMR COPX1, R, BI *
************************
```

Function:

Performs a normal COMPARE FOR SPACES, except that if the relation (R) is met, the boolean indicator specified by the boolean index (BI) is set to zero.

If the relation is not met, the boolean is set to one.

This S-instruction thus marks the result of the comparison, rather than branching.

COMPARE FOR ZEROS, MARK RESULT

```
********
* CZMR *
********
```

OP: 46

Format:

```
************************
* CZMR COPX1, R, BI *
************************
```

Function:

Performs a normal COMPARE FOR ZEROS on COPX1, except that if the relation (R) is met, the boolean indicator specified by the boolean index (BI) is set to zero.

If the relation is not met, the boolean is set to one.

This S-instruction thus marks the result of the comparison, rather than branching.

BOOLEAN MANIPULATION

SET ONE BOOLEAN

```
********
* SET1 *
********
```

OP: 32

Format:

```
************
* SET1 BI *
************
```

Function:

Sets the boolean indicator, specified by the boolean index (BI) into the reserved area, to a one.

CLEAR ONE BOOLEAN

```
********
* CLR1 *
********
```

OP: 33

Format:

```
************
* CLR1 BI *
************
```

Function:

Clears the boolean indicator, specified by the boolean index (BI) into the reserved boolean area, to a zero.

CLEAR BOOLEANS USING MASK

```
********
* CBUM *
********
```

OP: 36

Format:

```
**********************
* CBUM BOOLEAN.MASK *
**********************
```

Functions:

Clears the boolean indicators, in the reserved boolean area, which are matched with a zero bit in the boolean mask. It is thus a bit "AND" of the booleans with the boolean mask. The boolean mask has a length of 144 bits.

CLEAR TWO BOOLEANS

```
********
* CLRZ *
********
```

OP: 37

Format:

```
******************
* CLRZ BI1, BI2 *
******************
```

Function:

Clears the boolean indicators, specified by the boolean indexes (BI1 and BI2) into the reserved boolean area, to zero.

CLEAR THREE BOOLEANS

```
********
* CLR3 *
********
```

OP: 38

Format:

```
************************
* CLR3 BI1, BI2, BI3 *
************************
```

Function:

Clears the boolean indicators, specified by the boolean indexes (BI1, BI2 and BI3) into the reserved boolean area, to zero.

SET TWO BOOLEANS

```
********
* SET2 *
********
```

OP: 39

Format:

```
******************
* SET2 BI1, BI2 *
******************
```

Function:

Sets the boolean indicators, specified by the boolean indexes (BI1 and BI2) into the reserved boolean area, to one.

SET THREE BOOLEANS

```
********
* SET3 *
********
```

OP: 40

Format:

```
************************
* SET3 BI1, BI2, BI3 *
************************
```

Function:

Sets the boolean indicators, specified by the boolean indexes (BI1, BI2 and BI3) into the reserved boolean area, to one.

BIT ON

```
********
* BTON *
********
```

OP: 48

Format:

```
**********************
* BTON OPND1, COPX1 *
**********************
```

Function:

Takes the two operands (OPND1 and COPX1) and OR's them together with the result placed in the location denoted by COPX1.

Both OPND1 and COPX1 are guaranteed to be 8-bits wide.

BII OEE

```
********
* BTOF *
********
```

OP: 49

Format:

```
**********************
* BTOF OPND1, COPX1 *
**********************
```

Function:

Takes the OPND1 operand and complements it before performing an "AND" operation with the COPX1 operand. The result is placed in the location denoted by COPX1.

Both OPND1 and COPX1 are guaranteed to be 8-bits wide.

IESI BII

```
********
* TSTB *
********
```

OP: 50

Format:

```
*****************************************
* TSTB OPND1, COPX1, IND1, IND2, IND3 *
*****************************************
```

Function:

Both OPND1 and COPX1 are guaranteed to be 8-bits wide. The routine computes (OPND1 "AND" COPX1) and then sets the indicators according to the following rules:

```
IND1 := (OPND1 NEQ 0) AND ((OPND1 AND COPX1) = 0)

IND2 := (OPND1 NEQ 0) AND ((OPND1 AND COPX1) NEQ 0)
        AND ((OPND1 AND COPX1) NEQ OPND1)

IND3 := (OPND1 NEQ 0) AND ((OPND1 AND COPX1) = OPND1)
```

TESI NUMERIC

```
********
* TSTN *
********
```

OP: 55

Format:

```
*******************************
* TSTN BI1, BI2, BI3, COPX1 *
*******************************
```

Function:

Will clear all three indicators pointed to by the BI's.

If the value of the first BI (the actual value of the index and not the value of the indicator indexed to) is zero, skip test A. If BI2 is zero, skip test B. If BI3 is zero, skip test C.

The three tests are mutually exclusive and as soon as one is found to be true the operation will end.

The three tests are as follows:

A) The field defined by the COPX must contain all numeric characters where a numeric character is defined as hexadecimal Fd, where d=0-9, except for the right most EBCDIC character which may be any of the following: Fd, Cd, Dd, 4E, 60 (d as defined above). If the test is satisfied, set the boolean indicator indexed to by BI1. If tne test fails, go to test B.

B) The field defined by the COPX must have at least one leading blank (hexadecimal 40) followed by one or more EBCDIC character(s) where EBCDIC is defined in A) above. The field will have at least two characters, i.e., one blank and one EBCDIC character. The field must have only leading blanks followed by EBCDIC characters. If the test is satisfied, set the boolean indicator indexed to by BI2. If the test fails, go to test C.

C) The field defined by COPX must consist of all blanks (hexidecimal 40). If the test is satisfied, set the boolean indicator indexed to by BI3. If the test fails, go directly to the next S-op in line.

BOOLEAN COMPARISON

BRANCH IF BOOLEAN FALSE

```
********
* BIBF *
********
```

OP: 09

Format:

```
********************
* BIBF BI, BADDR *
********************
```

Function:

Tests the boolean indicator specified by the boolean index (BI) and then branches to the S-instruction specified by BADDR if the boolean is false; i.e., zero.

BRANCH IF BOOLEAN TRUE

```
********
* BIBT *
********
```

OP: 34

Format:

```
********************
* BIBT BI, BADDR *
********************
```

Function:

Tests the boolean indicator specified by the boolean index (BI) and then branches to the S-instruction specified by BADDR if the boolean is true; i.e., one.

IESI 2 BOOLEANS

```
********
* TST2 *
********
```

OP: 41

Format:

```
****************************************
* TST2 BTV1, BI1, BTV2, BI2, BADDR *
****************************************
```

Function:

Compares each boolean test value (BTV1 and BTV2) to the boolean indicator pointed to by the corresponding boolean index (BI1 and BI2).

If they compare equal, then a branch is made to the location specified by BADDR.

If they do not compare equal, then the next are compared in like manner.

If none compare equal, then the next instruction pointer is skipped until the next in-line S-instruction occurs.

IESI 3 BOOLEANS

```
********
* TST3 *
********
```

OP: 42

Format:

```
*************************************************
* TST3 BTV1, BI1, BTV2, BI2, BTV3, BI3, BADDR *
*************************************************
```

Function:

Compares each boolean test value (BTV1, BTV2 and BTV3) to the boolean indicator pointed to by the corresponding boolean index (BI1, BI2 and BI3).

If they compare equal, then a branch is made to the location specified by BADDR.

If they do not compare equal, then the next are compared in like manner.

If none compare equal, then the next instruction pointer is skipped until the next in-line S-instruction.

MISCELLANEOUS INSTRUCTIONS

CONVERT TO BINARY

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

OP: 00

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.

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

LOAD COMMUNICATE REPLY

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

OP: 01

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.

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

MAKE PRESENT

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

OP: 32

Format:

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

Function:

Loads the data segment specified by COPX1 and places 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.

COMMUNICATE

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

OP: 24

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.

HARDWARE MONITOR

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

OP: 35

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.

BEGINNING OF JOB (BOJ)

```
*******
* BOJ *
*******
```

OP: S1

Format:

```
*************
* BOJ COPX1 *
*************
```

Function:

This S-instruction is called at BOJ time to do anything that has to be done. It is thus used only once per RPG execution. It does the following:

- Gets the middle 32 bits from the RS.SWITCHES field in the run structure nucleus and writes them into the 32-bit field specified by COPX1.
- Writes the low order bit of each 4-bit switch into the 8 "U" indicators (U1 through U8) at boolean 131 through 138 (zero origin).

UNORDERED TABLE SEARCH

```
********
* UTSR *
********
```

OP: 52

Format:

```
*****************************************
* UTSR OPND1, COPX1, COPX2, COPX3, BI *
*****************************************
```

Function:

Will take the argument (OPND1) and proceed to search serially through an unordered table, whose first element (and hence the beginning address of the table) is described by an index (COPX1 which is usually subscripted), for a match (compare "equal"). The search will begin with an element (described by COPX2; 4 digit-unsigned field) within the table and will continue until either a match is found or when the last element (described by COPX3; also subscripted in the table is reached.

If a match is found, a flag will be set in the boolean indicator (as described by the boolean index, BI) and COPX2 will contain the index value of that element in the table where the match was found.

If no match is found, no flag will be set in the boolean indicator and COPX2 will contain an index value greater than the index value of the last table element (COPX3).

ORDERED TABLE SEARCH

```
********
* OTSR *
********
```

OP: 53

Format:

```
*************************************************************
* OTSR OPND1, COPX1, COPX2, COPX3, IND1, KEYS, BI1, BI2 *
*************************************************************
```

Function:

Will take the argument (OPND1) and proceed to search through an ordered table (non-ascending or non-descending order) whose first element (and hence the beginning address of the table) is described by an index (COPX1), for a match. The search will begin with an element (described by COPX2) within the table and will continue until either a match (or an element which most nearly satisfies the relational requirements) is found or when the last element (described by COPX3) is reached.

A flag (IND1) will be set if the table is nondescending.

The relational requirements ("=", "<", ">", or a combination between two) as described by KEYS will determine how the match between the argument and the table elements is to be made.

If a match (or an element which most nearly satisfies the relational requirements) is found, two flags will be set in the boolean indicators as addressed by BI1 and BI2. COPX2 will contain the index value of that element in the table where the match was found.

If no match is found, the flags will be reset in the boolean indicators and COPX2 will be set to 1.

Some comments regarding UTSR:

Before going into the description of the algorithm, three items should be noted:

1. This S-instruction imitates the RPGII's lookup routine.
2. This S-instruction does not pretend to be mathematically sound.
3. The user should be aware of this S-instruction's idiosyncracies and program accordingly.

It should be noted also that the table under inspection may be:

1. Declared ordered and non-ascending or non-descending, but can be really unordered.
2. Declared ordered and non-ascending or non-descending and is really ordered.
3. Declared unordered, in which case, the compiler will always call the Unordered Table Search (UTSR) S-instruction (the user should be aware that under UTSR the only relational requirement for a match is "=").

The algorithm:

1. Save the index of the first table element to be examined in "F".
2. Save the index of the last table element to be examined in "L".
3. Turn both booleans associated with the S-instruction initially off.

4. Set a REVERSED RELATION flag if:

   a. search for LSS or LEQ on an ascending table, or

   b. search for GTR or GEQ on a descending table.

5. If F>L then go to step 8.

6. Compare TABLE (F) to ARGUMENT. Go to step 14 if:

Table Type	Search Relation	Table(F) : Argument
Ascending	LSS	GTR or EQL
Ascending	LEQ	GTR
Ascending	EQL	EQL
Ascending	GEQ	GEQ
Ascending	GTR	GTR
Descending	LSS	LSS
Descending	LEQ	LEQ
Descending	EQL	EQL
Descending	GEQ	LSS
Descending	GTR	LSS or EQL

7. Add 1 to F and go to step 5.

8. If the REVERSED RELATION flag is off - go to step 13.

9. Subtract 1 from F.

10. If F is less than the index of the first element to search, move 1 to the first index and exit.

11. Set the first boolean if the last compare showed EQL, the REVERSED RELATION flag is on and the search was for LSS or GTR, or the REVERSED RELATION flag was off and the search was for EQL, LEQ, or GEQ. Otherwise, set the second boolean.

12. Set the first index to F and exit.

13. Move 1 to the first index and exit.

14. If the REVERSED RELATION flag is on, go to Step 9; otherwise, go to step 11.

CONVERT FIELD

```
*******
* CNV *
*******
```

OP: 54

Format:

```
**********************
* CNV COPX1, COPX2 *
**********************
```

Function:

Will convert a source (COPX1) into a result to be placed at a destination (COPX2).

The form of the data types as described by COPX1 and COPX2 for source and destination must conform to one of the following patterns since the form is not checked by the S-instruction itself.

COPX1 (SOURCE)	COPX2 (DESTINATION)
DECIMAL DATA	BINARY DATA
4 DIGITS	16 BITS
9 DIGITS	32 BITS
BINARY DATA	DECIMAL DATA
16 BITS	4 DIGITS
32 BITS	9 DIGITS

NOTE: Decimal data is in standard signed numeric form while binary data is in 2's complement form.

INDEX

ADD 3-5
ARITHMETIC 2-1
ARITHMETIC OPERANDS AND INSTRUCTIONS 3-1
BEGINNING OF JOB 3-71
BIT OFF 3-58
BIT ON 3-57
BOOLEAN COMPARISON 2-6, 3-62
BOOLEAN MANIPULATION 2-5, 3-50
BRANCH IF BOOLEAN FALSE 3-62
BRANCH IF BOOLEAN TRUE 3-63
BRANCH UNCONDITIONALLY 3-34
BRANCHING 2-3
BRANCHING OPERANDS AND INSTRUCTIONS 3-33
CLEAR BOOLEANS USING MASK 3-52
CLEAR ONE BOOLEAN 3-51
CLEAR THREE BOOLEANS 3-54
CLEAR TWO BOOLEANS 3-53
COMMUNICATE 3-69
COMPARE ALPHA, MARK RESULT 3-46
COMPARE ALPHANUMERIC 3-38
COMPARE FOR SPACES 3-41
COMPARE FOR SPACES, MARK RESULT 3-48
COMPARE FOR ZEROS 3-40
COMPARE FOR ZEROS, MARK RESULT 3-49
COMPARE NUMERIC 3-39
COMPARE NUMERIC TO FILE INDEX 3-45
COMPARE NUMERIC TO OUTPUT SWITCH 3-44
COMPARE NUMERIC, MARK RESULT 3-47
COMPARE ZERO TO FILE INDEX 3-43
COMPARE ZERO TO OUTPUT SWITCH 3-42
Container Size 1-5
CONVERT FIELD 3-76
CONVERT TO BINARY 3-66
Current Operand Index (COPX) 1-7
Current Operand Table (COP) 1-11
DATA LENGTH 1-12
DATA MOVEMENT 2-2
DATA MOVEMENT OPERANDS AND INSTRUCTIONS 3-12
DATA TYPE 1-12
DECREMENT 3-6
DECREMENT BY ONE 3-11
DISPLACEMENT 1-12
DIVIDE 3-9
EDIT INSTRUCTIONS 3-18

EDIT MICRO-OPERATORS 3-20
EDIT WITH EXPLICIT MASK 3-19
ENTER 3-35
EXIT 3-36
Figure 1-1: RPG Program Layout 1-2
Figure 1-2: Fixed Field Area 1-3
Figure 1-2: Fixed Field Area (8.0 and After) 1-4
GENERAL 1-1
HARDWARE MONITOR 3-70
In-Line COP Information 1-9
INCREMENT 3-4
INCREMENT BY ONE 3-10
INDEXING 1-14
INSTRUCTION SET 2-1
Literal String 1-7
LOAD COMMUNICATE REPLY 3-67
MAKE PRESENT 3-68
MISCELLANEOUS 2-7
MISCELLANEOUS INSTRUCTIONS 3-66
MOVE ALPHANUMERIC 3-13
MOVE ARRAY 3-32
MOVE NUMERIC 3-15
MOVE NUMERIC TO FILE INDEX 3-30
MOVE NUMERIC TO OUTPUT SWITCH 3-31
MOVE SPACES 3-17
MOVE ZEROS 3-28
MULTIPLY 3-8
NON-BOOLEAN COMPARISON 2-4
NON-BOOLEAN COMPARISON AND INSTRUCTIONS 3-37
NUMBER OF SUBSCRIPTS OR INDEXES 1-13
Operand (OPND) 1-6
ORDERED TABLE SEARCH 3-73
Program Parameters 1-5
RELATED DOCUMENTATION 1-1
S-Instruction Format 1-6
S-INSTRUCTIONS 3-1
S-LANGUAGE PROGRAMS 1-1
S-Operators 1-6
SCALED MOVE NUMERIC 3-29
SET ONE BOOLEAN 3-50
SET THREE BOOLEANS 3-56
SET TWO BOOLEANS 3-55
SUBSCRIPT FACTORS 1-13
SUBSCRIPT-OR-INDEX-FLAG 1-12
SUBTRACT 3-7
TABLE BOUND 1-14
TEST BIT 3-59
TEST NUMERIC 3-60
TEST 2 BOOLEANS 3-64
TEST 3 BOOLEANS 3-65

UNORDERED TABLE SEARCH 3-72