#2201 6729

Burroughs Corporation

BUSINESS MACHINES GROUP
SMALL SYSTEMS PLANT

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.		

Burroughs Corporation
BUSINESS MACHINES GROUP
SMALL SYSTEMS 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, FCMP, CNV, LDS		

SECTION	DESCRIPTION	PAGE
1.	General Description	1
1.1	Program Parameters	2
2.0	Formats	3
2.1	S-Instruction Format	3
2.1.1	S-Operators	3
2.1.2	OPND	3
2.1.3	Literal String	4
2.1.4	Operand Index (COPX)	4
2.1.5	In-Line COP Information	5
2.2	Current Operand Table (COP)	6
2.2.1	Segment Number	7
2.2.2	Displacement	7
2.2.3	Data Length	7
2.2.4	Data Type	7
2.2.5	Subscript - Index Flag	7
2.2.6	Number of Subscripts	7
2.2.7	Subscript Factors	8
2.2.8	Indexing	8
2.2.9	Table Bound	9
2.2.10	ASCII Flag	9
3.0	Instruction Set	10, 11
3.1	Arithmetic Operands and Instructions	12, 13
3.1.1	Add Three Address	14
3.1.2	Subtract Three Address	14
3.1.3	Add Two Address	14
3.1.4	Subtract Two Address	14
3.1.5	Multiply	14
3.1.6	Divide	15
3.1.7	Increment by One	15
3.1.8	Decrement by One	15
3.2	Data Movement Operands & Instructions	16
3.2.1	Move Alphanumeric	17
3.2.2	Move Spaces	17
3.2.3	Move Numeric	18
3.2.4	Move Zeros	18
3.2.5	Multiple Move Alphanumeric	19
3.2.6	Multiple Move Spaces	19
3.2.7	Multiple Move Numeric	19
3.2.8	Multiple Move Zero	19
3.2.9	Concatenate	19
3.2.10	Scaled Move Numeric	20
3.2.11	Move Translate	20
3.2.12	Examine	21, 22
3.2.13	Edit Instructions & Edit Micro-Oper	23
3.2.13.1	Edit	23
3.2.13.2	Edit with Explicit Mask	23

3.2.13.3	Edit Micro-Operators	24, 25, 26, 27, 28
3.3	Branching Operands & Instructions	29
3.3.1	Branch Unconditionally	29
3.3.2	Branch on Overflow	29
3.3.3	Set Overflow-Toggle	29
3.3.4	Perform Enter	30
3.3.5	Perform Exit	30
3.3.6	Enter	30
3.3.7	Exit	30
3.3.8	Go To Depending	31
3.3.9	Altered Go To Paragraph	31
3.3.10	Alter	31
3.4	Conditional Branch Operands & Inst	32
3.4.1	Compare Alphanumeric	32
3.4.2	Compare Numeric	33
3.4.3	Compare for Zeros	33
3.4.4	Compare for Spaces	33
3.4.5	Compare for Class	34
3.4.6	Compare Multiple	35
3.4.7	Compare Repeat	35
3.5	Miscellaneous Instruction	36
3.5.1	Communicate	36
3.5.2	Fetch	36
3.5.3	Convert	36
3.5.4	Load Data Segment N	36

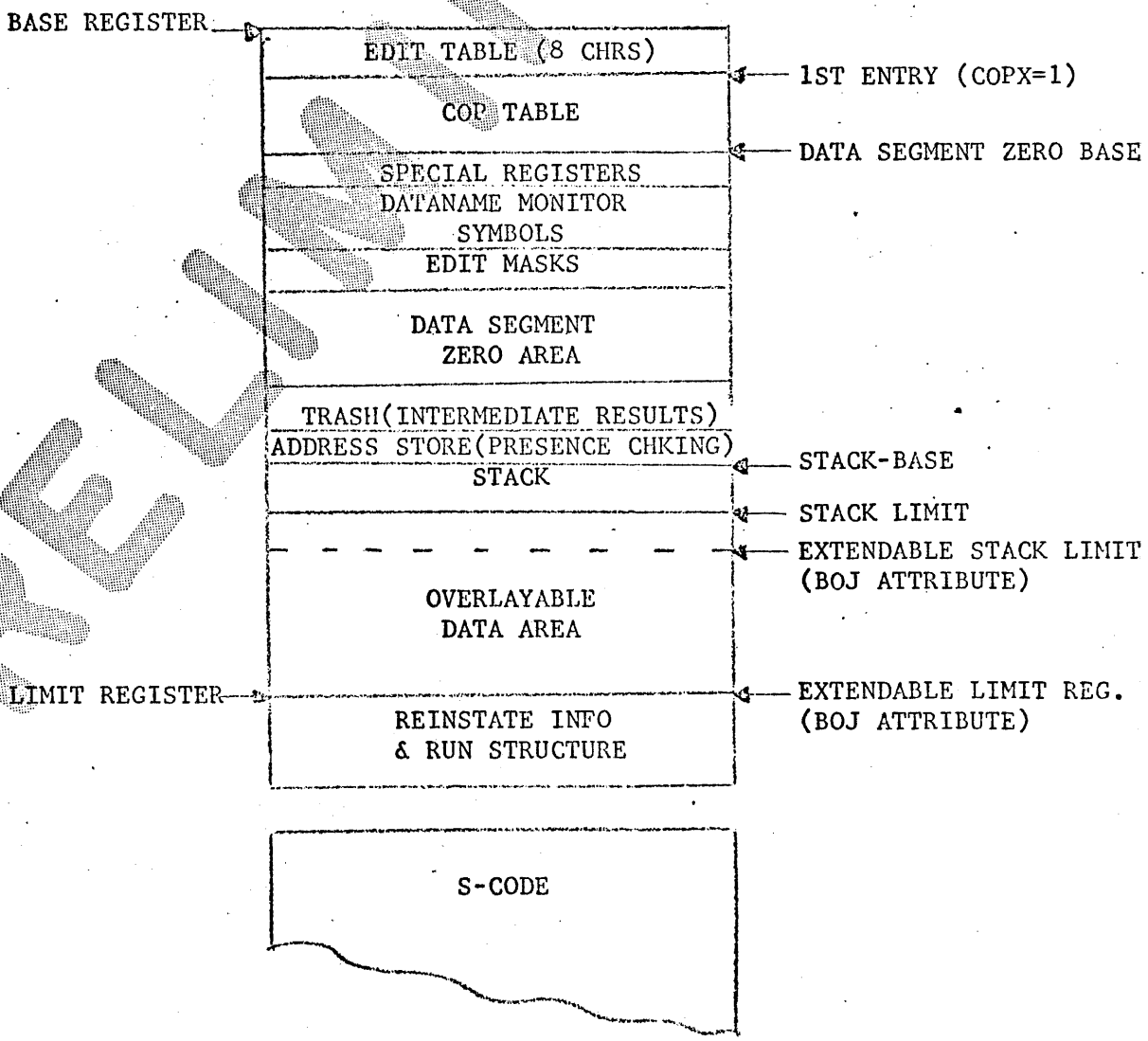
1. GENERAL DESCRIPTION

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 a data space only. All program code, organized in segment form is stored at any available location in memory.

The data space include a non-overlayable area which contains the COP Table and various other parameters such as Editing Masks and Record Areas.

Various parameters necessary for the running of the S-Langauge object code and maintained by the MCP are stored beyond the Limit Register.

A typical cobol program layout in memory is as follows:



1.1 Program Parameters

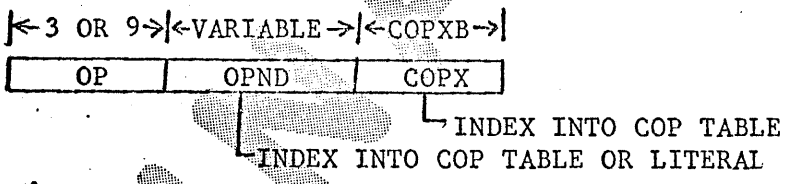
The parameters pertaining to a particular program are listed below. The number of bits to contain the parameter is listed below the parameter name.

BDISPB (5)	Branch displacement container size
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 (6)	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

2.0 FORMATS

2.1 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.

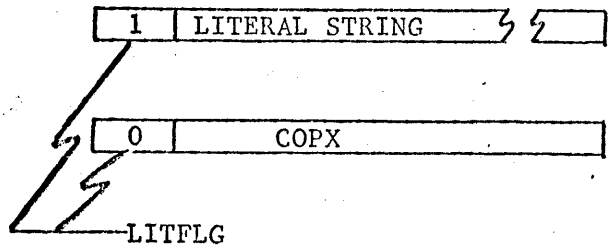


2.1.1 S-Operators

The most frequently used S-operators are encoded in a 3-bit S-operator denoted as OP1. If OP1=7, the operator is encoded in the next 6 bits denoted as OP2. If OP1<7, OP2 is omitted.

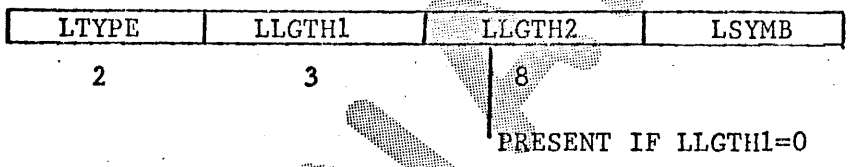
2.1.2 OPND

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. In special cases, where specified, an operand could be either contained in the instruction as a literal or could be 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:



2.1.3 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 are included in the code stream immediately following the LITFLG. The format is as follows:



The literal type is specified by a 2-bit LTYPE as follows:

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

The length of the literal in binary is encoded in LLGTH1 and LLGTH2. If the literal is <8 digits or characters, its length is encoded in LLGTH1 and LLGTH2 is omitted. If the literal is ≥ 8 digits or characters, its length is encoded in LLGTH2 with LLGTH1=0. The maximum literal length is 255 digits or characters excluding the sign.

2.1.4 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 513 and 1024 entries would require 10 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.

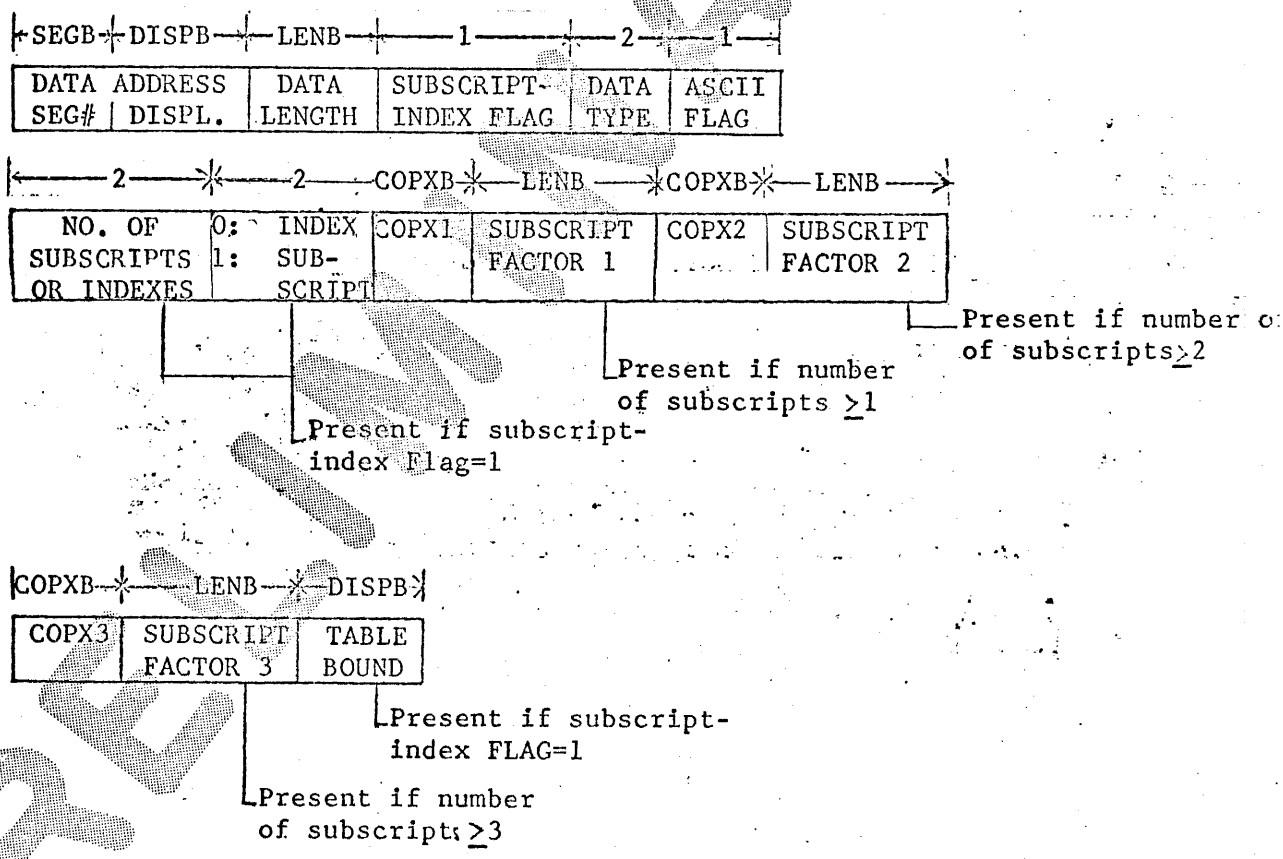
A COPX value of zero specifies that the COP Table entry is contained in line in the S-Instruction itself rather than in the COP Table. Note: The base address of the COP Table points to an unused entry.

2.1.5 In-Line COP Information

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.

The format for in-line COP information differs from its COP Table format (see section 2.2) when subscripting or indexing is required.

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



Note: COPX1, COPX2, or COPX3 may be in-line entries but may not be subscripted or indexed.

Note: A COPX for each index value or subscript (up to 3) must be present.

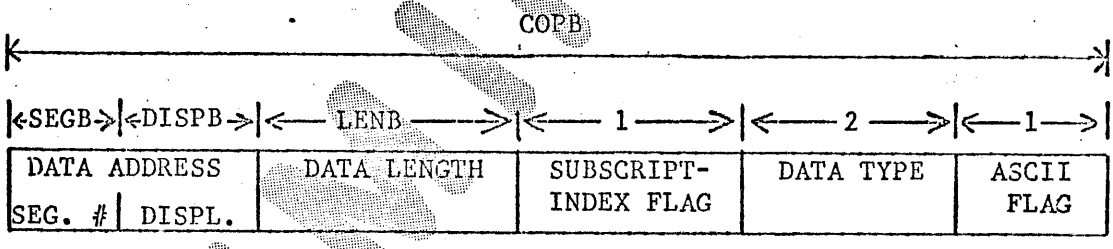
Note: A COPX for each subscript value (up to 3) is immediately followed by a subscript factor.

2.2 Current Operand Table (COP)

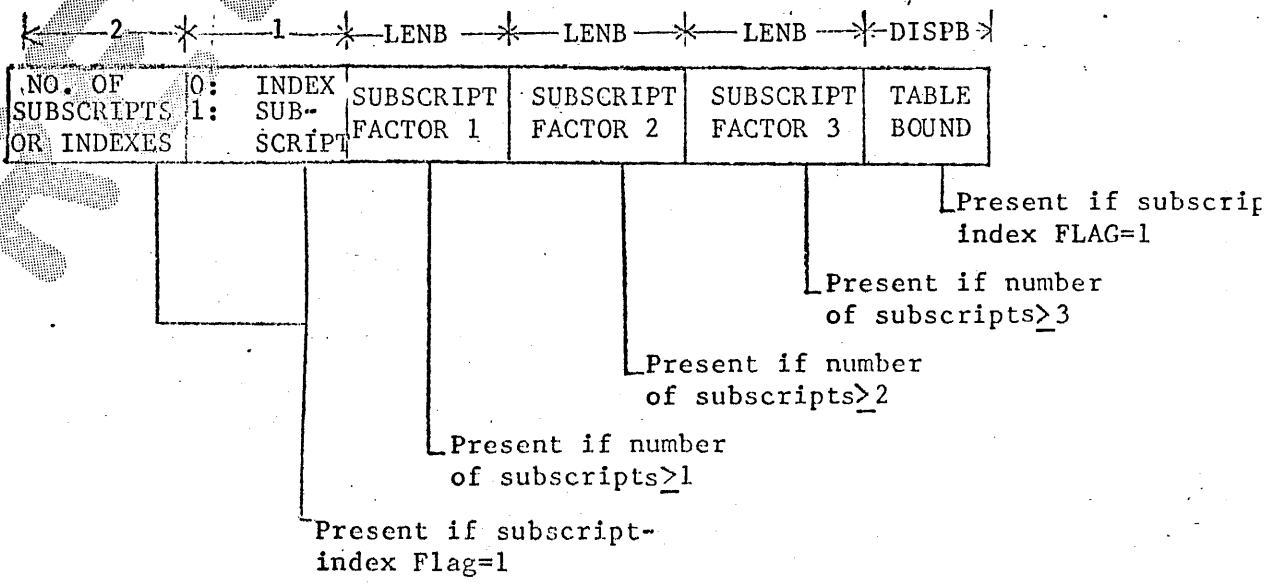
The COP Table consists of a set of entries, which contain the attributes of a variable. The width of one entry is a function of the source program and is determined by the number of bits required to express its attributes of segment number, displacement and length plus 4 bits for expressing a subscript flag, data type and ASCII flag.

When the attributes exceed one entry, multiple 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 the COP Table is as follows:



BITS 1 to ? 1 to 21 1 to 18



2.2.1 Segment Number

Segment number is expressed in binary and specifies the data segment number of the operand. The container size (SEGB) is function of the maximum number of data segments specified in the source program.

2.2.2 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 a 0 MOD 4 BIT address. The container size (DISPB) is a function of the maximum data displacement specified in the source program.

2.2.3 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.

2.2.4 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)

2.2.5 Subscript - Index Flag

The subscript-Index Flag bit is true to indicate subscripting or indexing and false otherwise. When true the next following entry(s) contain the necessary subscripting - indexing factors.

2.2.6 Number of Subscripts or Indexes

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

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

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

2.2.7 Subscript Factors

When subscripting is indicated, 1 to 3 fields of LENB bits containing the binary factor by which each subscript value is to be multiplied to obtain the proper digit address is required. The factor is the digit displacement between elements of the table. The value 1 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 is negative, its absolute value is used and an error communicate will be issued only if the final result exceeds the table-bound.

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

If the sum of the multiplied subscript values exceeds the value given by table-bound, an error communicate will be issued.

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

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.

2.2.8 Indexing

When indexing is indicated, a COPX for each index value (up to 3) 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 24-bit field. The value consists of six 4-bit decimal digits and is unsigned. The value is converted to binary and combined with the binary data address at execution time.

If the sum of the index value exceeds the table bound then an error communicate will be issued.

2.2.9 Table Bound

Table bound is a binary value used to specify the limit value for subscripting and indexing. Its container size is DISPB

2.2.10 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	MMVA	CAT	CPA
SUB	MVS	MMVS	SMVN	SPA
INC	MVN	MMVN	MVT	CPM
DEC	MNZ	MMVZ	EXA	
INC1				
DEC1				
DIV				

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

EDT
EDTE
CLASS

3.0 INSTRUCTION SET

Arithmetic

Name	Nmemonic	OP	Arguments
Increment	INC		OPND1, COPX1
Add	ADD		OPND1, COPX1, COPX2
Decrement	DEC		OPND1, COPX1
Subtract	SUB		OPND1, OPND2, COPX1
Multiply	MUL		OPND1, COPX1, COPX2
Divide	DIV		OPND1, OPND2, COPX1
Increment by one	INC1		COPX1
Decrement by one	DEC1		COPX1

Data Movement

Name	Nmemonic	OP	Arguments
Move Alphanumeric	MVA		OPND1, COPX1
Move spaces	MVS		COPX1
Move numeric	MVN		OPND1, COPX1
Move zeros	MVZ		COPX1
Multiple Move Alphanumeric	MMVA		N, OPND1, COPX1, ...COPXN
Multiple Move Spaces	MMVS		N, COPX1, ...COPXN
Multiple Move Numeric	MMVN		N, OPND1, COPX1, ...COPXN
Multiple Move Zeros	MMVZ		N, COPX1, ...COPXN
Concatenate	CAT		N, COPX1, OPND1, OPNDN
Edit	EDT		OPND1, COPX1, DADDR
Edit with Explicit Mask	EDTE		OPND1, COPX1. MASK
Scale Move Numeric	SMVN		OPND1, COPX1, V, SCL
Examine	EXA		M, T, COPX1, OPND1, COPX2 OPND2
MOVE TRANSLATE	MVT		V, OPND1, COPX1

3.0 Instruction Set (Cont'd)

Branching

Name	Nmemonic	OP	Arguments
Branch on overflow	BOF		V, BADDR
Set overflow	OFV		V
Branch Unconditionally	BUN		BADDR
Perform Enter	PERF		K, BADDR
Perform Exit	PXIT		K
Enter	NTR		BADDR
Exit	XIT		
Go to Depending	GTD		COPX1, N, BDISP,...BDISPN
Altered Go to Paragraph	GOPAR		DADDR
Alter	ALTER		DADDR, ACON

Conditionally Branching

Name	Nmemonic	OP	Arguments
Compare Alphanumeric	CPA		OPND1, COPX1, R, BADDR
Compare Numeric	CPN		OPND1, COPX1, R, BADDR
Compare for zeros	ZRO		COPX1, R, BADDR
Compare for Spaces	SPA		COPX1, R, BADDR
Compare for Class	CLASS		COPX1, C, BADDR
Compare Multiple	CPM		BADDR, COPX1, R1, OPND1, ...Rn, OPNDn, Rk
Compare Repeat	CRPT		OPND1, COPX1, R, BADDR

Miscellaneous

Name	Nmemonic	OP	Arguments
Communicate	COMM		COPX1
Fetch Communicate Message Pointer	FCMP		DADDR
Convert	CNV		OPND1, DADDR
Load Data Segment	LDS		DSECN, DADDR

3.1 Arithmetic Operands and Instructions

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)

There is no restriction as to type of operand permitted in an 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. If the result is zero, the sign is set to 1100.

Four-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.

Eight-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 eight 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 an defined below for the 4-bit unit. The leftmost four bits, except for a sign, are ignored.

3.1. Cont'd

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, when the field length of the result is shorter than either of the operands, the correct result is stored if it can fit. Results generated when 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 types of fields is identical.

No overlap of operands nor result fields are permitted in divide. Results generated under any condition of overlap are not specified.

3.1.1 Add Three Address | ADD OPND1, COPX1, COPX2 |

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

3.1.2 Subtract Three Address | SUB OPND1, OPND2, COPX1 |

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

3.1.3 Add Two Address | INC OPND1, COPX1 |

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

3.1.4 Subtract Two Address | DEC OPND1, COPX1 |

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

3.1.5 Multiply | MUL OPND1, COPX1, COPX2 |

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

3.1.6 Divide `DIV OPND1, OPND2, COPX1`

Algebraically divide a dividend denoted by OPND2 by a divisor denoted by OPND1 and store the quotient in location denoted by COPX1. Store the remainder in the location denoted by OPND2.

The result field length is the difference of the lengths of the two operands and must be denoted by COPX1.

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 an overflow toggle being set.

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.

3.1.7 Increment by One `INC1 COPX1`

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

3.1.8 Decrement by One `DEC COPX1`

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

3.2 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 operation are specified under the description of the particular operation.

See section 3.1 Arithmetic Operands and Instructions for a description of the four types of fields.

For multiple move operations, the number of destination fields (or origin fields when applicable) is specified by a 4-bit binary value N. The value ranging from 0000 to 1111 is used to indicate 1 to 16 destinations.

Note that the destination fields for a multiple move operation need not have the same data type nor data length.

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

3.2.1 Move Alphanumeric `MVA OPND1, COPX1`

Move 8-bit or 4-bit units from the origin location denoted by OPND1 to the destination location denoted by COPX1.

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

If the data type of the origin field is 4-bit, each 4-bit unit with the exception of the sign, if signed, is moved to the destination with 1111 if EBCDIC or 0011 if ASCII appended to the left of each 4-bit unit.

If the data type of the origin field is 8-bit, each 8-bit 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 length is lesser in size than the source length, the source data is truncated on the right.

Overlapping operand fields are permitted if the destination field is at least 3 character displaced and is at a higher addressed location than the original field. The data type of both fields must be 8-bit. It can be assumed that the source is moved 3 characters at a time into the destination field and that the move is from left to right.

3.3.3 Move Spaces `MVS COPX1`

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.

3.2.3 Move Numeric [MVN OPND1, COPX1]

Move 8-bit or 4-bit units from the origin location denoted by OPND1 to the destination location denoted by COPX1.

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

If the destination field is unsigned, the sign of the origin 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 origin 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 4-bits are moved.

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.

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 position of a field.

Overlapping operand fields are permitted if the destination field is at least 6 digits displaced and is at a higher addressed location than the original field. The data type of both fields must be 4-bit. It can be assumed that the source is moved 6 digits at a time into the destination field and that the move is from left to right.

3.2.4 Move Zeros [MVZ COPX1]

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.

3.2.5 Multiple Move Alphanumeric `MMVA N, OPND1, COPX1...COPXN`

Move 8-bit or 4-bit units from the origin location denoted by OPND1 to each of the N destination locations denoted by COPXN.

The rules specified for Move Alphanumeric are applicable.

3.2.6 Multiple Move Spaces `MMVS N, COPX1...COPXN`

Fill each of the N destination fields denoted by COPXN with spaces.

The rules specified for Move Spaces are applicable.

3.2.7 Multiple Move Numeric `MMVN N, OPND1, COPX1...COPXN`

Move 8-bit or 4-bit units from the origin location denoted by OPND1 to each of the destination locations denoted by COPXN.

The rules specified for Move Numeric are applicable.

3.2.8 Multiple Move Zero `MMVZ N, COPX1,...COPXN`

Fill each of the N destination fields denoted by COPXN with zeros of the appropriate type.

The rules specified for Move Zeros are applicable.

3.2.9 Concatenate `CAT N, COPX1, OPND1,...OPNDN`

Move each of the N fields denoted by OPND1 through OPNDN in the order specified into an output string starting at the location denoted by COPX1.

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.

3.2.10 Scaled Move Numeric `SMVN OPND , COPX1, V, SCL`

If V=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 are zeros.

If V=1, perform a Move Numeric operation after first subtracting the scale factor from the field length of the source field.

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).

3.2.11 Move Translate `MVT V, OPND1, COPX1`

Move 8-bit units from the origin location denoted by OPND1 to the destination location denoted by COPX1 translating enroute.

If V=0, the translation is ASCII to EBCDIC. If V=1, the translation is EBCDIC to ASCII.

The data type of the source and destination fields are ignored and are assumed to be unsigned 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 or 0010 0000 if ASCII).

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

Total overlap of operand field is permitted to allow in place translation.

3.2.12 Examine EXA M,T, COPX1, OPND1, COPX2, OPND2

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 parameter $T=T_1T_2T_3T_4$ as follows:

T_1T_2 = 00 undefined
01 Tally T_3T_4 occurrences of the character specified by OPND2.
10 Replace T_3T_4 occurrences of the character specified by OPND2.
11 Tally and replace T_3T_4 occurrences of the character specified by OPND2.

T_3T_4 = 00 ALL
01 (all) Leading
10 Until First
11 First

NOTE: $T_1T_2T_3T_4$ = 0111 and 1111 not specified and results are undefined.

The OPND2 argument is not present when T_1T_2 = 01.
The COPX2 argument is not present when T_1T_2 = 10.

The data type of the examined operand is assumed to be unsigned 8-bit.

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 4-bits 1111 if EBCDIC or 0010 if ASCII appended to the left.

3.2.12 Con't.

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 4 bits of the position replaced is set to the 4 bits 1111 if EBCDIC or 0011 if ASCII and the rightmost 4 bits receives the 4 bits from the replacing source. When 8-bit format is specified, the position replaced receives all 8 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.

The parameter M=0 denotes numeric items. Only the rightmost 4 bits of a character are used in comparing. The leftmost 4 bits are ignored.

The parameter M=1 denotes alphanumeric items. All 8-bits of a character are used in comparing.

3.2.13 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.

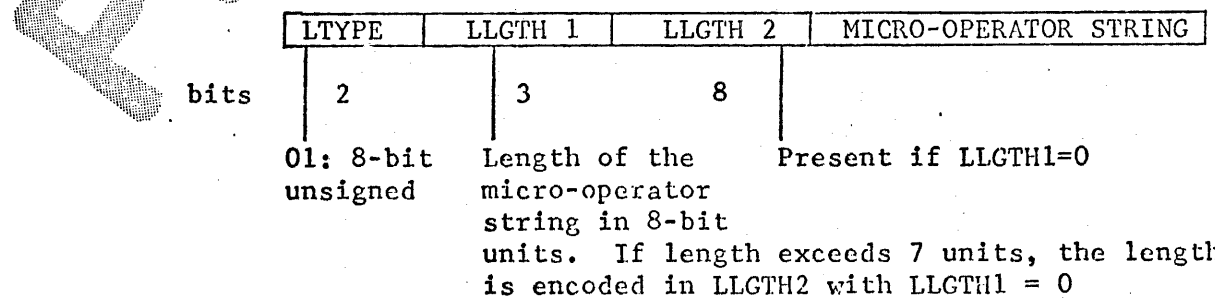
3.2.13.1 Edit `EDT OPND1, COPX1, DADDR`

Move data from the source location denoted by OPND1 to the destination location 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.

3.2.13.2 Edit with Explicit Mask `EDTE OPND1, COPX1, MASK`

Move data from the source location denoted by OPND1 to the destination location 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:



3.2.13.3 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 Set Z=1	Start Zero Suppress
1010 0011	CCP $\overline{P}$	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 or 2) used to indicate a conditional selection between two table constants or 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".

3.2.13.3 (Cont'd.)

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.

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 4 bits 1111 to the left before storing in the destination. If an 8-bit unit is moved, the 4-bits 1111 are substituted for the leftmost 4 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 4 bits 1111 to the left before storing in the destination. If an 8-bit unit is moved, it is moved unchanged.

3.2.13 (Con't.)

Move Suppress

The micro-operation move digit is performed if the 4-bit unit or the rightmost 4 bits of the 8-bit unit of the source field is not equal to 0000.

If the appropriate 4 bits of the source field unit is equal to 0000, the suppress toggle "Z" is inspected. If "Z" equals "1" indicating non-suppress mode, the micro-operation 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 ≠ 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 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.

3.2.13 (Cont'd.)

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	P=0		Move Table Entry 7 (Blank)
S=0	P=1		Move Table Entry 2 (Asterisk)
S=0		T=8	Move Table Entry 0 (Plus)
S=1		T=8	Move Table Entry 1 (Minus)
S=1		T=9	Move Table Entry 7 (Blank)
S=1		T=9	Move Table Entry 1 (Minus)
S=1		T=10	Move in-line Table Entry

Insert Suppress

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

3.2.13 (Con't.)

Insert Float

Move the table entry "T" and/or perform the micro-operation 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 ≠0 T=0...7	Move table entry T, then move digit
Z=0 Source ≠0 T=8 S=0	Move table entry 0 (plus)then move digit.
Z=0 Source ≠0 T=8 S=1	Move table entry 1(minus)then move digit.
Z=0 Source ≠0 T=9 S=0	Move table entry 7(blank)then move digit.
Z=0 Source ≠0 T=9 S=1	Move table entry 1(minus)then move digit.
Z=0 Source ≠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=1 T=8	Move table entry 1
Z=0 S=0 T=8	Move table entry 0
Z=0 S=1 T=9	Move table entry 1
Z=0 S=0 T=9	Move table entry 7
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 is moved; otherwise continue with the next in-line operator.

End of Mask

Terminate the micro-operator operations.

Start Zero Suppress

Set "Z" to the zero state.

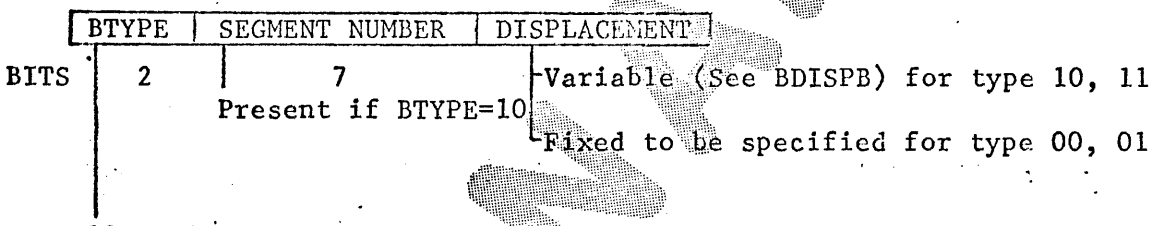
Complement check protect

Complement the state of "P".

3.3 Branching Operands and Instructions

A branch address argument "BADDR" can be any one of four types as specified below. Restrictions as to type for a specific instruction are specified in the description of the individual instruction.

BADDR has the following format:



00: Relative to beginning of this byte in a positive direction.
01: Relative to beginning of this byte in a negative direction.
10: Relative to a new code Segment Base (inter segment branch).
11: Relative to the current code Segment Base (intra segment branch).

Displacement is an unsigned binary value which specifies the bit displacement of an instruction relative to itself or to a segment base. The container size of the displacement is fixed (to be specified) for type 00 and 01 and is a program parameter (BDISPB) for type 10 and 11.

3.3.1 Branch Unconditionally `BUN BADDR`

Obtain the next instruction from the location specified by BADDR.

3.3.2 Branch on Overflow `BOF V, BADDR`

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

The overflow toggle is unchanged.

3.3.3 Set Overflow-Toggle `OFL V`

Set the overflow toggle to V.

Note: The overflow toggle is set to 1 when divide by zero is encountered.

3.3.4 Perform Enter [PERF K, BADDR]

Format a stack entry with the following format:

K	BTYPE	SEGMENT NO.	DISPLACEMENT
8	2	7	24

BITS

Obtain the 8-bit value K and the 2-bit value BTYPE from the instruction and insert it into the stack.

If the BTYPE inserted is 10, insert the current segment number into the stack.

Insert a displacement value relative to the active base and pointing to the next sequential S-Instruction into the stack.

Adjust the stack pointer (STKPTR) to point to the next possible entry.

Obtain the next instruction from the location specified by BADDR.

3.3.5 Perform Exit [PXIT K]

Compare the 8-bit value K with the K in the current stack entry and if unequal proceed to the next in-line S-Instruction. If equal, adjust the stack position (STKPTR) to point to the previous entry and obtain the next S-Instruction from the information contained in the removed stack entry.

3.3.6 Enter [NTR BADDR]

Same function as "PERF". K is assumed equal to 0.

3.3.7 Exit [XIT]

Same function as "PXIT". K is assumed equal to 0.

3.3.8 Go To Depending `GTD COPX1, N, BDISPO,...BDISPN`

Compare the 10 bit binary value N with the variable specified by COPX1. The variable is first converted to a binary value modulo 2^{24}.

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

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

BDISP is a unsigned binary value which specifies the bit displacement of an instruction relative to the current segment base. The container size of BDISP is BDISPB.

3.3.9 Altered Go To Paragraph `GOPAR DADDR`

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

3.3.10 Alter `ALTER DADDR, ACON`

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

BTYPE = 00 and 01 (relative branches) are not permitted in the "ACON".

See section 3.3.9 for definition of DADDR and ACON.

3.4 Conditional Branch Operands and Instructions

The two operands are compared and the result (greater, less, equal) is in turn compared to the relation (R) (greater, less, equal, not equal, greater or equal, less or equal) being tested, If the relation condition is met a transfer to the address (BADDR) given in the instruction occurs. Control is passed to the next sequential instruction otherwise. The relation (R) is defined as follows:

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

No overlap of fields is permitted. Results generated under any condition of overlap are not specified.

3.4.1 Compare Alphanumeric `CPA OPND1, COPX1, R, BADDR`

Compare the two operand fields according to their binary value.

Comparison proceeds from left to right.

When the field sizes are different and the equal size portions compare equal, the longer field is tested for trailing blanks (0100 0000 if EBCDIC or 0011 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.

That field, which has the left most character that is greater (less) than the corresponding character in the second field, is considered the larger (smaller).

Unsigned 8-bit data format is assumed with no checking to verify that the data types are both unsigned 8-bit.

3.4.2 Compare Numeric [CPN OPND1, COPX1, R, BADDR]

Compare the two operands fields according to their algebraic value 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 right-most 4 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 digits values are undefined.

3.4.3 Compare for Zeros [ZRO COPX1, R, BADDR]

Compare two operand fields according to their absolute binary value considering the second field to be comprised of all zeros (0000).

There is no restriction as to data type. In comparing an 8-bit character only the right-most 4 bits of the character are considered. The other bits are ignored. In comparing signed, 4-bit format, the sign position is ignored.

3.4.4 Compare for SPACES [SPA COPX2, R, BADDR]

Compare two operand fields according to their binary value considering the second field to be comprised of all spaces (0100 0000 if EBCDIC or 0011 0000 if ASCII).

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

3.4.5 Compare for Class [CLASS COPX1, C, BADDR]

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

In the numeric test, each character is range-checked for 1111 0000 through 1111 1001 and each digit is range-checked for 0000 through 1001. A sign in the sign position for signed data will not affect the outcome of a numeric test.

3.4.6 Compare Multiple `CPM BADDR, COPX2, R1, OPND1...Rn, OPNDn,Rk`

Compare the first operand field with each of the subsequent N operand fields under control of the variant denoted by the first bit of the relation (Ri) associated with the specific operand OPNDi.

The relation (R) is a four bit field which is defined as follows:

R = 0XXX compare alphanumeric
1XXX compare numeric

XXX = 000 QUIT
001 GTR
010 LSS
011 NEQ
100 EQL
101 GEQ
110 LEQ
111 undefined

The individual comparisons follow the same rules as the appropriate COMPARE ALPHANUMERIC OR COMPARE NUMERIC.

The instruction is terminated when:

A) one of the relations is met in which case program control passes to the address BADDR or,

B) Rk = 000 is found in which case program control passes to the next sequential instruction.

3.4.7 Compare Repeat `CRPT OPND1, COPX1, R, BADDR`

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 be greater than 2 and must divide evenly into the size of COPX1. Otherwise, the results of the compare may be erroneous.

3.5 Miscellaneous Instruction

3.5.1 Communicate `COMM COPX1`

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

The length is converted from a digit or character length to a bit length. The address is converted from the Data Segment Zero Base Relative Address to the Base Register Relative Address.

3.5.2 Fetch `FCMP DADDR`

Move the last 24-bits of information from the RS COMMUNICATE MSG PTR to the located specified by DADDR.

See section 3.5.4 for definition of DADDR.

3.5.3 Convert `CNV OPND1 DADDR`

Convert the operand denoted by OPND1 from a decimal value to an unsigned 24-bit binary value truncating or zero filling on the left if necessary.

The operand must be either unsigned 4-bit or unsigned 8-bit units. Place the result at the location specified by DADDR. See section 3.5.4 for definition of DADDR.

3.5.4 Load Data Segment N `LDSN DSEGN, DADDR`

Load the data segment specified by DSEGN and place the Base relative address of the segment at the location specified by DADDR.

The container size of DSEGN is SEGB.

DADDR is an unsigned binary value which specifies a digit displacement from the Data Segment Zero Base.

The container size of DADDR is DISPB.