PRODUCT SPECIFICATION

REV LTR	REVISION ISSUE DATE	APPROVED BY	REVISIONS
I	Cont'd		Pages 3-119 through 3-121: Added REDUCE OPERATOR explanation MARK VI.1 CHANGES
J	8/2/77	J. Hale	Page 1-1 to 3-174 B1700 changed to B1800/B1700 throughout. Page 2-5 Added INDEXED LOAD FIELD ADDRESS (ILFA) Op-code to Load Operators Instruction Set. Added LOAD ARRAY FIELD ADDRESS (LAFA) Op-code to Load Operators Instruction Set. Added LOAD FIELD ADDRESS (LFA) Op-code to Load Operators Instruction Set. Added LOAD FIELD ADDRESS FROM PREVIOUS (LFAP) Op-code to Load Operators Instruction Set. Page 2-6 Added REFER (REFR) Op-code to Load Operators Instruction Set. Page 3-64 Added INDEXED FIELD ADDRESS (ILFA) Operator to Load Operators Section. Page 3-69 Added LOAD ARRAY FIELD ADDRESS (LAFA) Operator to Load Operators Section. Page 3-70 Added LOAD FIELD ADDRESS (LFA) Operator to Load Operators Section. Page 3-72 Added LOAD FIELD ADDRESS FROM PREVIOUS (LFAP) Operator to Load Operators Section. Page 3-78 Added REFER (REFR) Operator to Load Operators Section.

#2201 2389

Burroughs Corporation

COMPUTER SYSTEMS GROUP
SANTA BARBARA PLANT

SDL S-LANGUAGE
B1700 SDL S-LANGUAGE

PRODUCT SPECIFICATION

REVISIONS

REV LTR	REVISION ISSUE DATE	PAGES REVISED ADDED DELETED OR CHANGE OF CLASSIFICATION	PREPARED BY	APPROVED BY
F	1-5-72	Sec 3.8 Added note Sec 3.34, 3.35, 3.36, 3.37, 3.38 Changed Descriptor format to allow variable lengths. Sec 3.36 Added second version of operator-CDFM Sec 3.45 Added second version of operator-RTRN Sec 3.63-3.67 Added new operators SVST, HMON, OVLY, PRFL, SLL		
G	3-27-73	Sec 1.1 Added order of stacks in S-Memory. Fig 1 Changed PPS to show Page #. Sec 2.1 Added 13 bit operators. Sec 2.4 Added Page # and changed segment # size. Sec 3.0 Added new operators. Sec 3.5 Changed BIN result to a type BIT. Sec 3.9 thru 3.32 Changed to 3.10 thru 3.33 Sec 3.9 Added SS1. Sec 3.14 DESC: Changed OP code. Sec 3.16 L: Stated values < 24 bits in length are loaded self relative Sec 3.18 AL: Corrected page table entry figure. 1=Memory, 0=Disk. Sec 3.50 thru 3.65 Changed to 3.67 thru 3.83 Sec 3.50 thru 3.66 Added EOI, XTEI, CNTR, CXIT,SLL,SSCH,TVEC,IVEC,SSD,SSWP, UBLK,DTKN,NTKN,DBLK,XFRM,HASH Sec 3.54 SSL: Changed from 3.67 Sec 3.74 LSP: Added EV, CS, NS, DB variants Sec 3.84 Added PADR operator.	WFK	4/19/73 4/25/73 5-23-73
H	11-10-75	Translation to upper case and lower case. Major revision. Renamed B1700 SDL S-LANGUAGE	K.M.K.	11-10-75
I	5/6/76	Page 1-14: Changed DISPLACEMENT on Figure 1.9 to 12/16/20 Page 2-7: Added REDUCE to SEARCH & SCAN OPERATORS Page 3-108: Added REDUCE to SEARCH & SCAN OPERATORS	K.M.K.	

Burroughs Corporation

BUSINESS MACHINES GROUP
SMALL SYSTEMS PLANT

B1700 SDL S-LANGUAGE

PRODUCT SPECIFICATION

REVISIONS

REV LTR	REVISION ISSUE DATE	PAGES REVISED ADDED DELETED OR CHANGE OF CLASSIFICATION	PREPARED BY	APPROVED BY
I	Cont'd.	Pages 3-119 through 3-121: Added REDUCE OPERATOR explanation		J Hale

TABLE OF CONTENTS

GENERAL

This product specification will describe the basic components and operators of the B1800/B1700 SDL S-Language, including its STACK MECHANISM, DATA DESCRIPTORS, and CODE and DATA ADDRESSES. After a preliminary discussion of the structure and operation of the S-Machine, its operators will be explained in detail.

RELATED PUBLICATIONS

NAME	NUMBER
SDL/UPL COMPILER	P.S. 2212 5389
B1800/B1700 SDL (BNF VERSION)	P.S. 2212 5405
B1800/B1700 SYSTEMS REFERENCE MANUAL	#1057155

COMPONENTS OF THE S-MACHINE

The SDL S-Machine is composed of the following basic elements:

1. Base-limit area

   This is the memory area for program data. It is the only area that is directly addressable and thus modifiable by SDL S-ops. This area is bounded by base and limit registers. All "absolute" data addresses in the S-machine are expressed as a bit offset from the base register. For addresses to be absolute in fact, a program's base register must be zero. The area is broken into two divisions; static memory (from base register to dynamic memory base) which is occupied by the SDL stacks, and dynamic memory (from dynamic memory base to the limit register) which is used for virtual data memory, i.e. SDL paged array page tables and resident pages.

2. Run stucture nucleus

   This contains information used by the MCP and SDL interpreter to implement an instance of the S-machine, i.e. a running SDL program.

3. Code segments and segment dictionaries

   Code Segments are virtual as in the other machines, but the Code Segment Dictionary is itself segmented, corresponding to the page-segment concept in the SDL language. Each entry in a Master Segment Dictionary represents a page of segments in the source program and points to a sub-dictionary with the entries for those segments. The Master Segment Dictionary is non-overlayable; the sub-dictionaries may be overlaid.

4. File information blocks (FIB's) and FIB dictionary

   These appear in (virtual) memory, one FIB per open file in use by the running program, used by the system for input/output operations.

5. Registers

These may be either in hardware registers or in memory depending on the state of the S-machine. The exact format and number of registers is important only to the SDL interpreter. Logically, they consist of the next instruction pointer (page, segment, and displacement), the current lexic level, and the stack top pointers for all stacks. Current LL and the DISPLAY stack pointer are the same register. Also in registers is enough information about the stacks to check for stack overflows. Underflows are not detected. Registers are initialized from the scratchpad area of the program parameter block in the code file. (See MCP reference manual). This area has the following format in SDL code files:

```
1  SCRATCHPAD,
   2  FILLER            BIT(48),
   2  PPS.BASE          BIT(24),
   2  FILLER            BIT(24),
   2  ES.BASE           BIT(24),
   2  ES.PPS.BITS       BIT(24),
   2  VS.BASE           BIT(24),
   2  FILLER            BIT(24),
   2  CS.BASE           BIT(24),
   2  CS.BITS           BIT(24),
   2  NS.BASE           BIT(24),
   2  FILLER            BIT(24),
   2  DISPLAY.BASE      BIT(24),
   2  FILLER            BIT(4),
   2  PROFILE.FLAG      BIT(1),
   2  FILLER            BIT(19),
   2  VS.BITS           BIT(24),
   2  NS.BITS           BIT(24),
   2  ES.BITS           BIT(24),
   2  PPS.BITS          BIT(24);
```

See the MCP Reference Manual for a more general discussion of code files, run structures, file information blocks, and the various dictionaries. These concepts are common to all the B1800/B170C S-machines. Note that SDL does not make use of data segments and data segment dictionary, implementing virtual data via an SDL intrinsic instead.

THE BASE-LIMIT AREA

The space is divided as shown in the diagram below, with the arrows indicating the direction of growth.

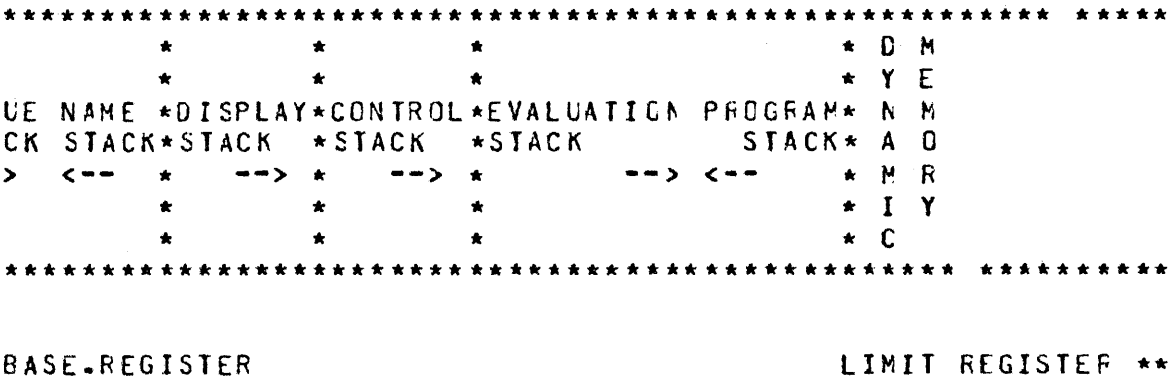


FIGURE I.1 SDL STACKS

VALUE STACK — Entries are values of data items, arbitrary in length, the characteristics of which are kept in descriptors in the NAME and EVALUATION stacks.

NAME STACK — Entries are DATA DESCRIPTORS, 48 bits in length, one descriptor for every data identifier which is currently active (not necessarily addressable) in the program. The descriptor for an array is 96 bits long, occupying two NAME STACK entries.

DISPLAY — The NAME STACK is divided into stack frames, each frame containing the descriptors for the names declared in one invocation of a procedure. Not all of these stack frames contain

descriptors which are currently accessible. The display cortains pointers into the NAME STACK, one pointer for each Lexic Level less than or equal to the current Lexic Level. Each pointer locates the base of the frame for currently aodressable names at that level. These entries are 32 bits long. For further discussicn of the display mechanism, the reader should consult literature on the implementation of ALGOL 60.

CONTROL STACK

Here are the NAME STACK pointers which locate the stack frames fcr every active procedure. Each time a prccedure which requires local cata cr parameter allocations, (i.e., recuires space on the NAME STACK) is entered, a new entry is pushed onto the CONTRCL STACK to point to its NAME STACK frame. Since the VALUE STACK contairs the data associated with NAME STACK descriptors, it too is divided into frames and the base of each frame is recorded ir the CONTROL STACK as it is allocatec. In addition to these two pointers, each entry contains the Lexic Level of the calling procedure anc the Lexic Level of the current entry. These are used by the S-Machine to maintain the DISPLAY. The fcrmat of a CONTRCL STACK entry is:

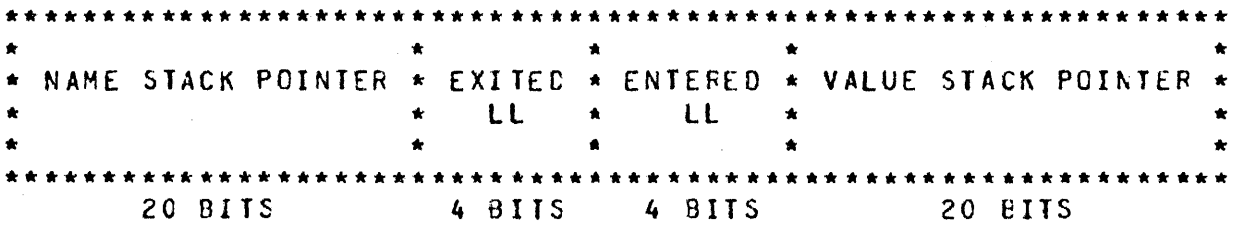


FIGURE I.2 CONTROL STACK ENTRY FORMAT

EVALUATION STACK

The EVALUATION STACK is used to hold data descriptors for the evaluation of expressions (expressions are compiled into reverse polish strings). It is also used to build actual parameter descriptors prior to their being transferred to the NAME STACK for a procedure call. Space for data during expression evaluation is allocated on top of the VALUE STACK which is kept up to date as descriptors are pushed on and popped off the EVALUATION STACK.

PROGRAM POINTER STACK

With the exception of the cycle operator used for looping, all transfers of control in the SDL machine are done via call-type operators. The next instruction pointer is saved for subsequent return by pushing it onto the PROGRAM POINTER STACK. The format of an entry in this stack is:

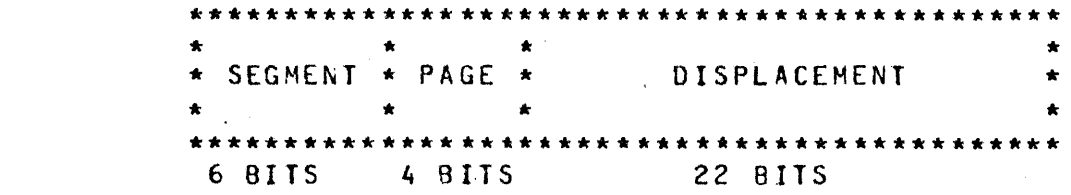


FIGURE I.3 PROGRAM POINTER STACK ENTRY FORMAT

DATA DESCRIPTORS

Simple (scalar) descriptors are 48 bits in length, with the following format:

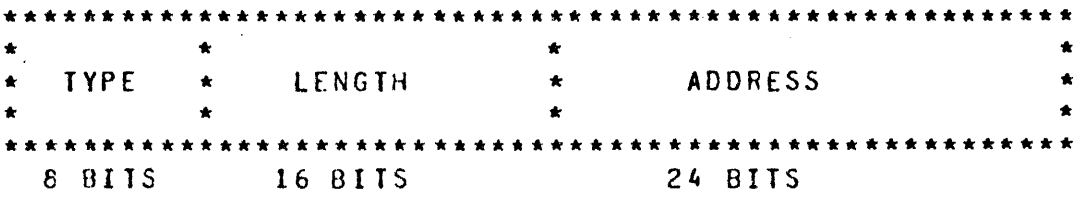


FIGURE I.4 SIMPLE DESCRIPTOR FORMAT

The address is specified as a bit offset from the base register (which is also the base of the VALUE STACK). The length is expressed in bits, regardless of the type.

One of the bits in the type field indicates whether or not this is an ARRAY DESCRIPTOR. When this bit is on, an additional 48 bits of information is appended giving the following format:

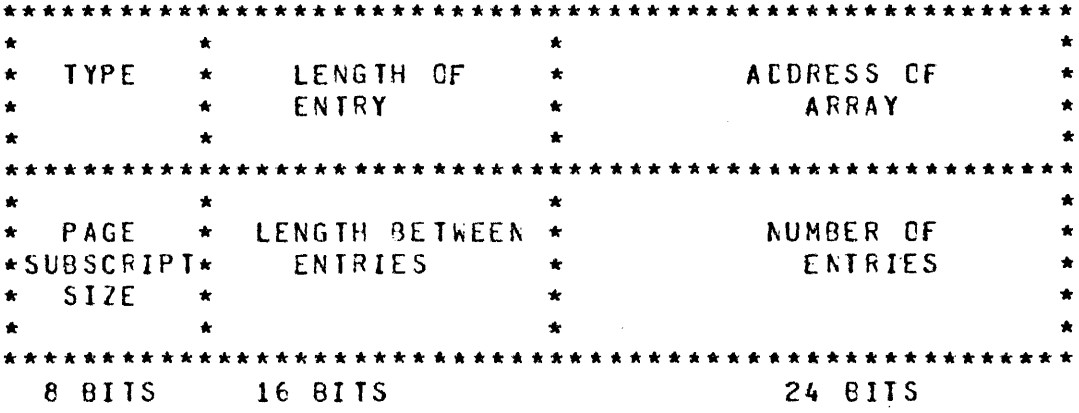


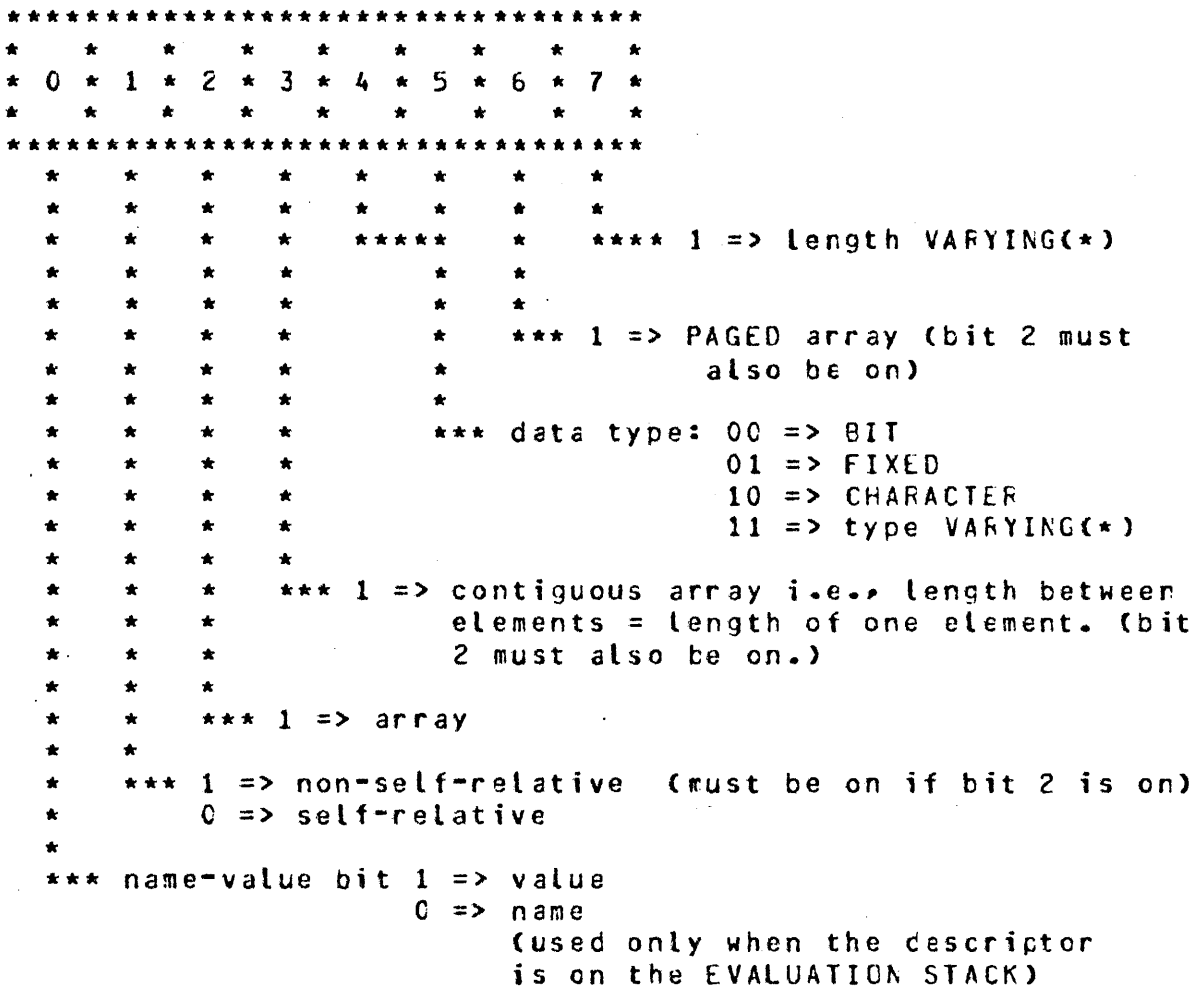
FIGURE I.5 ARRAY DESCRIPTOR FORMAT

The Page Subscript Size is only used when the PAGED ARRAY bit is on in the type field. It specifies the number of bits to shift an Array Subscript to obtain the corresponding Page Subscript (Page

sizes are always a power of two in SDL).

The length between entries is the difference between the address of one element and the address of the previous element. This must be greater than or equal to the length of one entry.

The Type Field of a descriptor has a single format, even though some bits are not meaningful in all contexts.



(*) Only used in the type field of in-line descriptors which are arguments of a CDFC operator and in the argument to a RTNC operator. The CDFC operator also uses BIT 6 in a different way; it indicates a varying array bound. In-line descriptors for other operators use BIT 0 to indicate the presence of a filler field. See IN-LINE DESCRIPTOR FORMAT.

FIGURE I.6 DESCRIPTOR TYPE FIELD FORMAT

When the data itself is 24 bits or less in lergth, it may be contained directly in the address portion of the descriptor, thus requiring less storage. In this case, the descriptor is saic tc be self-relative and the non-self-relative bit is off.

The use of the NAME-VALUE BIT is to distinguish, in the EVALUATION STACK, between descriptors which had an associated value lcaced on the VALUE STACK when they were pushed on the EVALUATICN STACK, and those which did nct. The purpose is to signal that a data item should be cut back from the VALLE STACK whenever this descriptor is cut back from the EVALUATICN STACK. The bit can only be set in non-self-relative descriptors.

PAGED ARRAY DESCRIPTORS

When the PAGED bit is on in an array descriptor, the address field of the descriptor does not point directly to the array, but rather is initialized to zero. An array load operator (ALA, AL) will then detect the first access to the array and invoke the SDL virtual memory manager to build a page table in dynamic memory. This table will be non-overlayable and the descriptor address field will be set to the page table address. The table contains one entry per array page, each with the format below:

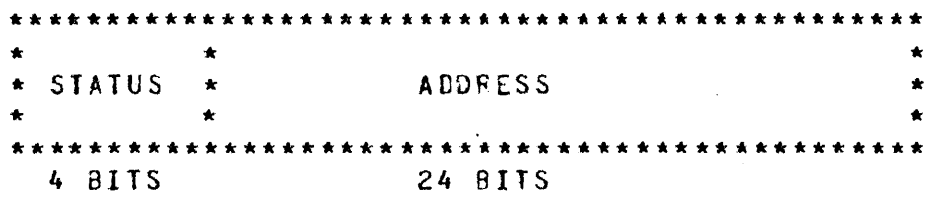


FIGURE I.7 PAGE TABLE ENTRY FORMAT

STATUS FIELD

bit 0	presence bit	1 =>	address is base relative memory address
		0 =>	address is disk address
bit 1	to be read only	1 =>	the next time this page is rolled out, turn this bit off and bit 2 on.
bit 2	read only	1 =>	this page may be overlaid without rolling it out to disk.
bit 3	unused		

An address field of zero indicates that this is a previously unaccessed page and may be created without need for a rollin.

DATA ADDRESSES

The S-Language addresses data via descriptors on the NAME STACK. At any point in an SDL program every accessible data item may be described by the Lexicographic Level at which it was declared and its ordinal location (occurrence number) within the declaration section at that level. A data address, then, consists of these two numbers which uniquely locate a descriptor in the NAME STACK. Addressing is done by using the DISPLAY to locate the NAME STACK frame corresponding to the required Lexic Level, and the occurrence number to locate the descriptor within that frame. For compactness, data addresses have a type field which indicates the sizes of the other fields, as indicated below:

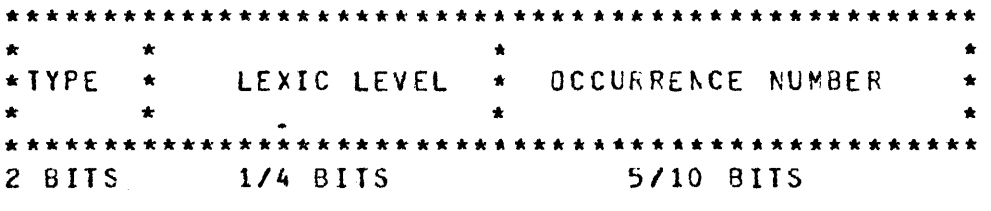


FIGURE I.8 DATA ADDRESS FORMAT

TYPE	LEXIC LEVEL BITS	OCCURRENCE NUMBER BITS	TOTAL BITS
00	4	10	16
01	4	5	11
10	1	10	13
11	1	5	8

When only one bit is used for Lexic Level, 0 means Lexic Level 0 and 1 is taken to mean the current Lexic Level, whatever it may be.

CODE ADDRESSES

Code addresses appear as arguments of operators which effect transfers of control. They are divided into three parts: the page number which selects the segment dictionary page, the segment number which selects the segment dictionary entry within that page, and the displacement which specifies a bit offset within the segment. For compactness, these numbers are encoded in different field sizes depending on a type field.

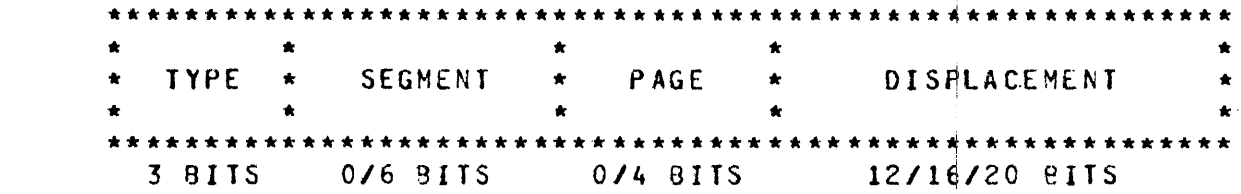
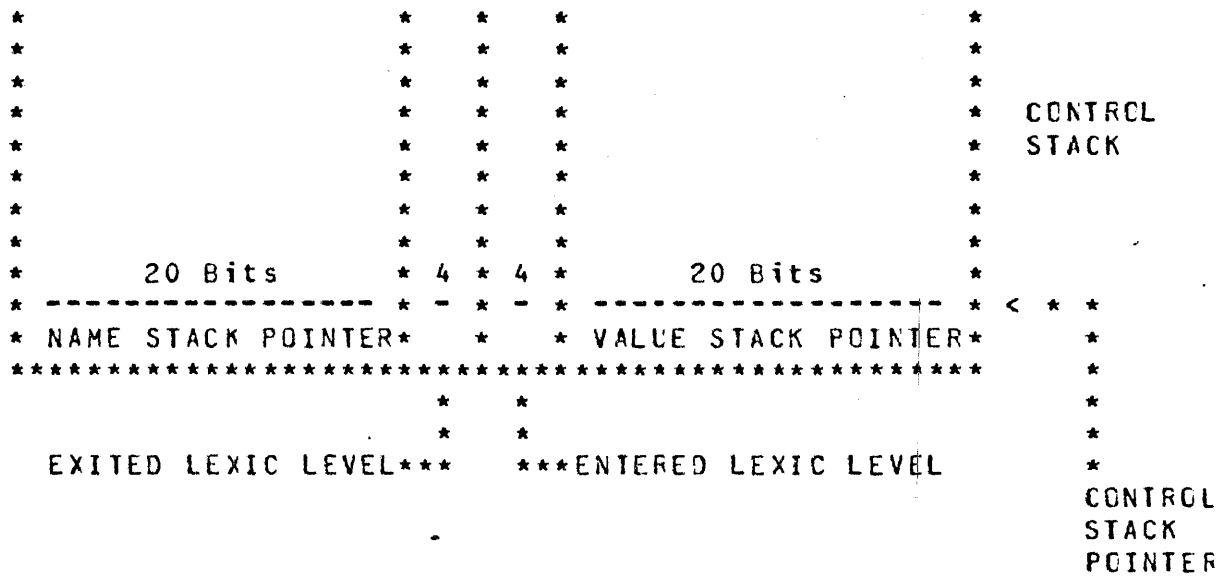


FIGURE I.9 CODE ADDRESS FORMAT

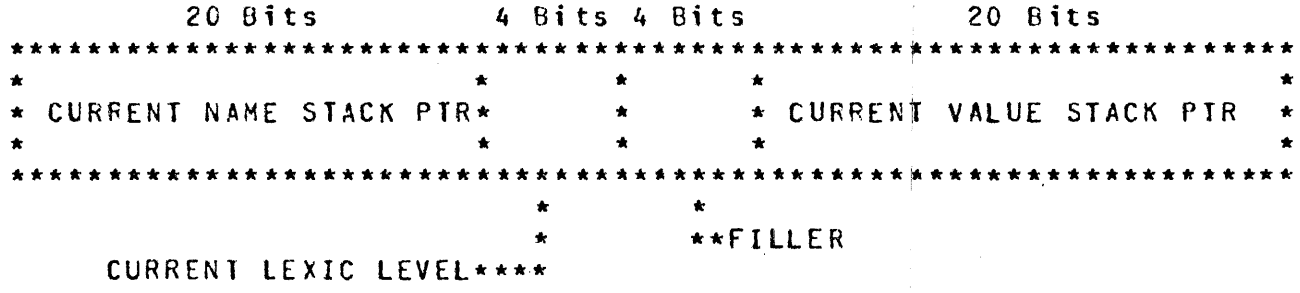
TYPE	SEGMENT BITS	PAGE BITS	DISPLACEMENT BITS	TOTAL BITS
000	CURRENT	CURRENT	12	15
001	CURRENT	CURRENT	16	19
010	6	CURRENT	12	21
011	6	CURRENT	16	25
100	6	4	12	25
101	6	4	16	29
110	6	4	20	33
111	NULL ADDRESS			

The Null Address (Type 111) is only used by the CASE operator and the length of the other fields is the same as the length of those fields in the other arguments to the case. The other fields are not used and are all zero.

CONTROL STACK MECHANISM



CURRENT CONTROL REGISTER



The following SDL declaration is used by algorithms shown under the operators which affect the CONTROL STACK.

```
DECLARE

    01 CONTROL.STACK (CS.SIZE)    BIT(48),

        02 CS.NSP                 BIT(20),

        02 CS.EXITED.LL           BIT(4),

        02 CS.ENTERED.LL          BIT(4),

        02 CS.VSP                 BIT(20),

    01 CURRENT.CONTROL            BIT(48),

        02 CURRENT.NSP            BIT(20),

        02 CURRENT.LL             BIT(4),

        02 FILLER                 BIT(4),

        02 CURRENT.VSP            BIT(20),



    (CSP,TCSP) FIXED;

CSP:=0;
```

The SDL operators that use the CONTROL STACK mechanism are:

MKS - MARK STACK

CDFM - CONSTRUCT DESCRIPTOR, FORMAL

CDFC - CONSTRUCT DESCRIPTOR, FORMAL CHECK

MKU - MARK STACK AND UPDATE

EXIT - EXIT

RTRN - RETURN

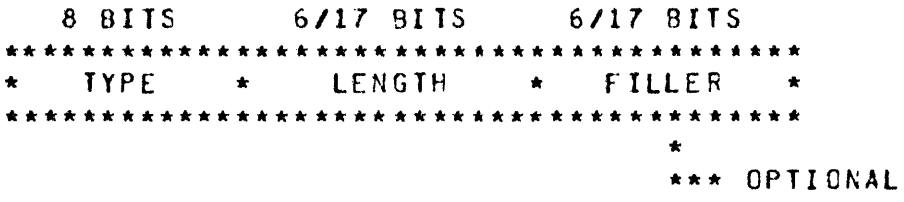
RTNC - RETURN FORMAL CHECK

XTEI - EXIT, ENABLE INTERRUPTS

IN-LINE DESCRIPTOR FORMATS

FOR CONSTRUCT DESCRIPTOR OPERATORS

SIMPLE DESCRIPTOR

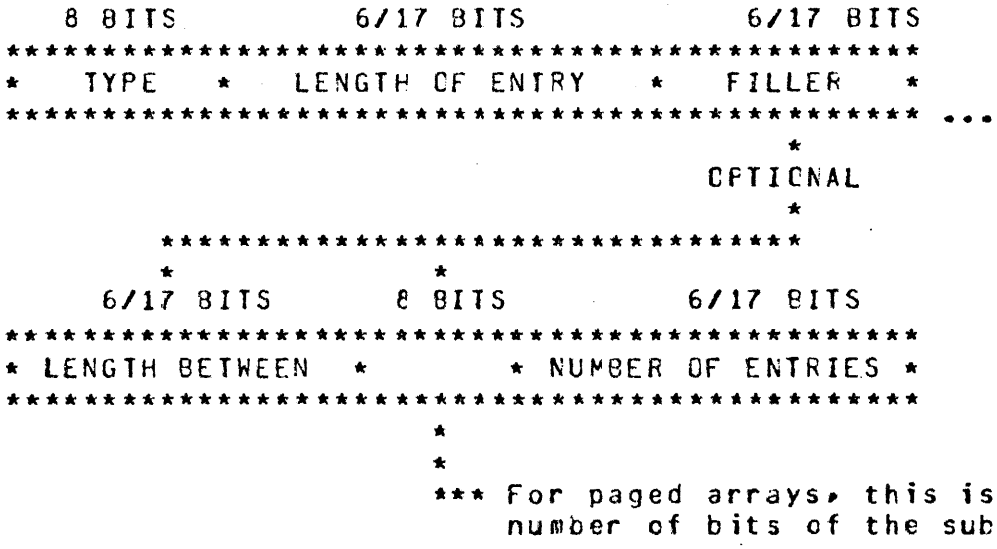


Notes:

1. The filler option is present only when bit 0 of the type field is on (=1).

2. Bit 2 of the type field is off (=0).

ARRAY DESCRIPTOR



neeced to obtain the page subscript (See AL: Array Load).

Notes:

1. The filler option is present only when bit 0 of the type field is on (=1).

2. The length between option is present only when bit three of the type field is off (=0).

3. Bit two of the type field is on (=1).

4. The page subscript size field is present only when bit six of the type field is on.

5. If bit six of the type field is on, then bits zero and three will be off.

6. The fields marked "6/17 BITS" are encodec as follcws:

First Bit	Meaning
0	5 bits follow
1	16 bits follow

USE OF THE EVALUATION STACK

Many of the SDL S-ops take operands from or leave results on the EVALUATION STACK. In fact, only the descriptor of the operand is on the EVALUATION STACK itself while the data, that is, the value of the operand, may be in the descriptor or elsewhere in the base-limit area. Conceptually, however, it is an operand with which the S-op is dealing. There are two classes of operands or results which may be on the EVALUATION STACK:

Address operands:

The descriptor is a pointer to the value of a declared data item. The descriptor on the EVALUATION STACK is non-self-relative and its name-value bit is off. This type of operand is appropriate for use as the destination of an S-op which moves data.

Value operands:

There are two classes of value operands.

Self-relative:

The descriptor on the EVALUATION STACK is marked self-relative and its name-value bit is off. Instead of the address field of the descriptor being a pointer to the data, the data itself is contained in the address field of the descriptor.

Non-self-relative:

The descriptor on the EVALUATION STACK is marked non-self-relative and its name-value bit is on. The data is on top of the VALUE STACK, located by the address field in the descriptor. When this type of operand is removed from the EVALUATION STACK, its value is removed from the VALUE STACK as well.

Value operands are temporary values as opposed to actual variables of the program.

A particular S-op often requires that its operands be of a particular class. It does not make sense, for example, for the destination operand of a STOD (store destructive) to be a value operand. Some S-ops put other restrictions on their operands, usually concerning type or length. Unless specifically indicated, these restrictions are not checked by the interpreter but, if not met, the results of the operation are undefined.

INSTRUCTION SET

Op codes are four, six, ten or thirteen bits in length. The lengths have been assigned according to static frequency of the S-op, thus compacting code space as much as possible.

RELATIONAL OPERATORS

NAME	MNEMONIC	OP CODE	ARGUMENTS
EQUAL TO	EQL	1010 01	
LESS THAN	LSS	1111 01 1010	
LESS THAN OR EQUAL TO	LEQ	1111 00 1110	
GREATER THAN	GTR	1111 00 1001	
GREATER THAN OR EQUAL TO	GEQ	1111 00 1101	
NOT EQUAL TO.	NEQ	1010 10	

ARITHMETIC OPERATORS

NAME	MNEMONIC	OP CODE
ADD	ADD	1011 01
SUBTRACT	SUB	1011 10
MULTIPLY	MUL	1111 00 0101
DIVIDE	DIV	1111 00 0110
MODULO	MOD	1111 00 0111
REVERSE SUBTRACT	RSUB	1111 10 1100

REVERSE DIVIDE	RDIV	1111 10 1101
REVERSE MODULO	RMOD	1111 10 1110
NEGATE	NEG	1111 01 0111
CONVERT TO DECIMAL	DEC	1111 1C 1000
CONVERT TO BINARY	BIN	1111 1C 1001

EXTENDED ARITHMETIC OPERATORS

NAME	MNEMONIC	OP CODE	ARGUMENTS
EXTENDED ADD	XADD	1111 11 1100 011	
EXTENDED SUBTRACT	XSUB	1111 11 1100 100	
EXTENDED MULTIPLY	XMUL	1111 11 1100 101	
EXTENDED DIVIDE	XDIV	1111 11 1100 110	
EXTENDED MODULO	XMOD	1111 11 1100 111	

LOGICAL OPERATORS

NAME	MNEMONIC	OP CODE
AND	AND	1111 0C 0001
OR	OR	1111 0C 0000
EXCLUSIVE-OR	EXOR	1111 0C 0010
NOT	NOT	1111 00 1011

STRING OPERATORS

NAME	MNEMONIC	OP CODE	
CONCATENATE	CAT	1100 11	
SUBSTRING ONE	SS1	1111 11 0100	T,V,C,L
SUBSTRING TWO	SS2	1111 0C 1000	T,V
SUBSTRING THREE	SS3	1010 00	T,V

STORE OPERATORS

NAME	MNEMONIC	OP CODE	ARGUMENTS
STORE DESTRUCTIVE	STOD	0010	
STORE NON-DESTRUCTIVE LEFT	SNDL	1010 11	
STORE NON-DESTRUCTIVE RIGHT	SNDR	1111 0C 0100	

CONSTRUCT DESCRIPTOR OPERATORS

NAME	MNEMONIC	OP CODE	ARGUMENTS
CONSTRUCT DES. BASE ZERO	CDBZ	1111 10 0100	DESCRIPTOR
CONSTRUCT DES. LOCAL DATA	CDLD	1110 0C	N, DES#1,..., DES#n
CONSTRUCT DES. FORMAL	CDFM	1111 01 0001	LL,E
CONSTRUCT DES. FORMAL CHECK	CDFC	1111 11 1101 000	LL,E,DES#1,..., DES#n
CONSTRUCT DES. FROM PREV.	CDPR	1110 1C	N, DES#1,..., DES#N

CONSTRUCT DES. FROM PREV. & ADD	CDAD	1110 01	N, DES#1,...,DES#n
CONSTRUCT DES. FROM PREV. & MULTIPLY	CDMP	1111 10 0101	N,DES#1,...,DES#r
CONSTRUCT DES. LEXIC LEVEL	CDLL	1111 10 0011	TYPE-LL-OC, DESCRIPTOR
CONSTRUCT DES. REMAPS	CDRM	1111 00 1111	DESCRIPTOR
CONSTRUCT DES. DYNAMIC	CDDY	1111 11 1110 000	TYPE

LOAD OPERATORS

NAME	MNEMONIC	OP CODE	ARGUMENTS
MAKE DESCRIPTOR	MDSC	1111 10 1010	
VALUE DESCRIPTOR	VDSC	1111 01 1000	
DESCRIPTOR	DESC	1100 10	TYPE-LL-OC
NEXT OR PREVIOUS ITEM	NPIT	1111 01 1101	V, TYPE-LL-OC
LOAD	L	1101 00	TYPE-LL-OC
LOAD ADDRESS	LA	0000	TYPE-LL-OC
LOAD ARRAY FIELD ADDR.	LAFA	1111 11 1111 011	TYPE, LENGTH
LOAD FIELD ADDRESS	LFA	1111 11 1111 001	TYPE, OFFSET, LENGTH, TYPE-LL-OC
LOAD FIELD ADDRESS FROM PREVIOUS	LFAP	1111 11 1111 010	TYPE, OFFSET, LENGTH
ARRAY LOAD VALUE	AL	1111 01 1100	TYPE-LL-OC
ARRAY LOAD ADDRESS	ALA	1101 01	TYPE-LL-OC
INDEXED LOAD VALUE	IL	1111 01 0000	TYPE-LL-OC
INDEXED LOAD ADDRESS	ILA	0001	TYPE-LL-OC
INDEXED LOAD FIELD ADDRESS	ILFA	1111 11 1111 000	TYPE, OFFSET, LENGTH

LOAD LITERAL	LIT	0100	TYPE, LENGTH, LITERAL
LOAD NUMERIC LITERAL	LITN	0011	LITERAL
LOAD NUMERIC ZERO	ZOT	0101	
LOAD NUMERIC ONE	ONE	011C	
REFER	REFR	1111 11 1111 100	

STACK OPERATORS

NAME	MNEMONIC	OP CODE	ARGUMENTS
BUMP VALUE STACK POINTER	BVSP	1111 10 1011	
DUPLICATE	DUP	1100 00	
DELETE	DEL	1111 0C 0011	
EXCHANGE	XCH	1011 0C	
FORCE VALUE STACK	FVS	1100 01	

PROCEDURE OPERATORS

NAME	MNEMONIC	OP CODE	ARGUMENTS
CALL	CALL	0111	TYPE-SEG-PAGE-DISP
IF THEN	IFTH	10C1	TYPE-SEG-PAGE-DISP
IF THEN ELSE	IFEL	1101 1C	ADDR TYPE,TYPE-SEG-PAGE-DISP
CASE	CASE	1111 01 0100	# OF ADDR, ADDR TYPE, TYPE-SEG-PAGE-DISP,...,TYPE-SEG-PAGE-DISP
UNDO	UNDO	1000	# OF LEVELS

UNDC CONDITIONALLY	UNDC	1111 01 0011	# OF LEVELS
RETURN	RTRN	1111 01 0101	# OF LEVELS
RETURN FORMAL CHECK	RTNC	1111 11 1101 001	# OF LEVELS, TYPE, LENGTH
EXIT	EXIT	1101 11	# OF LEVELS
CYCLE	CYCL	1110 11	DISPLACEMENT
MARK STACK	MKS	1011 11	
MARK STACK AND UPDATE	MKU	1111 01 1111	LL
ENABLE-DISABLE INTERRUPTS	EDI	1111 11 0101	V
EXIT-ENABLE INTERRUPTS	XTEI	1111 11 0110	V,# OF LEVELS
CO-ROUTINE ENTRY	CNTR	1111 11 1010 000	
CO-ROUTINE EXIT	CXIT	1111 11 1010 001	# OF LEVELS

SEARCH & SCAN OPERATORS

NAME	MNEMONIC	OP CODE	ARGUMENTS
REDUCE	RDUC	1111 11 1001 101	VARIANTS, TYPE-LL-OC
SEARCH SDL STACKS	SSS	1111 11 1110 001	
SEARCH LINKED LIST	SLL	1111 01 1010	COMPARE TYPE
SEARCH SERIAL LIST	SSL	1111 11 1000 000	COMPARE TYPE
SORT SEARCH	SSCH	1111 11 1011 100	
THREAD VECTOR	TVEC	1111 11 1011 001	
INITIALIZE VECTOR	IVEC	1111 11 1011 000	
SORT STEP DOWN	SSD	1111 11 1011 010	

SORT SWAP	SSWP	1111 11 1011 101	
SORT UNBLOCK	UBLK	1111 11 1011 011	
DELIMITED TOKEN	DTKN	1111 11 1001 CC1	TYPE-LL-OC, DEL1, DEL2
NEXT TOKEN	NTKN	1111 11 1001 000	TYPE-LL-OC, SEPARATOR, V
DEBLANK	DBLK	1111 11 1001 010	TYPE-LL-OC
CHARACTER FILL	CHFL	1111 11 1001 100	
TRANSLATE	XLAT	1111 11 1110 101	
FIND DUPLICATE CHARACTERS	FDUP	1111 11 1001 011	

MISCELLANEOUS OPERATORS

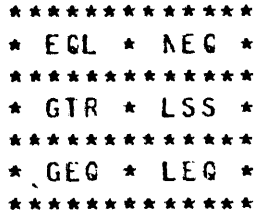
NAME	MNEMONIC	OP CODE	ARGUMENTS
TRANSFER MESSAGE	XFRM	1111 11 1010 010	DEST.VARIABLES SOURCE VARIABLE
HASH CODE	HASH	1111 11 1000 CC1	
SWAP	SWAP	1111 01 0110	
FETCH	FECH	1111 0C 1100	
FETCH AND SAVE	FECS	1111 11 1110 011	
DISPATCH	DISP	1111 01 1011	
HALT	HALT	1111 11 0010	
READ CASSETTE	RDCS	1111 01 0010	
LENGTH	LENG	1111 1C 0000	
LOAD SPECIAL	LSP	1111 01 1110	VARIANT
CLEAR ARRAY	CLR	1111 1C 0111	

COMMUNICATE	COMM	1111 10 0110	
REINSTATE	REIN	1111 10 0001	
FETCH CMP	FCMP	1111 10 0010	
DATA ADDRESS	ADDR	1111 01 1001	
SAVE STATE	SVST	1111 11 0001	
HARDWARE MONITOR	HMON	1111 11 0011	
OVERLAY	OVLY	1111 11 0000	
PROFILE	PRFL	1111 10 1111	ENTRY NUMBER
PARITY ADDRESS	PADR	1111 11 0111	
EXECUTE	EXEC	1111 11 1110 010	
COMMUNICATE WITH GISMO	CWG	1111 11 1110 110	
ADD TIMER	ADDT	1111 11 1100 000	
SUBTRACT TIMER	SUBT	1111 11 1100 001	

RELATIONAL OPERATORS

NAME	MNEMONIC	OP CODE	ARGUMENTS
EQUAL TO	EQL	1010 01	
LESS THAN	LSS	1111 01 1010	
LESS THAN OR EQUAL TO	LEQ	1111 00 1110	
GREATER THAN	GTR	1111 00 1001	
GREATER THAN OR EQUAL TO	GEQ	1111 00 1101	
NOT EQUAL TO	NEQ	1010 10	

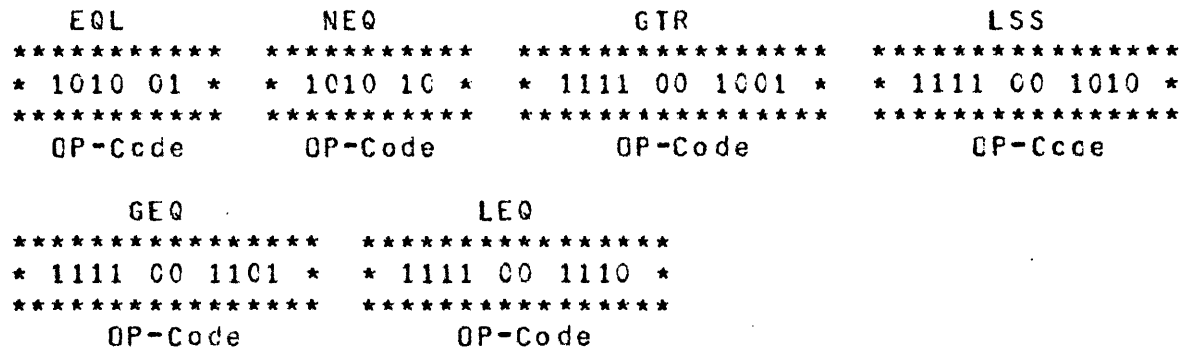
RELATIONAL OPERATORS



Syntax:

EQL	Equal to	(=)
NEQ	Not equal to	(_)
GTR	Greater than	(>)
LSS	Less than	(<)
GEQ	Greater than or equal to	(')
LEQ	Less than or equal to	(\)

Format:



a. Two operands are expected to be on top of the EVALUATION STACK. The lower operand is considered to be on the left side of the relation, while the top operand is considered to be on the right.

b. The relational operators do a comparison between two operands of any data type. The operands are removed from the stack and a self-relative descriptor of a 1-bit result is returned whose value is:

   1. When the condition is true - @(1)1@

   2. When the condition is false - @(1)0@

c. When both operands are FIXED, the operator does a true signed arithmetic compare.

d. When both operands are character strings, the compare is done from left to right, using blank fill on the right for the shorter string.

   For all other operand combinations leading zeros are supplied to the shorter of the two fields. No sign analysis is done and the operands are treated as positive magnitudes.

ARITHMETIC OPERATORS

NAME	MNEMONIC	OP CODE
ADD	ADD	1011 01
SUBTRACT	SUB	1011 10
MULTIPLY	MUL	1111 00 0101
DIVIDE	DIV	1111 00 0110
MODULO	MOD	1111 00 0111
REVERSE SUBTRACT	RSUB	1111 10 1100
REVERSE DIVIDE	RDIV	1111 10 1101
REVERSE MODULO	RMOD	1111 10 1110
NEGATE	NEG	1111 01 0111
CONVERT TO DECIMAL	DEC	1111 10 1000
CONVERT TO BINARY	BIN	1111 10 1001

ARITHMETICS

```
**************
* ADD * SUB *
**************
* MUL * DIV *
**************
* MOD *     *
**************
```

ADDITION

SUBTRACTION

MULTIPLICATION

DIVISION

MODULO

Syntax: ADD

SUB

MUL

DIV

MOD

Format:

```
    ADD          SUB           MUL              DIV
************  ************  ****************  ****************
* 1011 01 *  * 1011 10 *  * 1111 00 0101 *  * 1111 00 0110 *
************  ************  ****************  ****************
  OP-Code       OP-Code         OP-Code           OP-Code


       MOD
****************
* 1111 00 0111 *
****************
     OP-Code
```

Function:

a. Two operands are expected on the EVALUATION STACK.

   1. The bottom operand is considered to be on the "left" side of the operator.

   2. The top operand is considered to be on the "right".

b. The arithmetic operators perform 24-bit arithmetic on two operands. These operands may be of any data type.

c. Sign analysis will only be done if both operators are of type FIXED. With any other data type combinations, the magnitudes of the operands are evaluated.

d. For bit and character data, if the field is greater than 24 bits, only the low order 24 bits will be evaluated. When the field is less than 24 bits, zeros will be supplied on the left.

e. Both operands are cut back.

f. A 24 bit self-relative result is returned to the EVALUATION STACK. When both operands are type FIXED, the result will be type FIXED. In all other instances the result will be of type BIT.

g. ADD performs integer addition (+)

   SUB performs integer subtraction (-)

   MUL performs integer multiplication (*)

   DIV performs integer division (/)

h. DIV results in an integer value. Any remainder is truncated.

17/8 = 2

3/7 = 0

-7/3 =-2

i. The MOD operation is division resulting in the integer value of the remainder. It is evaluated by the following formula;

Y MOD Z = Y-((Y/Z)*Z), using integer value of h. above

For example:

7 MOD 3 = 7-((7/3)*3) = +1

-7 MOD 3 = (-7)-(((-7)/3)*3) = -1

3 MOD -7 = 3-((3/(-7))*(-7)) = 3

-3 MOD -7 = (-3)-(((-3)/(-7))*(-7)) = -3

Note that this is NOT the same as the conventional mathematical definition of MOD.

CONVERT TO BINARY

```
*******
* BIN *
*******
```

Syntax: BIN

Format:

```
****************
* 1111 10 1001 *
****************
     OP-Code
```

Function:

a. An operand of one (1) to eight (8) characters is expected on the EVALUATION STACK.

b. These characters are assumed to be numeric decimal digit characters.

c. The characters are treated as the decimal representation of an unsigned integer and are converted to the corresponding positive 24-bit binary number.

d. A self-relative descriptor, of type BIT is left on the EVALUATION STACK with the binary value.

CONVERT TO DECIMAL

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

Syntax: DEC

Format:

```
*****************
* 1111 1C 1CCC *
*****************
     OP-Code
```

Function:

a. Two operands are expected to be on the EVALUATION STACK.

   1. The top operand should yield a value of one (1) to eight (8).

      a) If not 1-8, a value of eight (8) will be used.

   2. The second operand should be 24 bits in length.

      a) If less than 24 bits, zero fill is provided on the left.

      b) If more than 24 bits, then only the low order 24 bits will be used.

b. The 24-bit value is assumed to be type BIT, i.e. unsigned.

c. Both operands are cut back.

d. Depending upon the value of the top descriptor on the

EVALUATION STACK a 1 to 8 character result is left on top of the EVALUATION and VALUE STACKS. These characters are the decimal representation of the 24-bit value.

e. Leading zeros in the result are not changed to blanks.

NEGATE

```
*******
* NEG *
*******
```

Syntax: NEG

Format:

```
*****************
* 1111 01 0111 *
*****************
     OP-Code
```

Function:

a. This operator pops an operand off the EVALUATION STACK, negates it, and pushes it back on top of the EVALUATION STACK as a FIXED, self-relative result which is the two's complement of the operand.

b. Operands of any type other than FIXED are treated as FIXED.

   1. Data items shorter than 24 bits are padded on the left with zeros (0).

   2.=Data items longer than 24 bits are left-truncated and treated as FIXED.

REVERSE ARITHMETICS

RSUB	RDIV
RMOD	

Syntax: RSUB

RDIV

RMOD

Format:

RSUB	RDIV	RMOD
1111 10 11C0	1111 10 11C1	1111 10 1110
OP-Code	OP-Code	OP-Code

Function:

a. These operators perform the same operation as their corresponding "forward" operators. The only difference is the order of the operands in the EVALUATION STACK.

1. Reverse subtract: The second operand in the EVALUATION STACK is subtracted from the operand on top of the stack.

2. Reverse divide: The second operand in the EVALUATION STACK is divided into the operand on top of the stack.

3. Reverse modulo: The second operand in the EVALUATION STACK is divided into the operand on top of the stack to obtain the residue.

EXTENDED ARITHMETIC OPERATORS

NAME	MNEMONIC	OP CODE	ARGUMENTS
EXTENDED ADD	XADD	1111 11 1100 011	
EXTENDED SUBTRACT	XSUB	1111 11 1100 100	
EXTENDED MULTIPLY	XMUL	1111 11 1100 101	
EXTENDED DIVIDE	XDIV	1111 11 1100 110	
EXTENDED MODULO	XMOD	1111 11 1100 111	

EXTENDED ARITHMETIC OPERATORS

XADD	XSUB
XMUL	XDIV
XMOD	

Syntax: XADD

XSUB

XMUL

XDIV

XMOD

Format:

XADD

1111	11	1100	011

OP-Code

XSUB

1111	11	1100	100

OP-Code

XMUL

1111	11	1100	101

OP-Code

XDIV

1111	11	1100	110

OP-Code

XMOD

1111	11	1100	111

OP-Code

Function:

a. Two operands are expected to be on the EVALUATION STACK.

b. The operands are popped from the EVALUATION STACK and the indicated operation is performed on the operands.

c. The operands are always treated as bit strings.

d. The result returned on the top of the EVALUATION STACK is non-self-relative, type BIT.

e. Addition/Subtraction

1. If the two operands are of different lengths, then the shorter is padded on the left with binary zeros.

2. The length of the sum/difference will be equal to the length of the longer operand.

f. Multiplication

1. The length of the product will be the sum of the lengths of the two operands.

g. Division/Modulo

1. The length of the result will be the length of the dividend.

2. For the Modulo operator, the dividend must be non-self-relative.

LOGICAL OPERATORS

NAME	MNEMONIC	OP CODE
AND	AND	1111 00 0001
OR	OR	1111 0C 0000
EXCLUSIVE-OR	EXOR	1111 0C 0010
NOT	NOT	1111 0C 1011

LOGICAL OPERATORS

AND	EXOR
CR	

Syntax: AND

OR

EXOR

Format:

AND	OR	EXOR
1111 00 0001	1111 00 0000	1111 00 0010
OP-Code	OP-Code	OP-Code

Function:

a. "NOT" is unary operator and is explained separately. All other logical operators expect two operands to be on the EVALUATION STACK.

b. The operands, regardless of their type, are operated upon bit by bit, starting from the right. When the operators are of unequal length the shorter one is padded on the left with zeros.

c. The length of the result is the length of the larger operand. When the result is 24 bits or less, the result is self-relative. on the VALUE STACK.

d. The result is always type BIT.

e. The two operands are cut back from the EVALUATION STACK and the result is pushed onto the EVALUATION STACK.

LOGICAL NOT

```
*******
* NOT *
*******
```

Syntax: NOT

Format:

```
****************
* 1111 00 1011 *
****************
     OP-Code
```

Function:

a. This operator expects one operand to be on the top of the EVALUATION STACK.

b. The result is of the same type and length as the operand, but each bit representing the result value is the one's complement of the corresponding bit of the operand.

c. If the operand is self-relative, the result will be self-relative, otherwise, the result will be non-self-relative.

STRING OPERATORS

NAME	MNEMONIC	OP CODE	
CONCATENATE	CAT	1100 11	
SUBSTRING ONE	SS1	1111 11 0100	T,V,C,L
SUBSTRING TWO	SS2	1111 00 1000	T,V
SUBSTRING THREE	SS3	1010 00	T,V

STRING CONCATENTATION

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

Syntax: CAT

Format:

```
***********
* 1100 11 *
***********
  OP-Code
```

Function:

a. This operator pops two operands off the EVALUATION STACK and generates a new descriptor that describes the concatentation of the two strings. The next-to-top operand must be non-self-relative, name-value bit on.

b. When the source data items are of type CHARACTER, the results will be type CHARACTER.

c. All other data type combinations will cause the result to be of type BIT.

SUBSTRING, ONE PARAMETER

```
*******
* SS1 *
*******
```

Syntax: <String Type Bit><Load Type Bit>
<Offset><Length>

Format:

```
*************************************************************
* 1111 11 0100 *     *       *         *                    *
*************************************************************
     OP-Code     *       *         *          *
                 *       *         *          *
                 *       *         *          *** 6 or 17 Bits
                 *       *         *              Specifies length
                 *       *         *
                 *       *         *
                 *       *         *** 6 or 17 Bits
                 *       *             Specifies offset
                 *       *
                 *       *
                 *       *** 1 Bit  0 = Load Address
                 *                  1 = Load Value
                 *
                 *
                 *** 1 Bit  0 = Length in bits
                           1 = Length in character
```

The Offset and Length fields are encoded as follows:

1. If the first bit in these fields is equal to (0) then 5 bits follow.

2. If the first bit in these fields is equal to (1) then 16 bits follow.

Function:

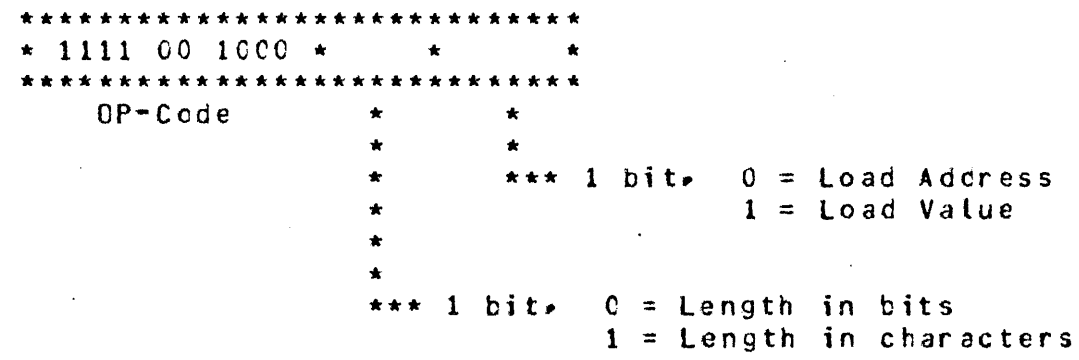
This operator functions the same as SS3 except that the offset and lergth fields are literals which follow in-line, rather than values on the EVALUATION STACK.

SUB-STRING, TWO PARAMETERS

```
*******
* SS2 *
*******
```

Syntax: SS2 <String Type Bit><Load Type Bit>

Format:



Function:

a. Two operands are expected to be found on the EVALUATION STACK.

   1. The top operand is the offset in bits or characters of the substring from the beginning of the string. If longer than 24 bits, only the low order 24 bits are used.

   2. The next operand is the string. It may be of any type. It must not be self-relative. If load type bit is 0, then it must be an address (i.e. name-value bit off).

b. The two operands are then cut back and a result is left on the EVALUATION STACK.

   1. Type:

a) String type bit = 0 then bit
String type bit = 1 then character

b) Load type bit = 0 then address
Load type bit = 1 then value (self-relative if length is 24 bits or less)

2. Length:

a) Length is equal to the original length of the string minus the offset.

3. Address:

a) Load type bit = 0 then the address in bits is equal to the old string access plus the offset.

b) Load type bit = 1 then the address is the address of the VALUE STACK if the substring length is greater than 24 bits.

1) If the load type bit is set and substring length is greater than 24 bits, the substring is loaded to the top of the VALUE STACK.

2) If the load type bit is set and the substring length is less than or equal to 24 bits the substring is loaded right justified into the address field of the top of the EVALUATION STACK.

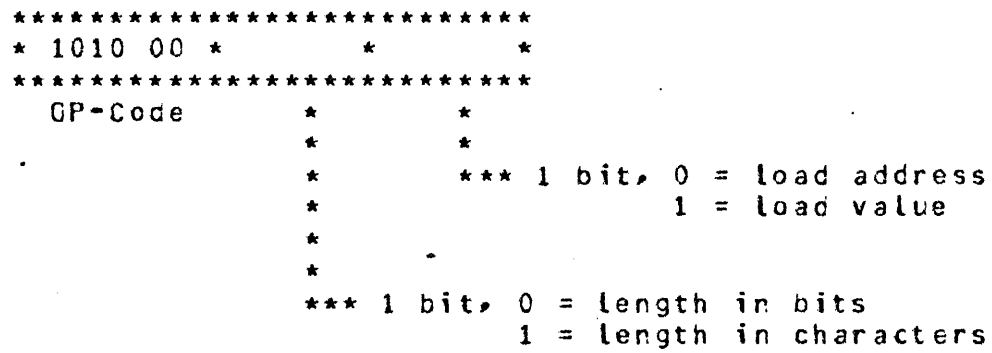
4. When the offset is greater than the length of the string, an error interrupt occurs.

SUB-STRING, THREE PARAMETERS

```
*******
* SS3 *
*******
```

Syntax: SS3 <String Type Bit><Load Type Bit>

Format:



Function:

a. Three operands are expected to be found or the EVALUATION STACK.

   1. The top operand is the length in bits or characters of the substring desired.

   2. The second operand is the offset in bits or characters from the beginning of the string.

   3. The third operand is the string. It may be of any type. It must not be self-relative. If load type bit is 0, then it must be an access (i.e. name-value bit off).

b. All three operands are cut back and a result is left on the EVALUATION STACK.

1. Type:

   a) String type bit = 0 then bit
      String type bit = 1 then character

   b) Load type bit = 0 then address
      Load type bit = 1 then value (self-relative if length is 24 bits or less)

2. Length:

   a) Length is equal to the length of the string minus the offset.

3. Address:

   a) Load type bit = 0 then the address in bits is equal to the old string address plus the offset.

   b) Load type bit = 1 then the address is the address of the top of the VALUE STACK if the substring length is greater than 24 bits.

      1) If the load type bit is set and the substring length is > 24 bits, the substring is loaded to the top of the VALUE STACK.

      2) If the load type bit is set and the substring is loaded right justified into the address field of the EVALUATION STACK.

4. When the offset plus the length is greater than the length of the string, an error interrupt occurs.

STORE OPERATORS

NAME	MNEMONIC	CP CODE	ARGUMENTS
STORE DESTRUCTIVE	STOD	0010	
STORE NON-DESTRUCTIVE LEFT	SNDL	1010 11	
STORE NON-DESTRUCTIVE RIGHT	SNDR	1111 0C 0100	

STORE NON-DESTRUCTIVE, DELETE LEFT

```
********
* SNDL *
********
```

Syntax: SNDL

Format:

```
************
* 1010 11 *
************
  OP-Code
```

Function:

a. This operator pops two operands off the top of the EVALUATION STACK.

b. The top (destination) operand must be non-self-relative and the name-value bit must be off, i.e. it must be an address.

c. It then copies the data described by the second (source) descriptor into the location described by the top (destination) descriptor.

d. When both the source and destination fields are of type CHARACTER, the data will be left justified in the destination field with either blank fill or truncation on the right.

e. Any other source-destination field type combinations yield right justified data in the destiration field with either zero fill or truncation on the left. This allows for type conversion and length truncation.

f. The source descriptor is then pushed back on to the EVALUATION STACK.

STORE NON-DESTRUCTIVE, DELETE RIGHT

```
********
* SNDR *
********
```

Syntax: SNDR

Format:

```
****************
* 1111 00 C100 *
****************
     OP-Code
```

Function:

a. This operator pops two operands cff the top cf the EVALUATION STACK.

b. The top (destination) operanc must be non-self-relative ard the name-value bit must be off, i.e. it must be an address.

c. It then copies the data described by the second (source) descriptor into the location cescribed by the top (destination) descriptor.

d. When both the source and destination fields are of type CHARACTER, the data will be left justifiec in the destination field with either blank fill or truncation on the right.

e. Any other source-destination field type combinaticns yield right justified data in the cestiration field with either zero fill or truncation on the left. This allows for type conversion and lergth truncation.

f. The destination descriptor is then pushed back on to the top of the EVALUATION STACK. (This is the only difference from SNDL).

STORE DESTRUCTIVE

```
********
* STOD *
********
```

Syntax: STOD

Format:

```
********
* 0010 *
********
```
OP-Code

Function:

a. This operator pops two operands off the top of the EVALUATION STACK.

b. The top (destination) operand must be non-self-relative and the name-value bit must be off, i.e. it must be an address.

c. It then copies the data described by the second (source) descriptor into the location described by the top (destination) descriptor.

d. When both the source and destination fields are of type CHARACTER, the data will be left justified in the destination field with either blank fill or truncation on the right.

e. Any other source-destination field type combinations yield right justified data in the destination field with either zero fill or truncation on the left. This allows for type conversion and length truncation.

CONSTRUCT DESCRIPTOR OPERATORS

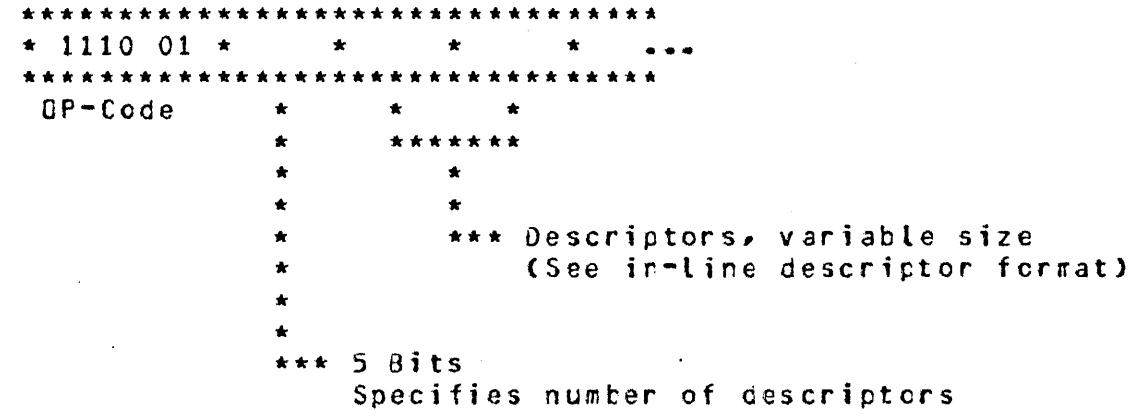
NAME	MNEMONIC	OP CODE	ARGUMENTS
CONSTRUCT DES. BASE ZERO	CDBZ	1111 1C 0100	DESCRIPTOR
CONSTRUCT DES. LOCAL DATA	CDLD	1110 0C	N, DES#1,..., DES#n
CONSTRUCT DES. FORMAL	CDFM	1111 01 0001	LL,E
CONSTRUCT DES. FORMAL CHECK	CDFC	1111 11 1101 CCC	LL,E,DES#1,..., DES#n
CONSTRUCT DES. FROM PREV.	CDPR	1110 1C	N, DES#1,..., DES#N
CONSTRUCT DES. FROM PREV. & ADD	CDAD	1110 01	N, DES#1,..., DES#n
CONSTRUCT DES. FROM PREV. & MULTIPLY	CDMP	1111 1C 0101	N,DES#1,..., DES#n
CONSTRUCT DES. LEXIC LEVEL	CDLL	1111 1C 0011	TYPE-LL-0C, DESCRIPTOR
CONSTRUCT DES. REMAPS	CDRM	1111 00 1111	DESCRIPTOR
CONSTRUCT DES. DYNAMIC	CDDY	1111 11 1110 CCC	TYPE

CONSTRUCT DESCRIPTOR PREVIOUS and ADD

```
********
* CDAC *
********
```

Syntax: CDAD <# Descriptors><Descriptor><Descriptor>...

Format:



Function:

a. The number of descriptors indicates the number of in-line descriptors. These will be used to build descriptors on the NAME STACK.

b. The descriptors may be simple or array.

c. The address portion is generated from the "filler" field in the descriptor and information from the previous entry on the NAME STACK.

d. The following formula is used:

$$A' = A+L+F$$

1. A' - is the new address part.

2. A - is the address part of the previous descriptor generated on the NAME STACK.

3. L - is the length part of the previous descriptor.

4. F - is the "filler" field in the descriptor if it is present.

e. The operator must be able to finc the address part of the previous entry whether it is a simple cr array descriptor.

f. The new and previous descriptors canrct be paged array descriptors.

CONSTRUCT DESCRIPTOR BASE ZERO

```
********
* CDBZ *
********
```

Syntax: CDBZ <descriptor>

Format:

```
****************************
* 1111 10 0100 *           *
****************************
     OP-Code          *
                      *
                      *
                      *** Descriptor, variable size
                          (See in-line descriptor format)
```

Function:

a. A descriptor is generated on the NAME STACK with zero (0) in the address field and other fields specified by this in-line descriptor.

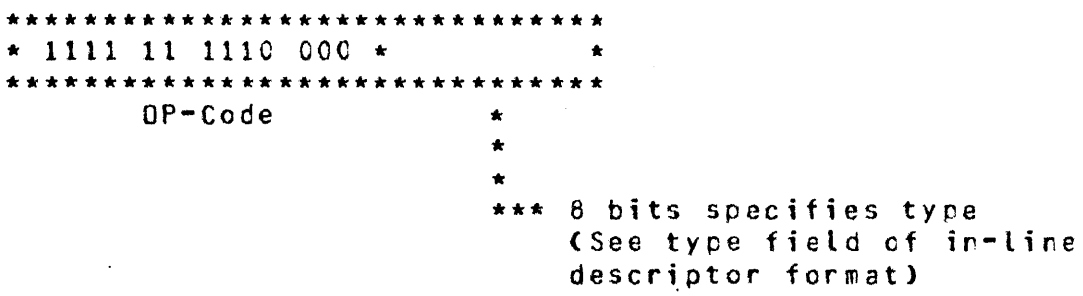
b. Paged array descriptors are not allowed.

CONSTRUCT DESCRIPTOR DYNAMIC

```
********
* CDDY *
********
```

Syntax: CDDY <Type>

Format:



Function:

a. This operator constructs descriptors for dynamic arrays and dynamic character strings or bit strings.

b. The type field follows the op-code, but the length and number of entries will be operands on the EVALUATION STACK.

c. The descriptor will be marked as non-self-relative, and, if array, will be marked contiguous.

d. The procedure to build the descriptor is as follows:

   1. The type field is placed in the NAME STACK.

   2. The type field is tested for simple or array type.

3. For simple items:

   a) Item length is the only entry on the EVALUATION STACK.

   b) The length is oopped off the EVALUATION STACK and placed in the NAME STACK.

   c) The VALUE STACK POINTER is used as the address.

   d) The VALUE STACK POINTER is updated by adding the length to it.

4. For array items

   a) Length and number of entries will be in the EVALUATION STACK.

   b) The length is popped off the EVALUATION STACK and placed in the NAME STACK.

   c) Length between entries is the same as length.

   d) The number of entries is popped off the EVALUATION STACK and placed in the NAME STACK.

   e) The VALUE STACK POINTER is used as the address.

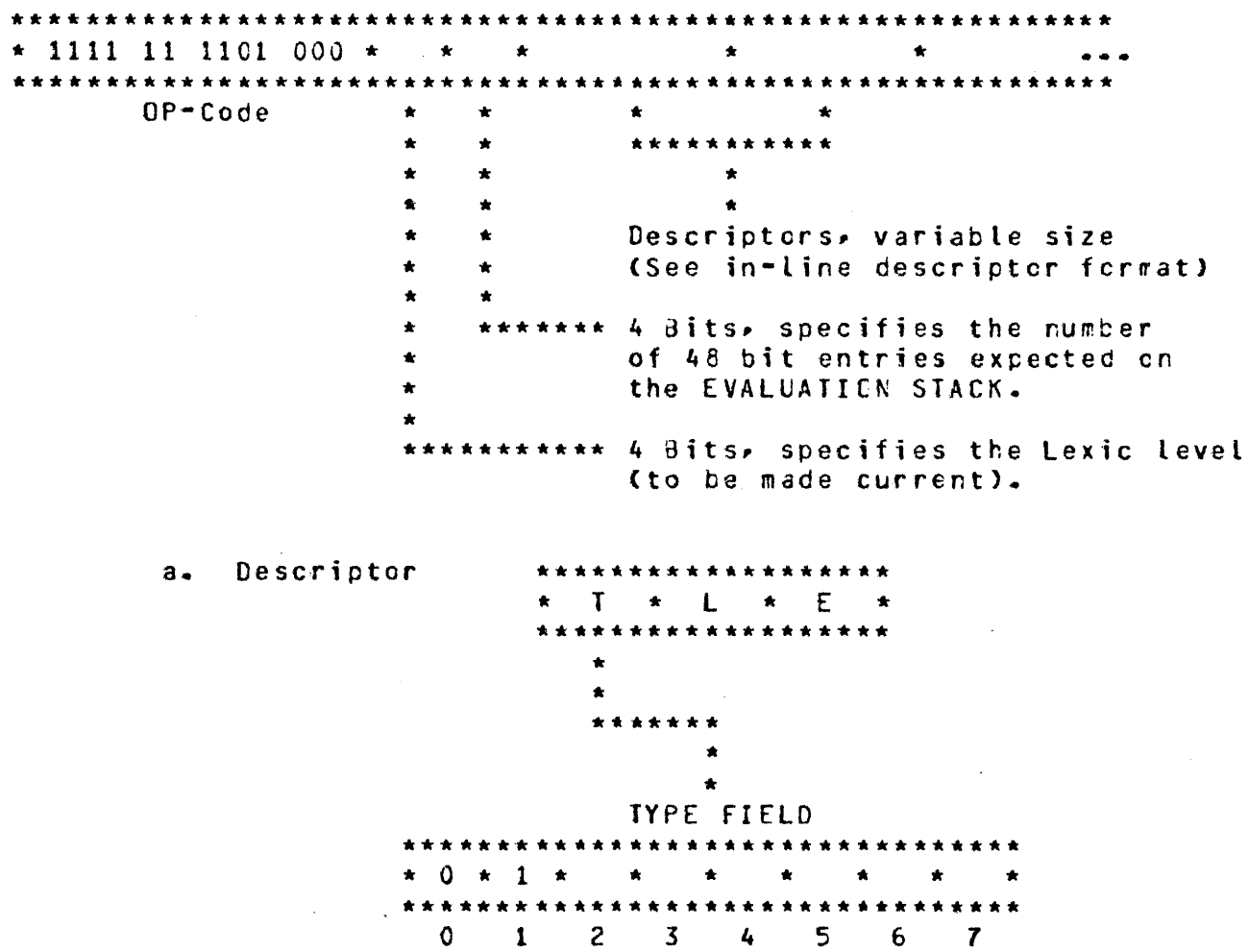
   f) The VALUE STACK POINTER is updated by multiplying the number of entries by the length and adding the result to the VALUE STACK POINTER.

CONSTRUCT DESCRIPTOR, FORMAL CHECK

```
********
* CDFC *
********
```

Syntax: CDFM <Lexic Level><#Descriptors>
<Descriptor><Descriptor> ....

Format:



a. Descriptor

Bits: 0 = Always 0

1 = Always 1

2 = If array then 1

3 = If array then 1, specifies that length between entries is equal to the length of entries

4,5= 00 BIT

01 FIXED

10 CHARACTER

11 VARYING

6 = If array bound varying then 1

7 = If length of data varying then 1

L = Length field - (6 or 17 bits). Appears only if type bit 7 is equal to zero (0).

E = Number of entries for array - (6 or 17 bits) Appears only if type bit 6 = 0 and type bit 2 = 1.

Depending on the conditions in the type field, different fields appear in the in-line descriptor;

Kind
Array (A)
Simple (S)
Length varying (LV)
Array bound varying (ABV)

THEN:

KIND	LV	ABV	FIELDS THAT APPEAR
S	FALSE	---	T, L
S	TRUE	---	T
A	FALSE	FALSE	T, L, E
A	FALSE	TRUE	T, L
A	TRUE	FALSE	T, E
A	TRUE	TRUE	T

Function: Refer to CONTROL STACK Mechanism

a. CS.ENTERED.LL(CSP-1) := CURRENT.LL := LL;
DISPLAY(CURRENT.LL) := CURRENT.NSP;

1. A previously executed MKS has initially set up CONTROL.STACK(CSP-1).

2. Since the format parameters have not been put on the NAME STACK, CURRENT.NSP has not yet been bumped.

b. Descriptors for the actual parameters are expected to be on the EVALUATION STACK.

c. The descriptors in the instruction are matched to descriptors in the EVALUATION STACK, starting with the last descriptor in the EVALUATION STACK and matching it with the first descriptor in the instruction.

d. The descriptors, after comparison, are loaded to the NAME STACK. The EVALUATION STACK is cutback. The VALUE STACK is unchanged.

e. When array bound varying is equal to 1, the array

bound is taken from the actual parameter.

f. When length varying is equal to 1, the length is taken from the actual parameter.

g. When the data type is varying, the data type is taken from the actual parameter.

h. When an array the actual array need not be contiguous. If it is, the formal and actual parameters must match identically. If there is a mismatch, an error interrupt occurs.

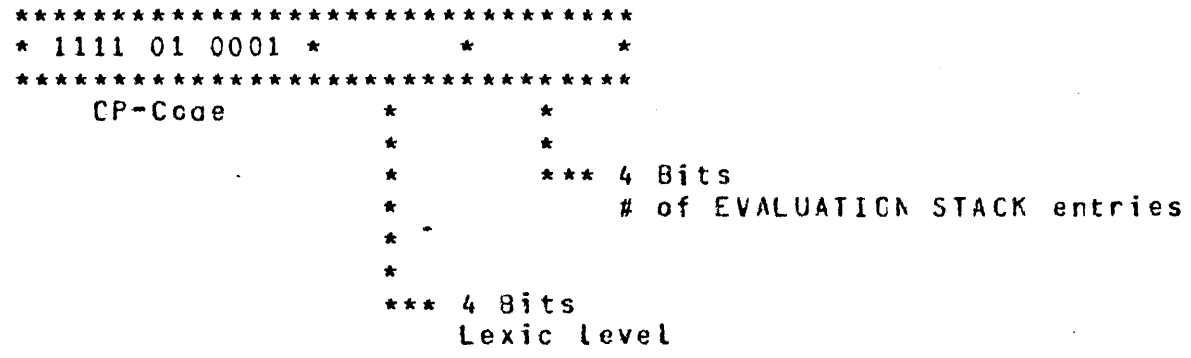
i. For paged arrays, both the paged array bit and the page subscript size must be copied.

CONSTRUCT DESCRIPTOR FORMAL

```
********
* CDFM *
********
```

Syntax: CDFM <Lexic Level><# of ES entries>

Format:



Function: Refer to CONTROL STACK Mechanism

a. CS.ENTERED.LL(CSP-1) := CURRENT.LL := LL;
DISPLAY(CURRENT.LL) := CURRENT.NSP;

1. A previously executed MKS has initially set up CONTROL.STACK(CSP-1)

2. Since the actual parameters have not yet been put on the NAME STACK, CURRENT.NSP has not yet been bumped.

b. Descriptors for the actual parameters are expected to be on the EVALUATION STACK.

c. The number (#) of EVALUATION STACK entries specified is transferred to the NAME STACK. (An entry is always

48 bits; therefore, an array passed as a parameter will be considered as two 48-bit entries).

d. If the name value bit is on, it is turned off.

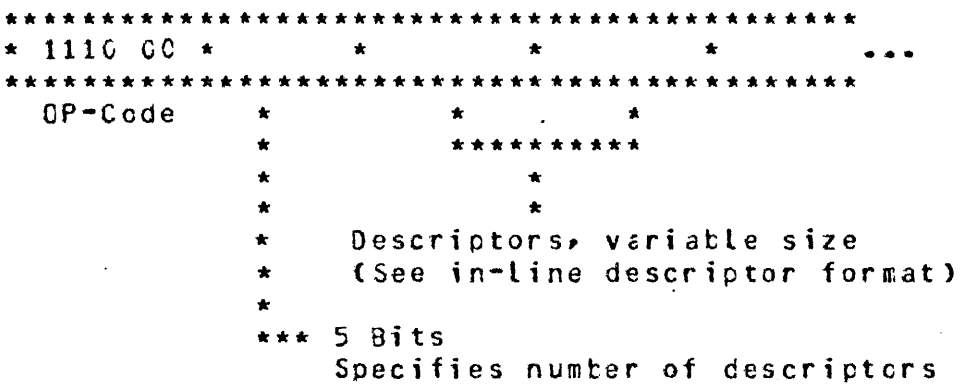
e. CDFM employs no form of parameter checking.

CONSTRUCT DESCRIPTOR LOCAL DATA

```
********
* CDLD *
********
```

Syntax: CDLD <# Descriptors><Descriptor><Descriptor>...

Format:



Function:

a. The number of descriptors indicates the number of in-line descriptors. These will be used as models to build descriptors on the NAME STACK.

b. The descriptors may be simple or array.

c. The filler option of the in-line descriptor format can never be present.

d. Array descriptors cannot have the length between entries option present.

e. When paged array is not indicated

1. If the descriptor is to be self-relative, its value (address field) will be zero.

2. If non-self-relative, the current value of the VALUE STACK POINTER is used as the address field of the generated descriptor. The VALUE STACK POINTER is then increased by the total length of the data item.

f. When a paged array is indicated.

1. The page subscript size option will be present.

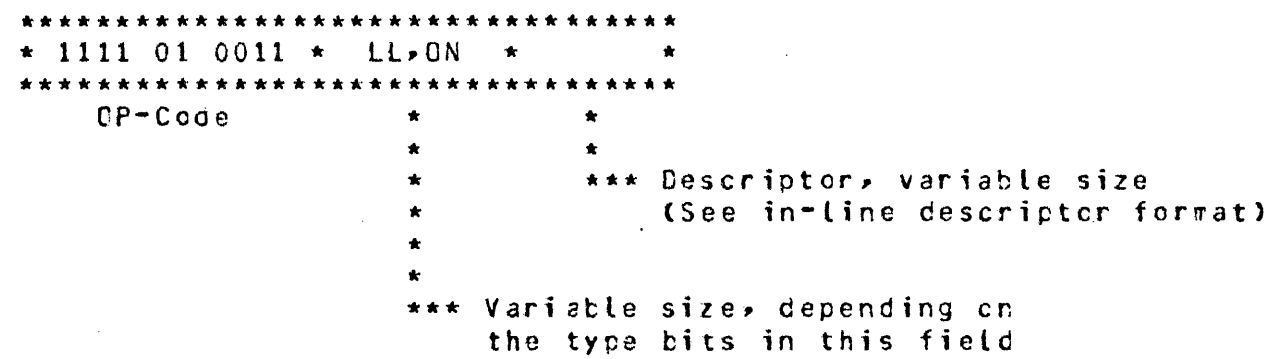
2. The address field of the generated descriptor is set to 0.

CONSTRUCT DESCRIPTOR LEXIC LEVEL

```
********
* CDLL *
********
```

Syntax: CDLL <Data Address><Descriptor>

Format:



Function:

a. This operator builds a descriptor on the NAME STACK using the in-line descriptor as a model.

b. The data address is used to find a data descriptor in the NAME STACK, its address field is then used for the address field of the new descriptor.

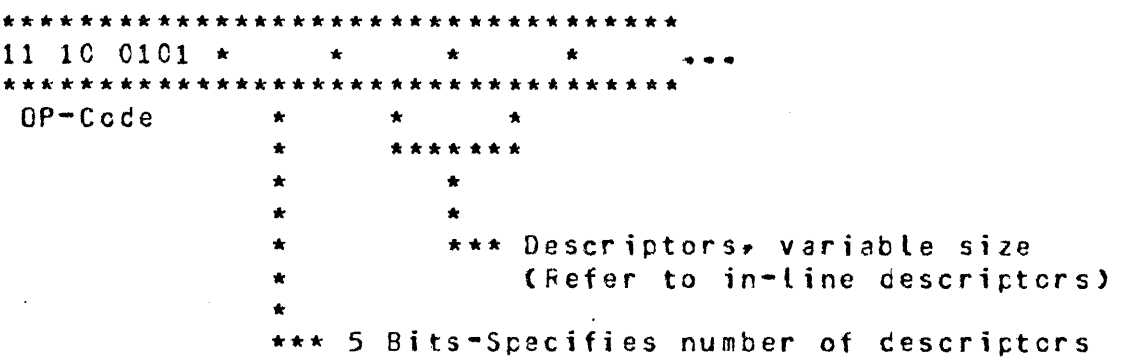
c. The data address will never reference a paged array descriptor.

CONSTRUCT DESCRIPTOR PREVIOUS & MULTIPLY

```
********
* CDMP *
********
```

Syntax: CDMP <#Descriptors><Descriptor><Descriptor>....

Format:



Function:

a. The number of descriptors indicates the number of in-line descriptors. These will be used to build descriptors on the NAME STACK.

b. The descriptors may be simple or array.

c. The address field is calculated using the following formula:

$$A' = A + L + F + ((\#E-1) * LB)$$

1. A' - is the new address part.

2. A - is the address part of the previous entry.

3. L - is the length part of the previous entry.

4. F - is the "filler" field in the descriptor, if it is present.

5. #E - is the "number of entries" part of the previous entry.

6. LB - is the "length between" part of the previous entry.

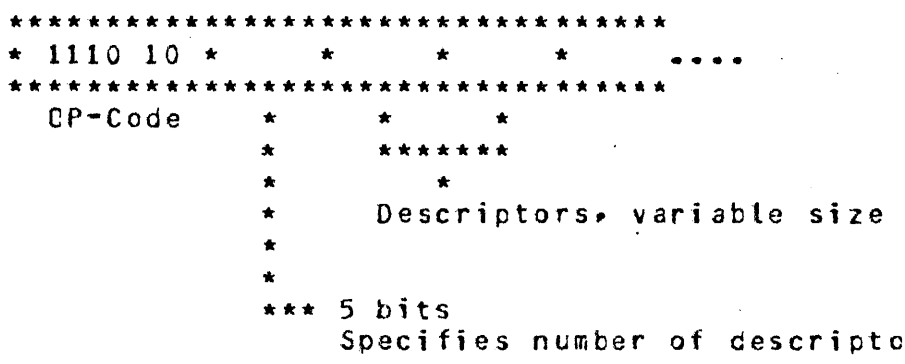
d. The previous entry is always assumed to be of type array. It will never be paged.

CONSTRUCT DESCRIPTOR FROM PREVIOUS

```
********
* CDPR *
********
```

Syntax: CDPR <#Descriptors><Descriptor><Descriptor> ....

Format:



Function:

a. The number of descriptors indicates the number of in-line descriptors. These will be used to build descriptors on the NAME STACK.

b. The descriptors may be simple or array.

c. The address portion is generated from the "filler" field in the descriptor and information from the previous entry on the NAME STACK.

d. The following formula is used:

$$A' = A+F$$

1. A' - is the new address part

2. A - is the address part of the top entry or the NAME STACK

3. F - is the "filler", field in the descriptor if it is present.

e. The previous descriptor built may be either simple or array.

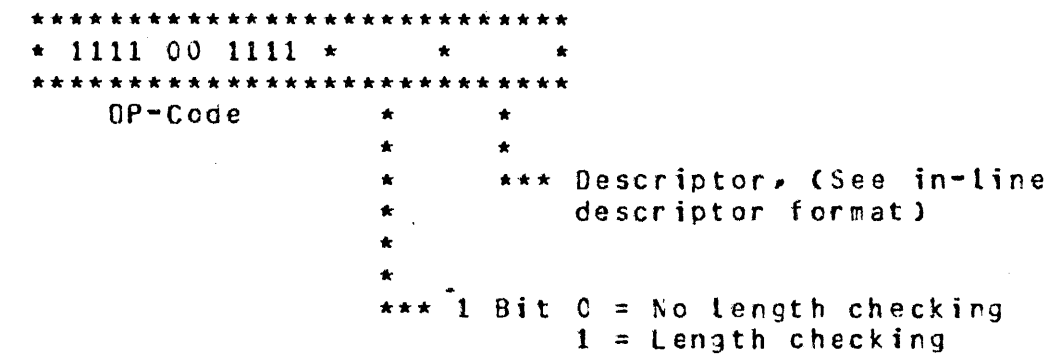
f. The new and previous data descriptors cannot be paged arrays.

CONSTRUCT DESCRIPTOR REMAPS

```
********
* CDRM *
********
```

Syntax: CDRM <Length-Check Variant><In-Line Descriptor>

Format:



Function:

a. This operator builds a descriptor on the NAME STACK by using:

   1. The in-line descriptor information

   2. An address obtained from the address field of the descriptor on top of the EVALUATION STACK.

b. When the length check variant is set (1) ther the length in the in-line descriptor is compared to the length field of the descriptor on the EVALUATION STACK. If it is greater a run time error will be signalled.

c. The descriptor on the EVALUATION STACK is removed.

LOAD OPERATORS

NAME	MNEMONIC	OP CODE	ARGUMENTS
MAKE DESCRIPTOR	MDSC	1111 10 1010	
VALUE DESCRIPTOR	VDSC	1111 01 1000	
DESCRIPTOR	DESC	1100 10	TYPE-LL-OC
NEXT OR PREVIOUS ITEM	NPIT	1111 01 1101	V, TYPE-LL-OC
LOAD	L	1101 00	TYPE-LL-OC
LOAD ARRAY FIELD ADDRESS	LAFA	1111 11 1111 011	TYPE, LENGTH
LOAD FIELD ADDRESS	LFA	1111 11 1111 001	TYPE,OFFSET,LENGTH TYPE-LL-OC
LOAD FIELD ADDRESS FROM PREVIOUS	LFAP	1111 11 1111 010	TYPE, OFFSET, LENGTH
LOAD ADDRESS	LA	0000	TYPE-LL-OC
ARRAY LOAD VALUE	AL	1111 01 1100	TYPE-LL-OC
ARRAY LOAD ADDRESS	ALA	1101 01	TYPE-LL-OC
INDEXED LOAD VALUE	IL	1111 01 0000	TYPE-LL-OC
INDEXED LOAD ADDRESS	ILA	0001	TYPE-LL-OC
INDEXED LOAD FIELD ADDRESS	ILFA	1111 11 1111 000	TYPE, OFFSET, LENGTH
LOAD LITERAL	LIT	0100	TYPE, LENGTH, LITERAL

LOAD NUMERIC LITERAL	LITN	0011	LITERAL
LOAD NUMERIC ZERO	ZOT	0101	
LOAD NUMERIC ONE	ONE	0110	
REFER	REFR	1111 11 1111 100	

ARRAY LOAD VALUE

```
******
* AL *
******
```

Syntax: AL <Data Address>

Format:

```
*************************
* 1111 01 1100 * LL,ON *
*************************
    OP-Code          *
                     *
                     *** Variable size depending upon type
                         bits in this field
```

Function:

a. The Lexic Level, Occurrence Number pair (Data Address) is used to address an entry in the NAME STACK.

   1. NAME STACK(DISPLAY(LL)+ON)

b. The low-order 24 bit field of the subscript on top of the EVALUATION STACK is compared to the "number of entries" field of the descriptor. If greater than or equal an invalid subscript error occurs.

c. If the paged array bit of the descriptor is off (not paged) then:

   1. The array descriptor is subscripted and a simple descriptor is generated on the top of the EVALUATION STACK, describing a copy of the value of the selected element of the array. This descriptor will be self-relative if the length of

the value is less than 24 bits, otherwise it will be non-self-relative, its name-value bit on, and the value on top of the VALUE STACK.

d. Paged arrays only.

1. If the address field of the descriptor is 0, (page table not yet allocated) then:

   a) The address of the current instruction (AL or ALA) is pushed onto the PROGRAM POINTER STACK.

   b) A copy of the subscript is pushed onto the EVALUATION STACK.

   c) The base-relative bit address (in the NAME STACK) of the paged array descriptor is pushed onto the EVALUATION STACK.

   d) The M-machine state is saved (in base-relative form) in RS.M.MACHINE.

   e) The segment, whose segment number is given in RS.INTRINSICS.LOC, is entered at displacement 0. (This is the memory management intrirsic).

   f) Upon exit, execution will begin at Step a.

2. The address field points to the first entry in the page table. The subscript is shifted to the right by the number of bits indicated by the page subscript size. The resulting number (the page table subscript) is used to access the page table entry.

3. If the presence bit in the page table entry is off then:

   a) The address of the current instruction is pushed onto the PROGRAM POINTER STACK.

   b) A copy of subscript is pushed onto the EVALUATION STACK.

c) The base-relative bit address of the page table entry is pushed on to the EVALUATION STACK.

d) Execution goes to Step d.1.d). (The memory manager is entered).

4. The address field of the page table entry points to the page. The number of bits indicated by page subscript size is extracted from the low order bits of the subscript.

5. This number (page subscript) is used to subscript into the page. A simple descriptor is generated on the top of the EVALUATION STACK as in Step c.

ARRAY LOAD ADDRESS

```
*******
* ALA *
*******
```

Syntax: ALA <Data Address>

Format:

```
********************
* 1101 01 * LL,ON *
********************
 OP-Code       *
               *
               *** Variable size, depending upon the
                   type bits in this field.
```

Function:

The execution of ALA is identical to that of array load, except that the resulting descriptor on the EVALUATION STACK is always non-self-relative and points to the array element itself rather than a copy. (See step c.1 of AL).

DESCRIPTOR

```
********
* DESC *
********
```

Syntax: DESC <Data Address>

Format:

```
********************
* 11C0 1C * LL,ON *
********************
    OP-Code   *
              *
              *** Variable size cepending upon
                  type bits in this field.
```

Function:

a. The Lexic Level, Occurrence Number pair (Data Address) locates a descriptor in the NAME STACK. The descriptor may be either simple or array.

b. A descriptor (48 bits) is generated on the EVALUATION STACK that points to the descriptor in the NAME STACK. The length field of the new descriptor will be 48 or 96 depending on the type of the descriptor in the NAME STACK. Its type will be BIT.

INDEXED LOAD VALUE

```
******
* IL *
******
```

Syntax: IL <Data Address>

Format:

```
************************
* 1111 01 0000 * LL,ON *
************************
    OP-Code          *
                     *
                     *** Variable size depending on the
                         type bits in this field.
```

Function:

a. The Lexic Level, Occurence Number pair (Data Address) locates a descriptor in the NAME STACK.

b. The descriptor must be non-self-relative.

c. When the descriptor is non-array, then,

   1. An operand (which is the index) is taken off the top of the EVALUATION STACK. A copy of the descriptor in the NAME STACK is then pushed onto the EVALUATION STACK and its address field is incremented by the index.

   2. The resulting address descriptor is then made a value by copying the data to the top of the VALUE STACK, changing the address field and setting the name-value bit.

d. When the descriptor is an array descriptor,

   1. Same as case c. above, except only the first 48 bits of the NAME STACK descriptor are picked up and the array bit is turned off in the type field.

   2. The array descriptor cannot be type PAGED.

INDEXED LOAD ADDRESS

```
*******
* ILA *
*******
```

Syntax: ILA <Data Address>

Format:

```
*****************
* 0001 * LL.ON *
*****************
OP-Code     *
            *
            *** Variable size depending on the
                type bits in this field.
```

Function:

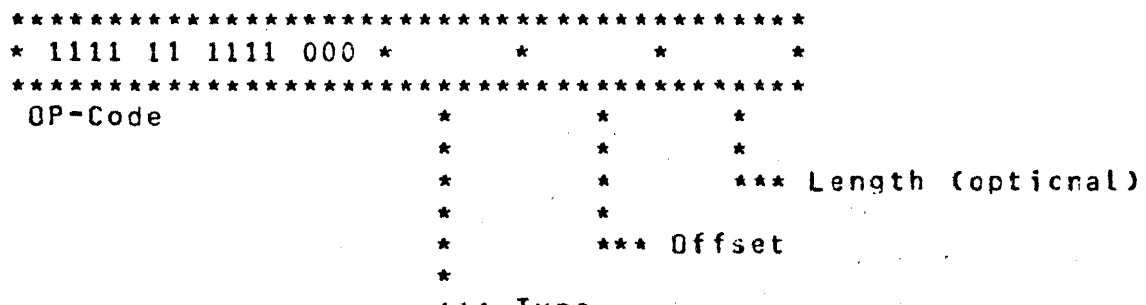
Indexed load address functions the same as irdexed load value; the only difference being that the resultant address descriptor is not converted to a value (Step c.2 of IL is omitted).

INDEXED LOAD FIELD ADDRESS

```
********
* ILFA *
********
```

Syntax: ILFA <type><offset>[<length>]

Format:



Function:

Using the value on top of the EVALUATION STACK as the base area, the user loads a subfield to the EVALUATION STACK by SUBBITting according to <offset> and <length> and builds type as specified by <type>. This S-OP is only used for indexing layouts (as REMAPS BASE items) and should eventually be phased out.

1. <TYPE> is 3 bits encoded as follows:

0 BIT	3 BIT (1)
1 FIXED	4 BIT (24)
2 CHARACTER	5-7 UNDEFINED

2. <OFFSET> is 9 or 17 bits, depending on the first bit.

3. <LENGTH> is 6 or 17 bits, depending on the first bit. <LENGTH> is not present if the type is FIXED, BIT (1), or BIT (24).

LOAD VALUE

```
*******
* L   *
*******
```

Syntax: L <Data Address>

Format:

```
********************
* 1101 00 * LL,ON *
********************
  OP-Code      *
               *
               *** Variable size, depending on
                   type bits in this field.
```

Function:

a. The Lexic Level, Occurrence Number pair (data address) is used to address the NAME STACK.

   1. NAMESTACK(DISPLAY((LL)+ON)

   2. The descriptor at this location must not be an array descriptor.

b. When the descriptor is self-relative or non-self-relative with a length less than or equal to 24, a self-relative descriptor is put on the EVALUATION STACK.

c. When the descriptor is non-self-relative with a length greater than 24:

   1. The descriptor is copied to the top of the EVALUATION STACK with the address field mocified to point to the top of the VALUE STACK.

   2. The data item is moved to the top of the VALUE STACK.

   3. The VALUE STACK POINTER is bumped by the size of the data items.

LOAD ADDRESS

```
******
* LA *
******
```

Syntax: LA <Data Address>

Format:

```
****************
* 0000 * LL,ON *
****************
OP-Code     *
            *
            *** Variable size, depending on the
                type bits in this field.
```

Function:

a. The Lexic Level, Occurence Number pair (Data Address) is used to address an entry in the NAME STACK.

   1. NAME STACK(DISPLAY(LL)+ON)

b. If the descriptor at that location is ar array type the 96 bit descriptor is copied to the EVALUATION STACK.

c. If the descriptor at that location is a non-array type then:

   1. If the data item is self-relative then build a descriptor on the EVALUATION STACK using the type and length from the NAME STACK and the address of the leftmost bit of the data item. Change the type of the new descriptor to non-self-relative. The data itself remains in the address portion of the descriptor in the NAME STACK.

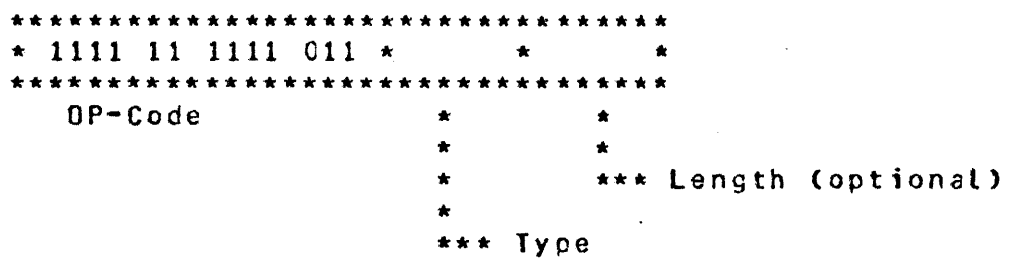
   2. If the data item is non-self-relative then the 48 bit descriptor is copied to the EVALUATION STACK.

LOAD ARRAY FIELD ADDRESS

```
********
* LAFA *
********
```

Syntax: LAFA <type> [<length>]

Format:



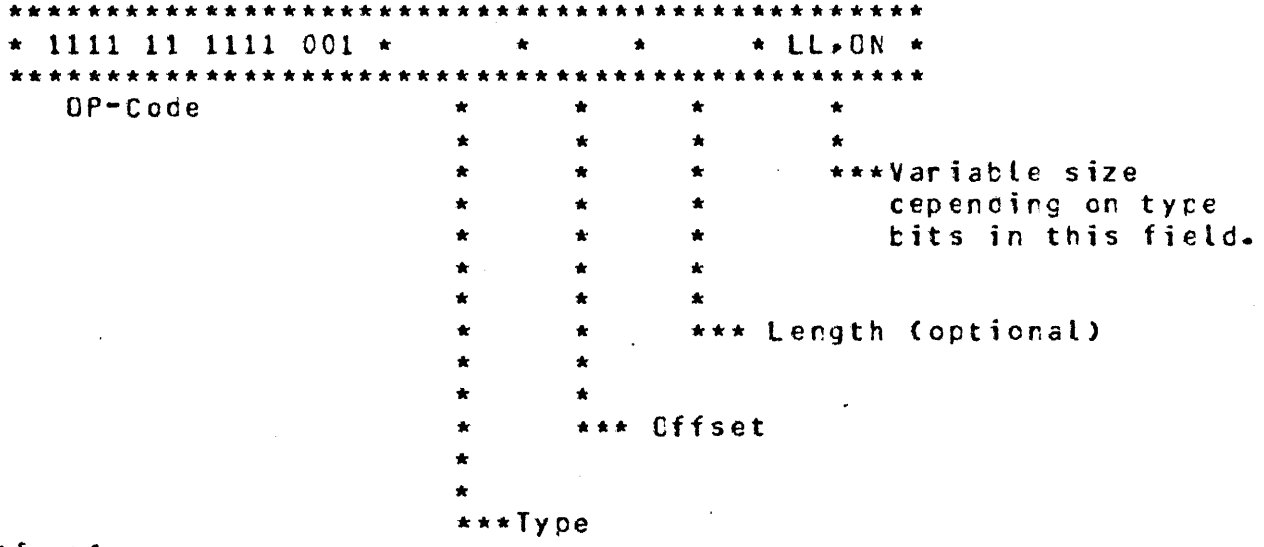
Function:

The top operand on the EVALUATION STACK is an array subscript; the next-to-top operand is a scaler string which is to be treated as an array with each element's type and length given by <type> and <length>. Load a descriptor of an element of this implied field, using the given subscript. Check array bounds by using the length field of an array area descriptor.

LOAD FIELD ADDRESS

```
*******
* LFA *
*******
```

Syntax: LFA <type><offset>[<length>]<data address>
Format:



Function:

Using the descriptor at <data address> to describe the base area the user loads a subfield tc the EVALUATION STACK by SUBBITting according to <offset> and <length> and builds type as specifieo by <type>.

1. <Type> is 3 bits, encoded
   0 BIT 3 BIT (1)
   1 FIXED 4 BIT (24)
   2 CHARACTER 5-7 UNDEFINED
   etc.

2. <Offset> is 9 or 17 bits, depending on the first bit.

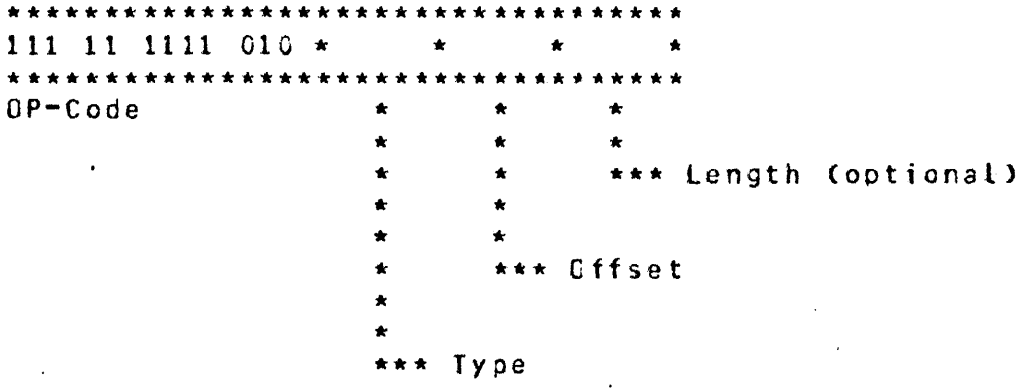
3. <Length> is 6 or 17 bits, depending cn the first bit. <Length> is not present if the type is FIXED, BIT (1), or BIT (24).

LOAD FIELD ADDRESS FROM PREVIOUS

```
********
* LFAP *
********
```

Syntax: LFAP <type><offset>[<length>]

Format:



Function:

Using the descriptor on top of the EVALUATION STACK to describe the base area, the user loads a subfield to the EVALUATION STACK by SUBBITting according to <offset> and <length> and builds type as specified by <type>.

1. <Type> is 3 bits encoded as follows:

0 BIT	3 BIT (1)
1 BIT	4 BIT (24)
2 BIT	5-7 UNDEFINED

2. <Offset> is 9 or 17 bits, depending on the first bit.

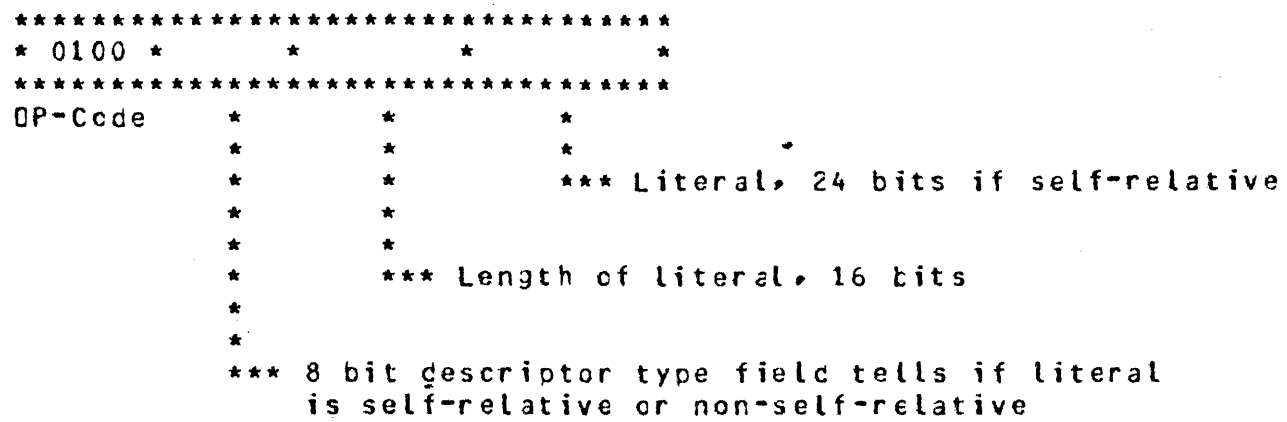
3. <Length> is 6 or 17 bits, depending on the first bit. (<Length> is not present if the type is FIXED, BIT(1), or BIT(24).)

LOAD LITERAL

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

Syntax: LIT <Type><Length><Literal>

Format:



Function:

This operator loads a constant to the top of the EVALUATION STACK.

1. When the literal is self-relative (less than or equal to 24 bits) the operator pushes the type, length, and literal onto the top of the EVALUATION STACK.

2. When the literal is non-self-relative, the type, length and VALUE STACK pointer are moved to the top of the EVALUATION STACK and the literal is placed on top of the VALUE STACK.

3. For literals less than twenty four bits in length the <literal> field must be twenty four bits long in the code file even though the <length> field specifies that the actual data is less than twenty four bits in length.

LOAD NUMERIC LITERAL

```
********
* LITN *
********
```

Syntax: LITN <Literal>

Format:

```
****************
* 0011 *       *
****************
OP-Code     *
            *
            *** 10 Bit Literal
```

Function:

This operator loads a 10 bit literal to the tcp of the EVALUATION STACK as a FIXED, self-relative data item.

MAKE DESCRIPTOR

```
********
* MDSC *
********
```

Syntax: MDSC

Format:

```
**********************
* 1111 10 1010 *
**********************
     OP-Code
```

Function:

a. A descriptor is expected to be on the EVALUATION STACK.

b. This descriptor must describe another descriptor.

c. This second descriptor (48 or 96 bits) is copied to the EVALUATION STACK after cutting back the original descriptor.

NEXT OR PREVIOUS ITEM

```
********
* NPIT *
********
```

Syntax: NPIT <Next or Previous Bit><Data Address>

Format:

```
*********************************
* 1111 01 1101 *     * LL,ON *
*********************************
    OP-Code        *       *
                   *       *
                   *       *** Variable size, depencing on
                   *           type bits in this field.
                   *
                   *  -
                   *** 1 Bit
                       0 => Previous
                       1 => Next
```

Function:

a. The Lexic Level, Occurrence Number pair (Data Address) describes a non-self-relative, non-array descriptor at

NAMESTACK (DISPLAY(LL)+ON)

```
********************
*  T  *  L  *  A  *
********************
```

b. The descriptor is then modified

```
********************
*  T' *  L' *  A' *
********************
```

1. Previous

   a) T' = T
      L' = L
      A' = A-L

2. Next

   a) T' = T
      L' = L
      A' = A+L

c. The modified descriptor replaces the old descriptor and a copy is also placed on the EVALUATION STACK.

LOAD NUMERIC ONE

```
*******
* ONE *
*******
```

Syntax: ONE

Format:

```
********
* 0110 *
********
OP-Code
```

Function:

This operator causes a FIXED, self-relative descriptor containing a one (1) to be placed cr the EVALUATION STACK.

REFER

```
********
* REFR *
********
```

Syntax: REFR <length>

Format:

```
****************************
* 1111 11 1111 100 *       *
****************************
 OP-Code             *
                     *
                     ***Length (6 or 17 bits depending on the
                        first bit).
```

Function:

Changes the descriptor on the top of the EVALUATION STACK to have the address of the second operand on the EVALUATION STACK. The length of the second operand is compared to the length in-line. If not equal, an exception is given.

VALUE DESCRIPTOR

```
********
* VDSC *
********
```

Syntax: VDSC

Format:

```
****************
* 1111 01 1000 *
****************
    CP-Code
```

Function:

a. A descriptor is expected to be on the EVALUATION STACK. The following conditions are expected, but NOT checked:

   1. The name-value bit must be OFF.

   2. The non-self-relative bit must be ON.

   3. The descriptor is always 48 bits.

b. The descriptor is moved to the VALUE STACK.

c. The descriptor on the EVALUATION STACK is replaced by a 48 bit, non-self-relative descriptor that points to the descriptor just moved to the VALUE STACK.

LOAD NUMERIC ZERO

```
*******
* ZOT *
*******
```

Syntax: ZOT

Format:

```
********
* 0101 *
********
OP-Code
```

Function:

This operator causes a FIXED, self-relative descriptor to be generated and placed on the EVALUATION STACK. The descriptor will contain the number zero.

STACK OPERATORS

NAME	MNEMONIC	OP CODE	ARGUMENTS
BUMP VALUE STACK POINTER	BVSP	1111 1C 1011	
DUPLICATE	DUP	1100 0C	
DELETE	DEL	1111 0C 0011	
EXCHANGE	XCH	1011 00	
FORCE VALUE STACK	FVS	1100 01	

BUMP VALUE STACK POINTER

```
********
* BVSP *
********
```

Syntax: BVSP

Format:

```
******************
* 1111 10 1011 *
******************
    OP-Code
```

Function:

a. The descriptor on the top of the NAME STACK must be a simple, non-self-relative descriptor, that points to the top of the VALUE STACK.

b. The low order 16 bits of the value described by the descriptor on the top of the EVALUATION STACK are put into the length field of the descriptor on top of the NAME STACK.

c. The top of stack pointer for the VALUE STACK is incremented by this 16 bit value.

NOTE: This operator is not generated by the SDL compiler after the IV.0 release.

DELETE

```
*******
* DEL *
*******
```

Format:

```
******************
* 1111 00 0011 *
******************
     OP-Code
```

Function:

The top descriptor on the EVALUATION STACK is deleted along with its associated value in the VALUE STACK if the name-value bit is on.

DUPLICATE

```
*******
* DUP *
*******
```

Syntax: DUP

Format:

```
***********
* 1100 00 *
***********
  OP-Code
```

Function:

a. This operator takes the 48 bit (simple) descriptor on the top of the EVALUATION STACK, duplicates it exactly and pushes it onto the top of the EVALUATION STACK.

b. If the name-value bit is on, the VALUE STACK portion will not be duplicated.

FORCE VALUE STACK

```
*******
* FVS *
*******
```

Syntax: FVS

Format:

```
***********
* 1100 01 *
***********
  OP-Code
```

Function:

a. An operand is expected to be on the EVALUATION STACK.

b. If the operand is a non-self-relative value, no action is taken. Otherwise the operand is converted to a non-self-relative value by copying its data to the top of the VALUE STACK, setting its descriptor's address field to the copied data and changing the type field to non-self-relative value.

EXCHANGE

```
*******
* XCH *
*******
```

Syntax: XCH

Format:

```
***********
* 1011 00 *
***********
  OP-Code
```

Function:

a. This operator swaps the two top descriptors or the EVALUATION STACK.

b. Both entries will always be 48 bit (simple) descriptors.

c. The name-value bit may be set in one, but not both, of the descriptors.

PROCEDURE OPERATORS

NAME	MNEMONIC	OP CODE	ARGUMENTS
CALL	CALL	0111	TYPE-SEG-PAGE-DISP
IF THEN	IFTH	1001	TYPE-SEG-PAGE-DISP
IF THEN ELSE	IFEL	1101 10	ADDR TYPE,TYPE-SEG-PAGE-DISP
CASE	CASE	1111 01 0100	# OF ADDR, ADDR TYPE, TYPE-SEG-PAGE-DISP,...., TYPE-SEG-PAGE-DISP
UNDO	UNDO	1000	# OF LEVELS
UNDO CONDITIONALLY	UNDC	1111 01 0011	# OF LEVELS
RETURN	RTRN	1111 01 0101	# OF LEVELS
RETURN FORMAL CHECK	RTNC	1111 11 1101 001	# OF LEVELS, TYPE,LENGTH
EXIT	EXIT	1101 11	# OF LEVELS
CYCLE	CYCL	1110 11	DISPLACEMENT
MARK STACK	MKS	1011 11	
MARK STACK AND UPDATE	MKU	1111 01 1111	LL
ENABLE-DISABLE INTERRUPTS	EDI	1111 11 0101	V
EXIT-ENABLE INTERRUPTS	XTEI	1111 11 0110	V,# OF LEVELS
CO-ROUTINE ENTRY	CNTR	1111 11 1010 000	
CO-ROUTINE EXIT	CXIT	1111 11 1010 001	# OF LEVELS

CALL

```
********
* CALL *
********
```

Syntax: CALL <Code Address>

Format:

```
****************
* 0111 *       *
****************
OP-Code     *
            *
            *** Type, Segment, Displacement
```

Function:

a. The code address type may not be of type "null".

b. The location (page number, segment number, displacement) of the first bit following the instruction is pushed into the PROGRAM POINTER STACK.

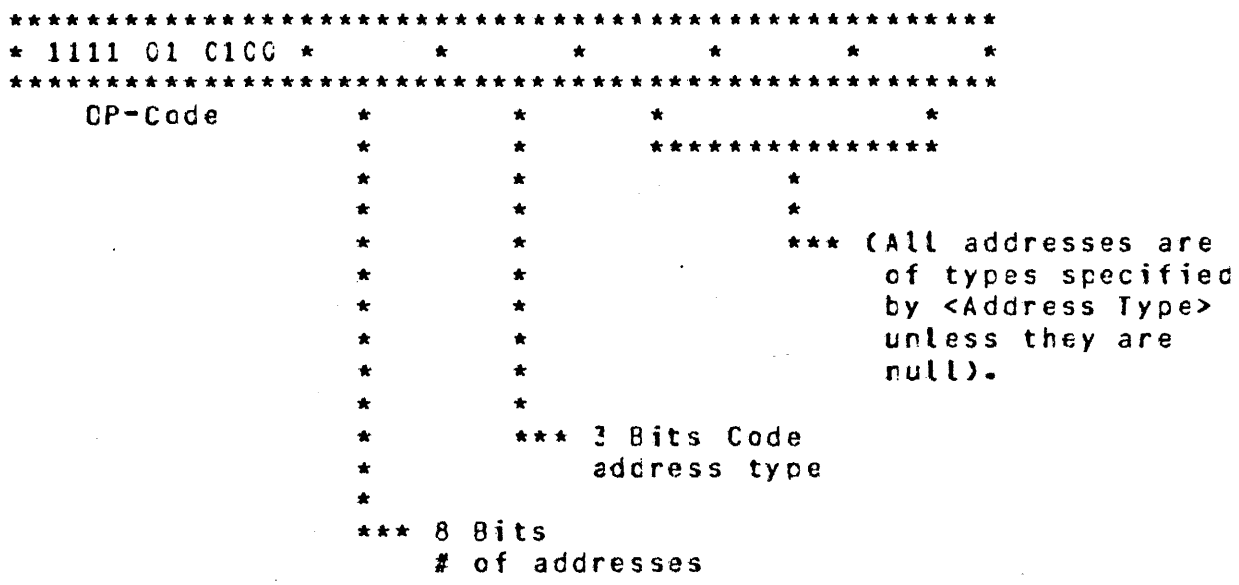
c. The code address in the instruction is used as the address of the next instruction to execute. This may require a look up in the segment dictionary to locate the segment.

CASE

```
********
* CASE *
********
```

Syntax: CASE <# of Code Addresses><Address Type>
<Code Address><Code Address> .....

Format:



Function:

a. An operand is expected to be on the EVALUATION STACK. The value of this operand is usec to select one of the code addresses. 0 selects the first address, 1 the second, etc.

b. When the value is out of range, an error interrupt occurs.

c. When the proper address has been isolated a "CALL" is performed on that address.

d. Any of the addresses may be null addresses (type field=111) in which case the rest of the address will be padded with zeros (even though the type is null the address size is the same as the non-null addresses).

e. When a null address is selected the operation is terminated.

COROUTINE ENTRY

```
********
* CNTR *
********
```

Syntax: CNTR

Format:

```
**********************
* 1111 11 1010 000 *
**********************
       OP-Code
```

The table asscciated with each of the co-routine operators has the following format.

```
DECLARE

     01 TABLE,

          02  NUMBER.OF.ENTRIES     BIT(4),

          02  ENTRY.ADDRESS         BIT(32),

          02  PPS.COPY(16)          BIT(32),
```

Function:

a. The descriptor on the top of the EVALUATION STACK will contain the address of TABLE.

b. The current code address is pushed onto the PROGRAM POINTER STACK.

c. The number of elements of PPS.CCPY, specifiec by

NUMBER.OF.ENTRIES is pushed cnto the PRCGRAM POINTER STACK.

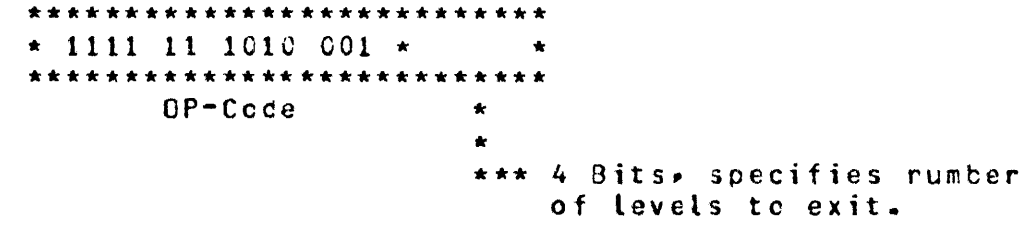
d. The address of the next instruction is taken from ENTRY.ADDRESS.

CO-ROUTINE EXIT

```
********
* CXIT *
********
```

Syntax: CXIT <# of Levels to exit>

Format:



See CNTR for TABLE format.

Function:

a. The descriptor on the top of the EVALUATION STACK will contain the address of TABLE.

b. Number of levels is stored in NUMBER.OF.ENTRIES.

c. The current code address is stored in ENTRY.ADDRESS.

d. The number of entries on the top of the PROGRAM POINTER STACK, (specified by # of levels) is copied to PPS.COPY (0) through PPS.COPY (# OF levels-1) if the # of levels is 0, nothing is copied.

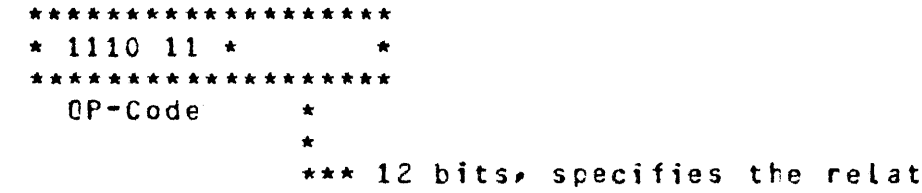
e. An UNDO is performed, using the # of levels as the number of entries to pop off the PROGRAM POINTER STACK.

CYCLE

```
********
* CYCL *
********
```

Syntax: CYCL <displacement>

Format:



Function:

a. When the twelve bits are read by the processor, the program pointer will point the the first bit of the next instruction.

b. The twelve bit displacement is then subtracted from the program pointer to give a new next instruction address.

c. This operator does not cause a change in code segment.

ENABLE DISABLE INTERRUPTS

```
*******
* EDI *
*******
```

Syntax: EDI <Enable-disable bit>

Format:

```
**********************
* 1111 11 0101 *     *
**********************
    CP-Code        *
                   *
                   *** 1 bit, 0 => Enable
                              1 => Disable
```

Function:

Disable

a. Disable extracts from the interrupt queue a high priority interrupt and places the port, channel, and reference address associated with it on top of the EVALUATION STACK.

b. The MCP's high-priority interrupt handling routine is entered (via a CALL). If there is no high priority interrupt in the queue the next in line S-op is executed.

Enable

a. Enable checks the interrupt queue for a high priority interrupt; if one exists it places the associated port, channel, and reference address in the

EVALUATION STACK.

b. A CALL to the MCP's high priority interrupt handling routine is then performed. If no high priority interrupt exists an UNDO is performed to terminate handling of high pricrity interrupts.

NOTE: For MCP use only.

EXIT

```
********
* EXIT *
********
```

Syntax: EXIT <# of Levels>

Format:

```
**********************
* 1101 11 *    #    *
**********************
  OP-Code      *
               *
               *** Number of levels to exit (4 bits)
```

Function: Refer to Control Stack Mechanism

a. DISPLAY and CONTROL STACK are updated by the following algorithm:

```
CURRENT.NSP:=CS.NSP(TCSP:=CSP:=CSP-1);
CURRENT.VSP:=CS.VSP(CSP);
DO SEARCH FOREVER:
    IF CS.EXITED.LL(TCSP) = 0 THEN UNDO SEARCH;
    IF CURRENT.LL = CS.ENTERED.LL(TCSP:=TCSP-1) THEN
    DO;
        DISPLAY(CURRENT.LL):=CS.NSP(TCSP);
        UNDO SEARCH;
    END;
END SEARCH;
```

```
CURRENT.LL:=CS.EXITED.LL(CSP);
```

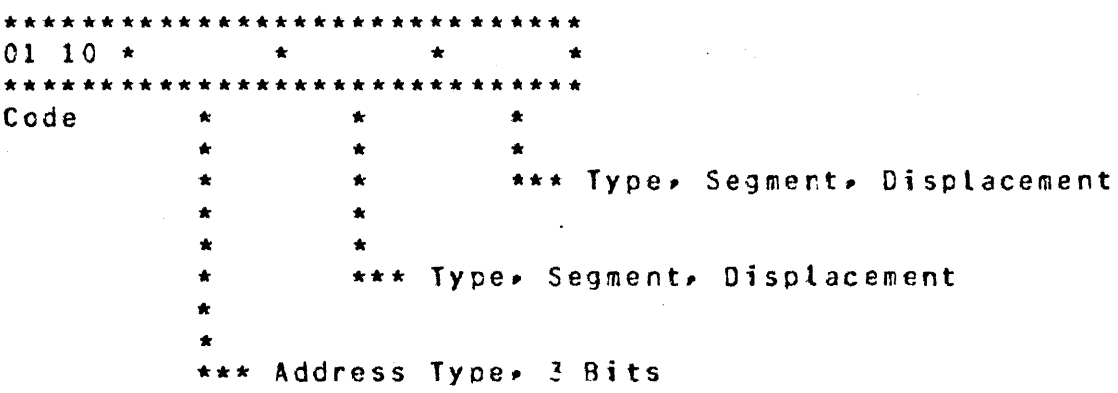
b. An UNDO is performed on the PROGRAM POINTER STACK, using the number of levels given in the instruction.

IF THEN ELSE

****
* IFEL *
****

Syntax: IFEL <Address Type><Code Address><Code Address>

Format:



The type of both code addresses must be the same as the <Address Type>. The code address may not be of type "null".

Function:

a. An operand is taken from the EVALUATION STACK. The rightmost bit of the value of the operand is examined.

1 = True
0 = False

b. If true then,

1. A CALL instruction is executed using the first code address.

c. If false then,

1. A CALL instruction is executed using the second code address.

d. The return code address that is pushed into the PROGRAM POINTER STACK by the call points to the bit following this instruction.

IF THEN

```
********
* IFTH *
********
```

Syntax: IFTH <Code Address>

Format:

```
****************
* 1001 *       *
****************
OP-Code    *
           *
           *** Type, Segment, Displacement
```

Function:

a. An operand is taken from the EVALUATION STACK. The rightmost bit of the value of the operand is examined.

1 = True
0 = False

b. If true then,

1. A CALL instruction is executed using the code address given in the instruction.

2. The return code address that is pushed into the PROGRAM POINTER STACK points to the bit following this instruction.

c. If false then,

1. This instruction is terminated and the next instruction in line is executed.

MARK STACK

```
*******
* MKS *
*******
```

Syntax: MKS

Format:

```
************
* 1011 11  *
************
  OP-Code
```

Function:

Refer to CONTROL STACK mechanism

The CONTROL STACK is updated by the following algorithm:

```
CS.NSP(CSP):=CURRENT.NSP;

CS.EXITED.LL(CSP):=CURRENT.LL;

CS.ENTERED.LL(CSP):=C;

CS.VSP(CSP):=CURRENT.VSP;

CSP:=CSP + 1;
```

MARK STACK AND UPDATE

```
*******
* MKU *
*******
```

Syntax: MKU <Lexic Level>

Format:

```
**************************
* 1111 01 1111 *        *
**************************
    OP-Code          *
                     *
                     *** 4 bits, LL
                         Lexic Level being entered.
```

Function:

Refer to CONTROL STACK mechanism

The CONTROL STACK and DISPLAY are updatec by the following algcrithm:

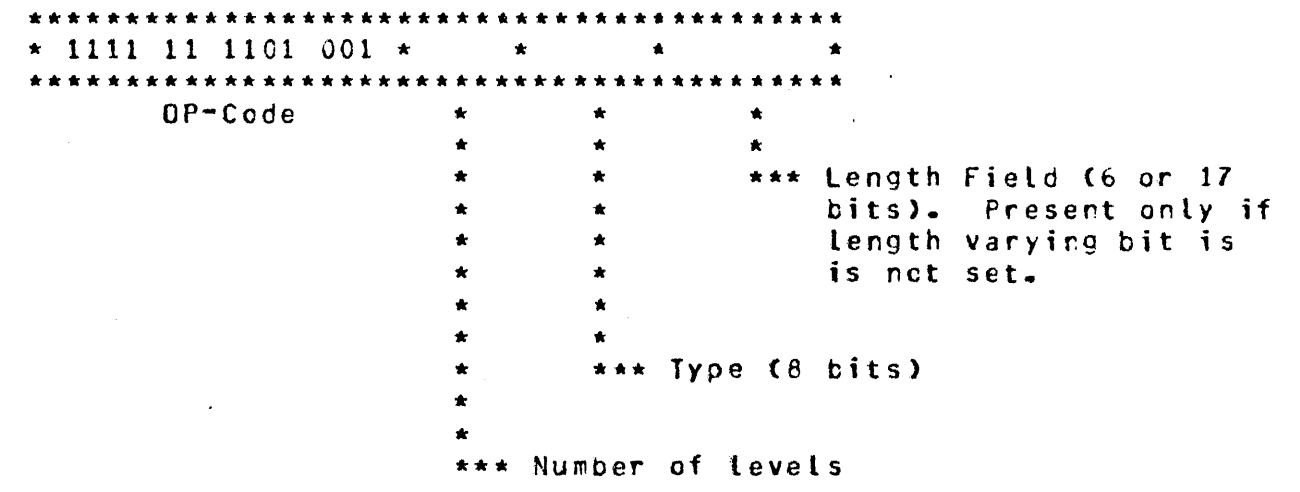
```
CSP.NSP(CSP):=CURRENT.NSP

CS.EXITED.LL(CSP):=CURRENT.LL;

CS.ENTERED.LL(CSP):=CURRENT.LL:=LL;

CS.VSP(CSP):=CURRENT.VSP;

CSP:=CSP+1;

DISPLAY(CURRENT.LL):=CURRENT.NSP;
```

RETURN FORMAL CHECK

****
* RTNC *
****

Syntax: RTNC <# of Levels><Type Field><Length Field>

Format:



Function:

Refer to Control Stack Mechanism

a. Type Field

Bits: 0 - Not used

1 - Not used

2 - Not used

3 - Not used

4-5 00 BIT
01 FIXED
1C CHARACTER
11 VARYING

6 - Always 0

7 - 1 if length is varying

b. DISPLAY and CONTROL STACK are updated using the following algorithm;

```
CURRENT.NSP:=CS.NSP(TCSP:=CSP:=CSP-1);

CURRENT.VSP:=CS.VSP(CSP);

DO SEARCH FOREVER;

    IF CS.EXITED.LL(TCSP)=0 THEN

        UNDO SEARCH;

    IF CURRENT.LL=CS.ENTERED.LL(TCSP:=TCSP-1)

        THEN

    DO;

        DISPLAY(CURRENT.LL):=CS.NSP(TCSP);

        UNDO SEARCH;

    END;

END SEARCH;

CURRENT.LL:=CS.EXITED.LL(CSP);
```

c. The value to be returned is an operand on the top of the EVALUATION STACK.

d. The type and length of the data must match the type and length specified in the instruction unless the type or length is varying.

e. If the operand on the EVALUATION STACK is not self-relative, the data to be returned is moved to the top of the VALUE STACK, i.e. changed to a value from an address.

f. An UNDO is performed on the PROGRAM POINTER STACK, using the number of levels given in the instruction.

RETURN

```
********
* RTRN *
********
```

Syntax: RTRN <# of Levels>

Format:

```
**********************
* 1111 01 C1C1 *     *
**********************
    OP-Code        *
                   *
                   *** 4 Bit field that specifies
                       the number of levels
```

Function: Refer to Control Stack Mechanism

a. RTRN performs no check on the type and length fields of the value returned and is the return that is normally emitted by the compiler. RTNC does perform checking and is available as a compiler option.

b. The value to be returned is an operand on the top of the EVALUATION STACK.

c. If the name value bit is off and the descriptor is non-self-relative, then an error condition occurs.

d. DISPLAY and CONTROL STACK are updated by the following algorithm;

```
CURRENT.NSP:=CS.NSP(TCSP:=CSP:=CSP-1);

CURRENT.VSP:=CS.VSP(CSP);
```

```
DO SEARCH FOREVER;
    IF CS.EXITED.LL(TCSP)=0 THEN
        UNDO SEARCH;
    IF CURRENT.LL=CS.ENTERED.LL(TCSP:=TCSP-1)
        THEN
    DO;
        DISPLAY(CURRENT.LL):=CS.NSP(TCSP);
        UNDO SEARCH;
    END;
END SEARCH;
CURRENT.LL:=CS.EXITED.LL(CSP);
```

e. If the operand on the EVALUATION STACK is not self-relative, the data to be returned is moved to the top of the VALUE STACK, i.e. changed to a value from an address.

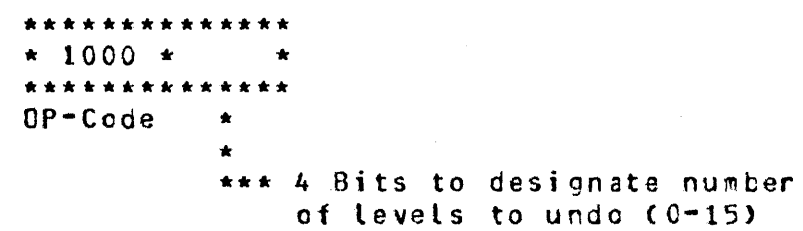
f. An UNDO is performed on the PROGRAM POINTER STACK, using the number of levels given in the instruction.

UNDO

```
********
* UNDO *
********
```

Syntax: UNDO <# of Levels>

Format:



Function:

a. The PROGRAM POINTER STACK is cut back by the number of levels specified in the instruction.

b. The next entry in the PROGRAM POINTER STACK is then used as the pointer to the next instruction. This entry is then also cut back from the PROGRAM POINTER STACK.

UNDO CONDITIONAL

```
********
* UNDC *
********
```

Syntax: UNDC <# of Levels>

Format:

```
************************
* 1111 01 0011 *      *
************************
    OP-Code         *
                    *
                    *** 4 Bits, specifies number of
                        levels to undo (0-15)
```

Function:

a. An operand is taken from the top of the EVALUATION STACK.

b. The rightmost bit of the operand is interrogated

   1 = TRUE

   0 = FALSE

c. When true an UNDO operator is performed for the number of levels indicated.

d. When false the instruction is terminated and the next in-line instruction is executed.

EXIT, ENABLE INTERRUPTS

```
********
* XTEI *
********
```

Syntax: XTEI <UNDO or EXIT bit><# of Levels to exit>

Format:

```
*****************************
* 1111 11 0110 *     *     *
*****************************
    OP-Code        *     *
                   *     *
                   *     *** 4 Bits, specifies number of
                   *           of levels to exit
                   *
                   *
                   *** 1 Bit, variant to specify if UNDO or EXIT
                       0 => UNDO
                       1 => EXIT
```

Function:

The execution of XTEI is identical to that of EXIT, except that interrupts will be enabled before the operator terminates.

NOTE: For MCP use only.

SEARCH & SCAN OPERATORS

NAME	MNEMONIC	CP CODE	ARGUMENTS
REDUCE	RDUC	1111 11 1001 101	VARIANTS, TYPE-LL-OC
SEARCH SDL STACKS	SSS	1111 11 1110 001	
SEARCH LINKED LIST	SLL	1111 01 1010	COMPARE TYPE
SEARCH SERIAL LIST	SSL	1111 11 1000 CCC	COMPARE TYPE
SORT SEARCH	SSCH	1111 11 1011 100	
THREAD VECTOR	TVEC	1111 11 1011 001	
INITIALIZE VECTOR	IVEC	1111 11 1011 000	
SORT STEP DOWN	SSD	1111 11 1011 010	
SORT SWAP	SSWP	1111 11 1011 101	
SORT UNBLOCK	UBLK	1111 11 1011 011	
DELIMITED TOKEN	DTKN	1111 11 1001 001	TYPE-LL-OC, DEL1, DEL2
NEXT TOKEN	NTKN	1111 11 1001 000	TYPE-LL-OC, SEPARATOR, V
DEBLANK	DBLK	1111 11 1001 010	TYPE-LL-OC
CHARACTER FILL	CHFL	1111 11 1001 100	
TRANSLATE	XLAT	1111 11 1110 101	
FIND DUPLICATE CHARACTERS	FDUP	1111 11 1001 011	

CHARACTER FILL

```
********
* CHFL *
********
```

Syntax: CHFL

Format:

```
**********************
* 1111 11 1001 100 *
**********************
        OP-Code
```

Function:

a. Two operands are taken from the EVALUATION STACK.

b. The top operand is the source, and must be eight bits in length.

c. The second operand is the destination.

d. The source descriptor may be self-relative or non-self-relative.

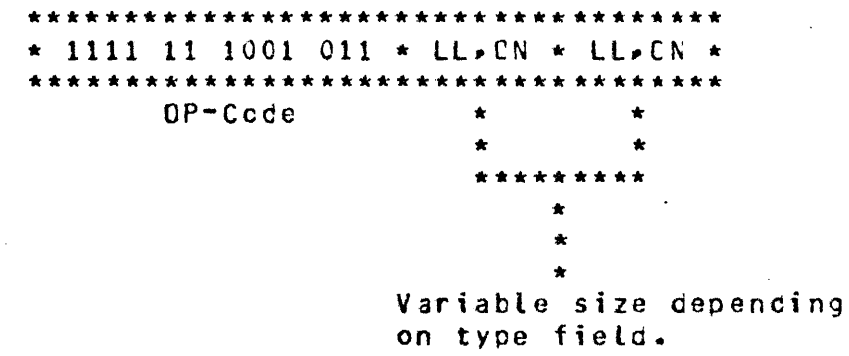
e. The eight bits the source field is moved left-justified into the destination field, repeatedly, until the destination field is filled.

FIND DUPLICATE CHARACTERS

```
********
* FDUP *
********
```

Syntax: FDUP <Data Address><Data Address>

Format:



Function:

a. The text to be scanned is initially described by the first data address.

b. The second data address describes the non-duplicate text.

c. The text will be scanned until three or more duplicate characters are found.

d. Upon return the text descriptor is modified to describe the remaining text.

e. The non-duplicate text descriptor is modified to

describe the non-duplicated text that was scanned.

f. The number of duplicate characters will be left as a 24 bit item on the top of the EVALUATION STACK.

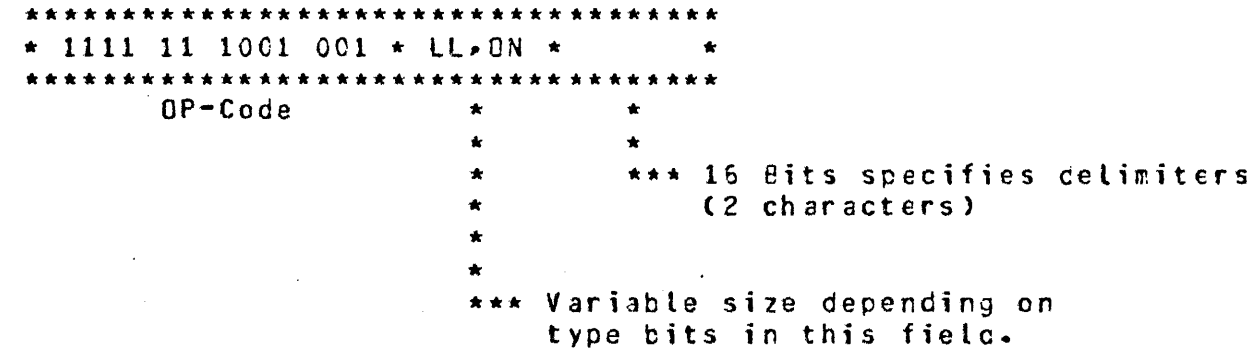
g. The duplicated character will be left as the second item on the EVALUATION STACK and will be of type CHARACTER, length 1.

DELIMITED TOKEN

```
********
* DTKN *
********
```

Syntax: DTKN <Data Address><Delimiters>

Format:



Function:

a. The descriptor located by data address is used to access the first source character.

b. The address of the first character is token-start.

c. Characters are compared sequentially to each of the eight bit delimiters until a match is found.

d. Current address will point to the character which matches one of the delimiters.

e. A descriptor is left on top of the EVALUATION STACK that is:

1. Non-self-relative.

2. The address is the address of token-start.

3. The length is equal to the current address minus token-start.

f. The address field of the NAME STACK descriptor for first character is set to the current address.

DEBLANK

```
********
* DBLK *
********
```

Syntax: DBLK <Data Address>

Format:

```
*****************************
* 1111 11 1001 010 * LL,CN *
*****************************
       OP-Code          *
                        *
                        *** Variable size depending on
                            type bits in this field.
```

Function:

a. The descriptor located by <Data Address> is used to access the first source character.

b. Characters are then passed serially until a non-blank character is found.

c. The address of the first non-blank character found is put into the address field of the descriptor described by <Data Address>.

INITIALIZE VECTOR

```
********
* IVEC *
********
```

Syntax: IVEC

Format:

```
**********************
* 1111 11 1011 CCC *
**********************
       OP-Code
```

See SSD for keys table format.

VECTOR TABLE

```
   24 Bits       24 Bits       24 Bits       24 Bits     4 Bits
*************************************************************
*  VECTOR    *  VECTOR     *     KEY      *    VECTOR    *       *
*   BASE     *  LEVEL.1    *    TABLE     *     LIMIT    * FLAGS *
*  ADDRESS   *    SIZE     *  ADDRESS     *      SIZE    *       *
*************************************************************
```

SORT VECTOR

```
      2          10              20
*********************************************
*           *           *  VECTOR ELEMENT.1  *
*           *           *     RECORD         *
*  FLAGS    *   LINK    *    ADDRESS         *
*           *           *  VECTOR ELEMENT.n  *
*********************************************
```

Function:

This operator is similar to Thread Vector, with the exception that:

1. The vector has never been initialized (had a winner).

2. The initial value of the bit displacement into the vector is zero (0).

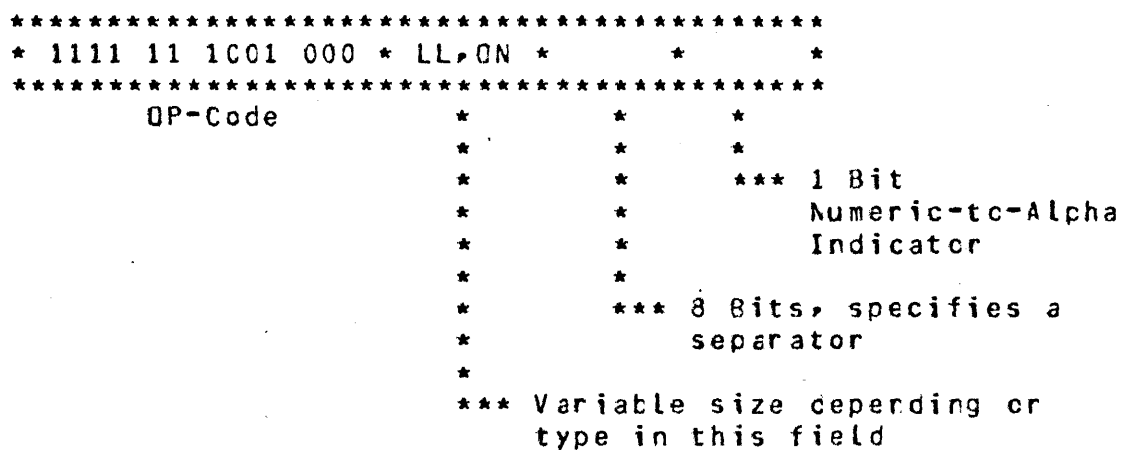
3. Each new value is incremented by 64. (Vector element length * 2).

NEXT TOKEN

****
* NTKN *
****

Syntax: NKTN <Data Address><Separator>
<Numeric-to-Alpha Indicator>

Format:



Function:

a. The descriptor described by data address is used to access the first source character.

b. The address of the first character is token-start.

c. If this character is a special character (less than "A") then set the current address to token-start +8 and go to (g).

d. If the numeric-to-alpha indicator is set (1), then

set stopper to "A".

e. If the numeric-to-alpha indicator is not set (0) and the first character is numeric then set stopper to "0". Otherwise, set stopper to "A".

f. Sequentially compare characters to stopper until one is found which is less than stopper and not equal to "separator". The current address will point to this character.

g. A descriptor is left on top of the EVALUATION STACK that is:

1. Non-self-relative of type character.

2. The address is the address of token-start.

3. The length is equal to the current address minus token-start.

h. The data address for first character is set to the current address.

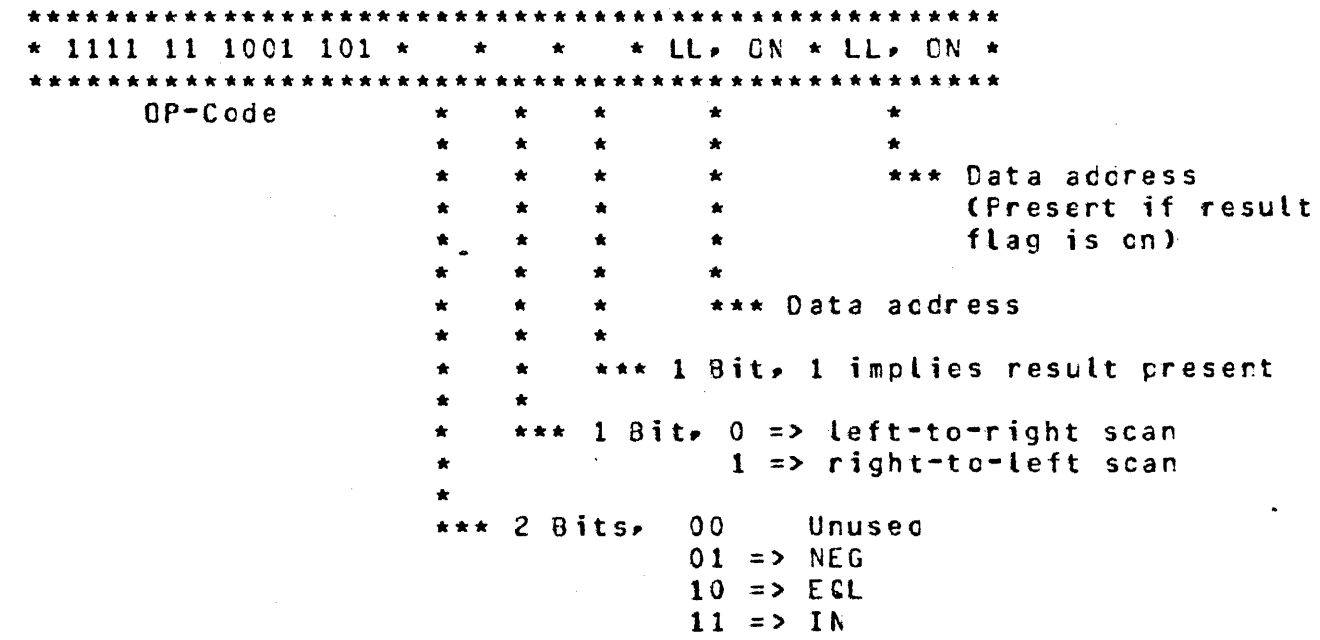
i. It is assumed that a special character such as "%" will follow the image to be scanned, in order that scanning will terminate.

REDUCE

```
********
* RDUC *
********
```

Syntax: RDUC <Type><Result.flag><Object>[<Result>]

Format:



Function:

a. The character string located by the object data address is scanned in the direction indicated for a character which satisfies a prescribed condition.

   1. If the type is EQL, an operand is taken from the EVALUATION STACK which must be 8 bits in length. The condition is satisifed by a character which is the same as this operand.

2. If the type is NEQ, an operand is taken as in the EQL case, but the condition is satisfied by a character which is different from the operanc.

3. If the type is IN, the operand on the EVALUATION STACK must be of length 256 bits. A character satisfies the condition if, when usec as an index into the operanc bit string, it selects a bit which is on.

Note: If the operand does not meet the length requirements, an error will be generated.

b. The scan terminates when either a character is found satisfying the cordition, or the object string is exhausted. At termination, a one-bit result is left on the EVALUATION STACK, @(1)0@ if nc satisfying character was found or a @(1)1@ if the character was found. In addition, the descriptor of object (and that of result, if present) is upcatec:

1. If the direction is left-to-right:

$$A_o' = A_o + x$$

$$L_o' = L_o - x$$

$$A_r' = A_o$$

$$L_r' = x$$

2. If the direction was right-to-left:

$$A_o' = A_o$$

$$L_o' = x + 8$$

$$A_r' = A_o + x + 8$$

$$L_r' = L_o - (x + \varepsilon)$$

where:

A_o is the original address of object

A_o' is the new address of object

L_o is the original length of object

L_o' is the new length of object

A_r' is the new address of result

L_r' is the new length of result

x is the difference between the address of the character satisfying the concition and the original address of object.

Note: Lengths here are calculated in bits even though they are character strings. The type field of the result descriptor will always be set to non-self-relative CHARACTER.

SEARCH LINKED LIST

```
*******
* SLL *
*******
```

Syntax: SLL<Compare Type>

Format:

```
************************
* 1111 01 1010 *      *
************************
    OP-Code         *
                    *
                    *
                    *** 3 Bits, specifies compare type
```

The compare type specifies the desired relation. It is encoded as follows:

1 - Greater than
2 - Less than
3 - Not equal to
4 - Equal to
5 - Greater than or equal to
6 - Less than or equal to

Function:

a. Four descriptors are expected to be on the top of the EVALUATION STACK. These descriptors represent the following items:

Descriptor	Meaning
First(top)	Link Location
Second	Compare Variable
Third	Compare Field Location
Fourth	Structure Address

1. Link Location is a template which describes the field in the structure which contains the base relative address of the next structure to be examined.

2. Compare Variable determines the value to be compared to the structure. Its length must be less than or equal to 24.

3. Compare Field Location is a template whose length field is the length of the field to which compare variable is to be compared and whose address field is the offset (in the structure) of the field to be compared against. The length must be less than or equal to 24.

4. Structure Address is the base relative address of the first structure to be examined.

b. The linked list is searched until either the comparison succeeds or the end of the list is found. The last element of the list must have a link field with binary 1's (i.e. @FFFFFF@ if the link field is 24 bits wide).

c. If the search succeeds, then the base relative address of the current structure is left on the EVALUATION STACK as a 24 bit-value.

d. If the search fails, then @FFFFFF@ is the value left on the EVALUATION STACK.

NOTE: This operator will not terminate if it is unable to make a successful comparison and cannot find the end of the list (a link field of all 1's).

SORT SEARCH

```
********
* SSCH *
********
```

Syntax: SSCH

Format:

```
**********************
* 1111 11 1C11 1CC *
**********************
        OP-Code
```

See SSD for keys table format.

Function:

a. This operator expects two items to be on the EVALUATION STACK.

   1. The top item is the buffer limit address.

   2. The second item is the address of the search control table.

SEARCH CONTROL TABLE

Bits	24	24	24	24	24
	CONTROL RECORD ADDRESS	KEY TABLE ADDRESS	CURRENT RECORD INDEX	SCAN & COMPARE TYPE	RECORD SIZE

b. SSCH compares each record in a buffer with a control

record on a specified key.

c. The record may be compared from the top of the buffer down or from the bottom of the buffer up.

d. The compare can be for

1. LSS (less than)
2. LEQ (less than or equal to)
3. GEQ (greater than or equal to)

3. When a record is found that satisfies the comparison a one (1) is returned. Otherwise a zero is returned.

SORT STEP DOWN

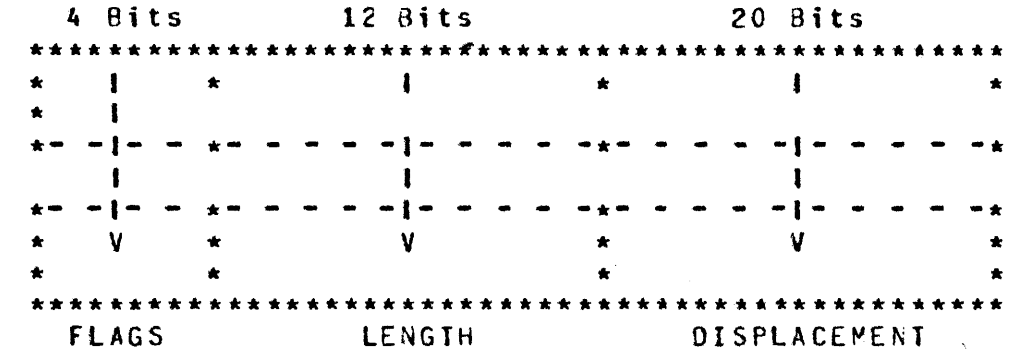
```
*******
* SSD *
*******
```

Syntax: SSD

Format:

```
**********************
* 1111 11 1011 010 *
**********************
        OP-Code
```

KEYS TABLE



Function:

a. This operator expects three items to be on the EVALUATION STACK.

   1. The value of the top item is the key table address.

   2. The second and third items are the left and right record addresses respectively.

b. The left and right records are compared according to the keys and return a value of one (1) when the right record is greater, according to the keys, than the left record.

SEARCH SERIAL LIST

```
*******
* SSL *
*******
```

Syntax: SSL <Compare Type>

Format:

```
 ***************************
 * 1111 11 1CCC COC *      *
 ***************************
        OP-Code          *
                         *
                         *** 3 Bits, specifies compare type
```

The compare type specifies the desired relation. It is encoded as follows:

1 = Greater than

2 = Less than

3 = Not equal to

4 = Equal

5 = Greater than or equal to

6 = Less than or equal to

Function:

a. Four descriptors are expected to be on the top of the EVALUATION STACK. These descriptors represent the following items.

Descriptor	Meaning
First(top)	Table Length
Secord	Compare Value
Third	First Item
Fourth	Compare Field

1. Table Length is the length, in bits, of the table to be searched.

2. Compare Value determines the value to be compared to the structure. Its length can be greater than 24.

3. First Item gives the length and address of the first item in a serial list of items that are of identical structure.

4. Compare Field gives the length and the offset within the structure of the field to which Compare Value is to be compared. Unlike SLL, the length may be greater than 24 bits.

b. The serial list of items is searched beginning with first item until Compare Value satisfies Compare Type with Compare Field, or until the end of the list is reached.

c. If the search succeeds, then the base relative address of the item containing the "successful" compare field is left on the top of the EVALUATION STACK and a 1-bit value of 1 (one) is left as the second item on the EVALUATION STACK.

d. If the search fails, then the end address of the table + 1 is left on the top of the EVALUATION STACK, and a 1-bit value of 0 (zero) is left as the second item on the EVALUATION STACK.

SEARCH SDL STACKS

```
*******
* SSS *
*******
```

Syntax: SSS

Format:

```
**********************
* 1111 11 1110 001 *
**********************
      OP-Code
```

Function:

a. Four operands are removed from the EVALUATION STACK.

   First(top) - Stack Base
   Second - Stack Top
   Third - Compare Base
   Fourth - Compare Top

b. The stack to be searched will consist of SDL descriptors and will be searched from base to top (or vice versa) for a simple descriptor whose address lies between compare base and compare top.

c. Array descriptors and self-relative descriptors may be ignored.

d. If the search is successful then a one bit value of (1) is left on the top of the EVALUATION STACK.

e. If the search is not successful then a one bit value of (0) is left on the top of the EVALUATION STACK.

SORT SWAP

```
********
* SSWP *
********
```

Syntax: SSWP

Format:

```
**********************
* 1111 11 1011 101 *
**********************
      OP-Code
```

Function:

a. This operator pops two operands off the EVALUATION STACK.

b. The values of the two operands are interchanged without regard to type.

c. When the two fields are of unequal length, the swap is limited to the length of the shorter field. The longer field is filled from the left; low order bits are undisturbed.

THREAD VECTOR

```
********
* TVEC *
********
```

Syntax: TVEC

Format:

```
**********************
* 1111 11 1011 001 *
**********************
       OP-Code
```

See SSD for keys table format.
See IVEC for vector table format.

Function:

a. This operator expects two items to be on the EVALUATION STACK.

   1. The top item is a bit displacement into the sort vector, where the two vector elements to compare reside.

   2. The second item is the address of the 100 bit vector table.

b. The operator then compares the two vector elements (according to the sort keys) and stores a winner.

c. When the store address is equal to the vector limit the EVALUATION STACK is cut back and the instruction pointer advanced.

d. When the store address is not equal to the vector limit the bit displacement is updated and the EVALUATION STACK and program pointer remain unchanged.

SORT UNBLOCK

```
********
* UBLK *
********
```

Syntax: UBLK

Format:

```
**********************
* 1111 11 1011 011 *
**********************
       OP-Code
```

a. This operator handles blocking for the sort operators. It moves data from a source field to a destination field and updates the blocking characteristics.

b. Four items are expected on the EVALUATION STACK.

   1. The first item is the destination address.

   2. The second item is the source address.

   3. The third item is the length of the data transfer.

   4. The fourth item is the address of the "Pseudo-Sort-Fib".

c. When the block count goes to zero (0) a value of one (1) is returned.

d. When the block count is other than zero a value of zero (0) is returned.

TRANSLATE

```
********
* XLAT *
********
```

Syntax: XLAT

Format:

```
**********************
* 1111 11 1110 101 *
**********************
      OP-Code
```

Function:

a. Five operands are taken from the EVALUATION STACK.

   1. Descriptor for the result field. (An address operand).

   2. A self-relative descriptor whose value is the size of the items in the result field and in the translate table.

   3. The translate table. (May be a value or an address operand).

   4. A self-relative descriptor whose value is the size of the items in the source field.

   5. A descriptor for the source field. (An address operand).

b. Each of the items in the source field is used to subscript into the table to obtain an item which is then placed into the result field in the position that corresponds to the position of the original item obtained from the source.

c. This process continues until

   1. The source field is exhausted.

   2. The result field is full.

   3. An error occurs (e.g. translate error).

d. If either the source or the result is not a multiple of its respective item size then the last item translated or the last translated value will be truncated as required.

e. Both source and table item sizes must be equal to or less than 24 bits in length, otherwise a run-time error occurs.

f. The table need only be large enough to accommodate those items which will actually appear in the source. That is, the upper end of the table need not be present if it will never be accessed. However, attempting to access a table item which is beyond the actual size of the table will cause a run-time error.

MISCELLANEOUS OPERATORS

NAME	MNEMONIC	OP CODE	ARGUMENTS
TRANSFER MESSAGE	XFRM	1111 11 1010 010	DEST.VARIABLES SOURCE VARIABLE
HASH CODE	HASH	1111 11 1000 001	
SWAP	SWAP	1111 01 0110	
FETCH	FECH	1111 00 1100	
FETCH AND SAVE	FECS	1111 11 1110 011	
DISPATCH	DISP	1111 01 1011	
HALT	HALT	1111 11 0010	
READ CASSETTE	RDCS	1111 01 0010	
LENGTH	LENG	1111 10 0000	
LOAD SPECIAL	LSP	1111 01 1110	VARIANT
CLEAR ARRAY	CLR	1111 10 0111	
COMMUNICATE	COMM	1111 10 0110	
REINSTATE	REIN	1111 10 0001	
FETCH CMP	FCMP	1111 10 0010	
DATA ADDRESS	ADDR	1111 01 1001	
SAVE STATE	SVST	1111 11 0001	
HARDWARE MONITOR	HMON	1111 11 0011	
OVERLAY	OVLY	1111 11 0000	
PROFILE	PRFL	1111 10 1111	ENTRY NUMBER

PARITY ADDRESS	PADR	1111 11 0111
EXECUTE	EXEC	1111 11 1110 010
COMMUNICATE WITH GISMO	CWG	1111 11 1110 110
ADD TIMER	ADDT	1111 11 1100 000
SUBTRACT TIMER	SUBT	1111 11 1100 001

ADDRESS

```
********
* ADDR *
********
```

Syntax: ADDR

Format:

```
****************
* 1111 01 1001 *
****************
    OP-Code
```

Function:

a. A descriptor is expected to be on the top of the EVALUATION STACK. This descriptor must be

   1. Non-array

   2. Non-self-relative

   3. The name-value bit must be off.

b. The type field of the descriptor is set to self-relative, BIT.

c. The length field of the descriptor is set to 24.

ADD TIMER, SUBTRACT TIMER

```
****************
* ADDT * SUBT *
****************
```

Syntax: ADDT <cell number>

SUBT <cell number>

Format:

```
        ADDT
 ******************************
 * 1111 11 1100 000 *         *
 ****************************** *
       OP-Code          *
                        *
                        *** 16 Bits specifies cell number

        SUBT
 ******************************
 * 1111 11 1100 001 *         *
 ******************************
       OP-Code          *
                        *
                        *** 16 Bits specifies cell number
```

Function:

a. This operator assumes that DISPLAY(18) points to cell zero of an array of 48-bit cells.

b. The cell number is used to subscript into the array to locate the indicated cell.

c. The high-resolution timer is added (subtracted) from the indicated cell.

d. An adjustment will be made to the timer to exclude the time required by the operator from being included in the cell time.

e. If no high-resolution timer is present on the system, then this operator will not change the timing cell.

CLEAR ARRAY

```
*******
* CLR *
*******
```

Syntax: CLR

Format:

```
*****************
* 1111 10 0111 *
*****************
    OP-Code
```

Function:

a. The descriptor on tcp of the EVALUATION STACK must be a non-paged array descriptor.

b. When the array is of type CHARACTER, the array elements are blank filled.

c. For any other data types the array elements will be zero filled.

COMMUNICATE

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

Syntax: COMM

Format:

```
****************
* 1111 10 011C *
****************
    OP-Code
```

Function:

a. The descriptor on the top of the EVALUATION STACK is moved to RS.COMMUNICATE.MSG.PTR.

b. When the name-value bit is on, it is turned off in the RS.COMMUNICATE.MSG.PTR.

c. The descriptor is removed from the EVALUATION STACK and the VALUE STACK is cut back, if necessary.

d. The M-machine state is stored in the appropriate parts of the RS.NUCLEUS.

e. The program whose RS.NULCEUS address is given ir the RS.COMMUNICATE.LR is then instatec.

COMMUNICATE WITH GISMO

```
*******
* CWG *
*******
```

Syntax: CWG

Format:

```
**********************
* 1111 11 1110 11C *
**********************
        OP-Code
```

Function:

a. A descriptor is expected to be on the top of the EVALUATION STACK.

b. If the descriptor is self-relative it is made non-self-relative by copying its data to the VALUE STACK.

c. The address field of the descriptor is made absolute and placed in the "T" register.

d. The length field is placed in the "L" register.

e. A swapper value of 14 is placed in the "X" register and GISMO is called.

f. The descriptor on the EVALUATION STACK is not removed.

DISPATCH

```
********
* DISP *
********
```

Syntax: DISP

Format:

```
*****************
* 1111 01 1011 *
*****************
     OP-Code
```

Function:

a. Two operands are removed from the EVALUATION STACK.

b. The top operand is an address operand of the I/O descriptor to be dispatched.

c. The low order 7 bits of the second operand should be encoded as follows:

```
******************
*       *        *
******************
    *        *
    *        *
    *        *** 4 Bits, specifies Channel
    *
    *
    *** 3 Bits, specifies Port
```

d. These two operands are passed as parameters to GISMO, which then performs the I/O operation.

e. GISMO returns a value which describes the results of the dispatch. A 24-bit self-relative descriptor is left on the EVALUATION STACK. The value has the following meaning.

0 = Dispatch register lockout bit set

1 = Successful dispatch

2 = Successful dispatch, but missing device

EXECUTE

```
********
* EXEC *
********
```

Syntax: EXEC

Format:

```
**********************
* 1111 11 1110 010 *
**********************
       OP-Code
```

Function:

The value of the top operand on the EVALUATION STACK will be considered to be the next op-code to be executed. This is for the testing of experimental op-codes in the interpreter.

This operator is not in release irterpreters.

FETCH COMMUNICATE MESSAGE POINTER

```
********
* FCMP *
********
```

Syntax: FCMP

Format:

```
****************
* 1111 10 0010 *
****************
     OP-Code
```

Function:

a. If the RS.MCP.BIT is set then the RS.COMMUNICATE.MSG.PTR is accessed.

b. If the RS.MCP.BIT is not set then the RS.REINSTATE.MSG.PTR is accessed.

c. The accessed field is assumed to be a descriptor and is placed on the top of the EVALUATION STACK.

FETCH, FETCH AND SAVE

```
****************
* FECH * FECS *
****************
```

Syntax: FECH
FECS

Format:

```
      FECH                      FECS
*****************      ***********************
* 1111 00 1100 *      * 1111 11 1110 011 *
*****************      ***********************
     OP-Code                   OP-Code
```

Function:

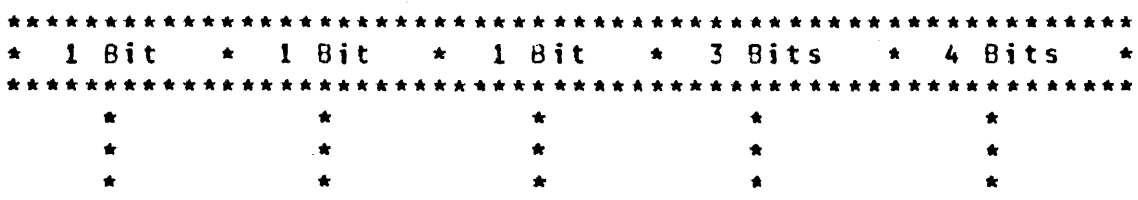
a. An operand is taken from the EVALUATION STACK. Its value indicates which item is to be examined on the Interrupt Queue (Refer to the MCP manual).

    0 => Use the top item
   -1 => Use the top high priority interrupt time.
         Otherwise the value is the reference
         address +24 of the result descriptor
         desired.

b. Two descriptors are left on the EVALUATION STACK.

   1. The top item is a Bit (24) self-relative data item whose value is the address of the desired I/O result descriptor.

   2. The second item is a Bit (10) self-relative data item whose value is the port and channel of the I/O operation. This has the following format:



c. When the operator is Fetch, then the item is removed from the Interrupt Queue.

d. When the operator is Fetch and Save, ther the information is left in the Interrupt Queue.

e. If the Interrupt Queue was empty, then both descriptors have a value of zero (0).

HALT

```
********
* HALT *
********
```

Syntax: HALT

Format:

```
****************
* 1111 11 0010 *
****************
     OP-Code
```

Function:

a. The M-machine state is stored in the appropriate parts of the RS.NUCLEUS. (Refer to the MCP manual).

b. The low-order 24 bits of the value of the operand on the top of the EVALUATION STACK, is moved to the T-register.

c. This operand is popped off the EVALUATION STACK and the M-instruction "HALT" is executed.

HASH CODE

```
********
* HASH *
********
```

Syntax: HASH

Format:

```
**********************
* 1111 11 1CCC CC1 *
**********************
     OP-Code
```

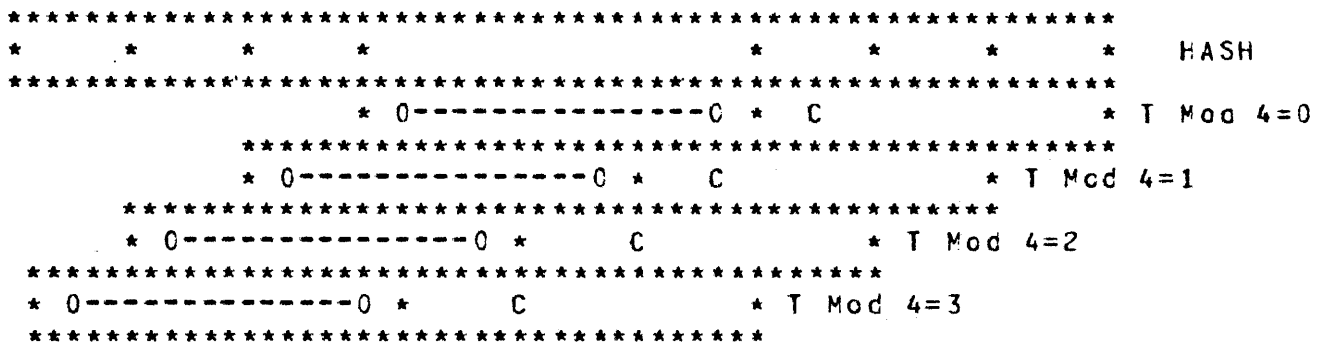
Function:

a. The algorithm for generating the hash code can best be described by an SDL procedure.

```
PROCEDURE HASH.CODE(TOKEN) BIT (24);

    FORMAL TOKEN CHARACTER VARYING;

    DECLARE

        (L,T)   BIT(24),

        C       CHARACTER(1),

        01      HASH BIT(27),

                02 FILLER BIT(3),

                02 TOTAL BIT(24);

IF (HASH:=L:=LENGTH(TOKEN)) > 15 THEN L:=15;

    T:=0;

/* C DESCRIBES THE CHARACTER PRECEDING TOKEN */
```

```
        DESCRIPTOR(C):=@48C008@ CAT

            DATA.ADDRESS(TOKEN) -8;

            DO HASH.IT FOREVER;

                IF L <BUMP T THEN RETURN TOTAL;

                BUMP TOTAL [-(T MOD 4)] BY NEXT.ITEM(C);

            END HASH.IT;

        END HASH.CODE;
```

b. Each character of TOKEN is being right acjusted ir a field of zeros and added to TOTAL in one of four positions.



HARDWARE MONITOR

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

Syntax: HMON

Format:

```
****************
* 1111 11 0011 *
****************
    OP-Code
```

Function:

a. An operand is taken from the top of the EVALUATION STACK.

b. The low order eight bits of the operand's value will besed as the operand of a monitor micro-instruction.

LENGTH

```
********
* LENG *
********
```

Syntax: LENG

Format:

```
****************
* 1111 10 0000 *
****************
     OP-Code
```

Function:

a. An operand is taken from the EVALUATION STACK.

b. A self-relative, fixed result is returned to the top of the EVALUATION STACK.

   1. When the operand was of type CHARACTER, the value of the result is equal to the length field in the operand's descriptor divided by eight.

   2. When the operand was of any type other than CHARACTER, the value is equal to the length field in the operand's descriptor.

LOAD SPECIAL

```
*******
* LSP *
*******
```

Syntax: LSP Variant

Format:

```
************************
* 1111 01 1110 *      *
************************
    OP-Code          *
                     *
                     *** 5 Bit Variant indicating value to load
```

Function:

a. The variant field indicates a value to be loaded to the top of the EVALUATION STACK, usually as a 24 bit, self-relative data item.

Variant	Value
0	Base register (absolute address)
1	Limit register (base relative)
2	S-Memory size in bits
3	M-Memory size in bits
4	CONTROL STACK top (base relative)
5	EVALUATION STACK top (base relative)
6	CONTROL STACK size in bits
7	NAME STACK top (base relative)
8	DISPLAY BASE (base relative)
9	CONSOLE SWITCHES
10	SPO.INPUT.PRESENT (1 bit)
11	PROGRAM.SWITCHES (40 bits)

OVERLAY

```
********
* OVLY *
********
```

Syntax: OVLY

Format:

```
****************
* 1111 11 0000 *
****************
     OP-Code
```

Function:

a. An operand is taken from the top of the EVALUATION STACK.

b. The value of the operand will be used by GISMO as an index into the interpreter dictionary.

c. The interpreter dictionary will specify the action to be taken. (Refer to the B1800/B1700 MCP manual).

READ CASSETTE

```
********
* RDCS *
********
```

Syntax: RDCS

Format:

```
****************
* 1111 01 0010 *
****************
     OP-Code
```

Function:

a. Three operands are expected to be on top of the EVALUATION STACK.

b. Top item - address operand where a one or zero is to be stored according to whether a hash total read from the tape is good or bad.

   Second item - a self-relative value of one if a hash total is to be read from the tape, else zero.

   Third item - an address operand where data from the tape is to be stored.

c. The following conventions apply;

   1. At least one record will be read.

   2. A sufficient number of records will be read to fill the buffer requested (third operand).

   3. The cassette will not be stopped in the middle of a record.

4. Cassette record lengths should be multiples of 16 bits.

5. If the size of the buffer is not a multiple of 16 and a HASH.TOTAL checking was requested, then a bad.hash (0) indication will be returned.

6. If NO.HASH.TOTAL checking was requested, then a good hash (1) will be returned.

PARITY ADDRESS

```
********
* PADR *
********
```

Syntax: PADR

Format:

```
****************
* 1111 11 0111 *
****************
    OP-Code
```

Function:

a. Starting at absolute address zero (0), S-Memory is scanned until MAXS is reached.

b. If a parity error is detected, the error is corrected and its location is placed on top of the EVALUATION STACK as a self-relative 24 bit data item. The address returned points to the beginning of the byte of S-memory in which the parity error occurred.

c. If no error is detected, a value of @FFFFFF@ is placed on the EVALUATION STACK.

NOTE: For MCP use only.

PROFILE

```
********
* PRFL *
********
```

Syntax: PRFL <Profile Array Index>

Format:

```
************************
* 1111 10 1111 *      *
************************
    OP-Code         *
                    *
                    *** 12 Bits, specifies index
```

Function:

a. DISPLAY(16) contains the base relative address of the profile array. This array is between the limit of DISPLAY and the base of the CONTROL STACK. Each element of the array is 16 bits.

b. The entry number given by <profile array index>, is bumped by one.

REINSTATE

```
********
* REIN *
********
```

Syntax: REIN

Format:

```
*****************
* 1111 10 0001 *
*****************
     OP-Code
```

Function:

a. The descriptor on the top of the EVALUATION STACK is assumed to describe the RS.COMMUNICATE.MSG.PTR or RS.NUCLEUS of the program to be reinstated. (Refer to the MCP manual for a description of a run structure.)

b. This descriptor should have the name-value bit off.

c. The reinstating program's S-machine state is stored in the appropriate parts of its RS.NUCLEUS.

d. The address of the reinstating program's RS.NUCLEUS is stored in the reinstated program's RS.COMMUNICATE.LR.

e. The address field of the descriptor on the EVALUATION STACK contains the address of the RS.NUCLEUS of the program that is instated when the descriptor is removed from the EVALUATION STACK.

SAVE STATE

```
********
* SVST *
********
```

Syntax: SVST

Format:

```
*****************
* 1111 10 00C1 *
*****************
     OP-Code
```

Function:

The current state of the S-machine is saved in RS.M.MACHINE (Refer to the B1600/E1700 MCP manual).

SWAP

```
********
* SWAP *
********
```

Syntax: SWAP

Format:

```
****************
* 1111 01 0110 *
****************
     OP-Code
```

Function:

a. This operator takes two operands from the EVALUATION STACK.

   1. The top (source) operand may be a value or an address operand.

   2. The second (destination) operand must be an address operand.

      a) The length of the destination determines the width of the data "swappec".

b. Destination field:

   1. When the length of the destination is greater than 24 bits, only the rightmost 24 bits are isolated, otherwise the entire field is isolated.

c. Source field:

   1. A value equal in length to the destination field is isolated in the source field.

2. The bits are taken from the right.

   a) When the source length is shorter thar the destinatior field, leading zeros are supplied.

d. The source field is moved to the destiratior field and the former value of the destinatior field is returred as a result on the EVALUATICN STACK.

e. The swap must be performed such that no other processor, sharing the same memory can simultaneously swap out the same area of memory.

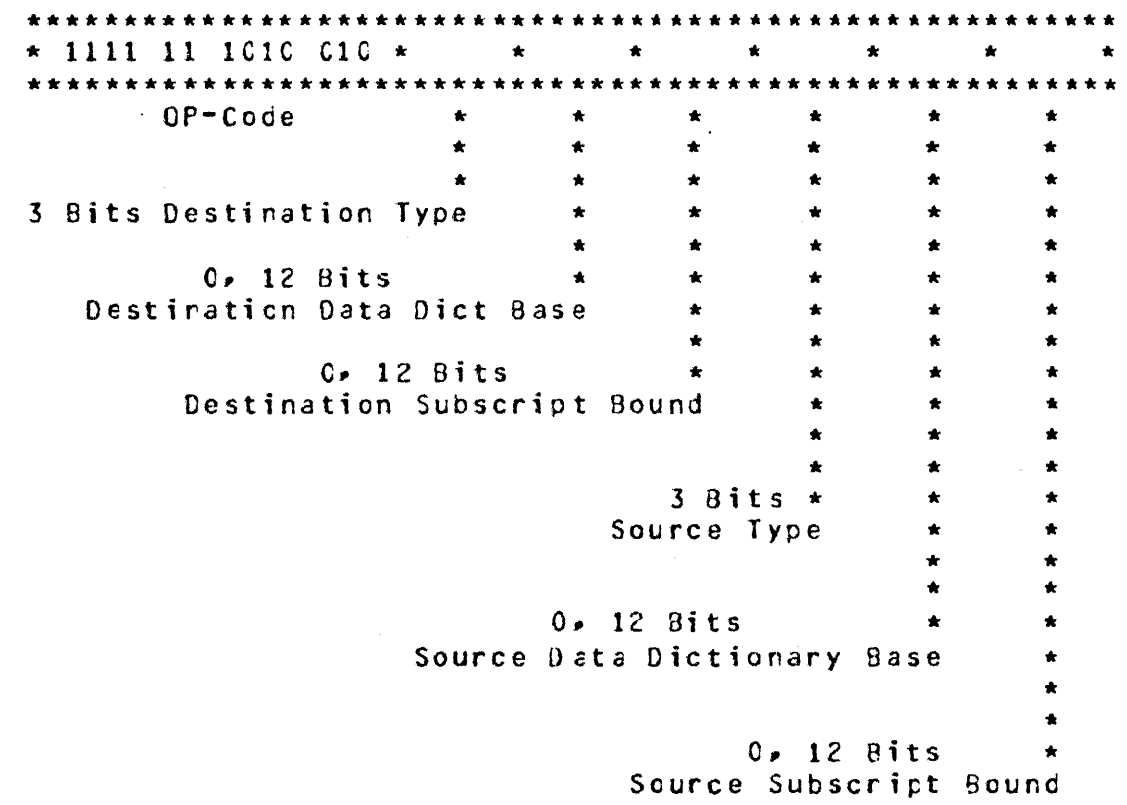
TRANSFER MESSAGE

```
********
* XFRM *
********
```

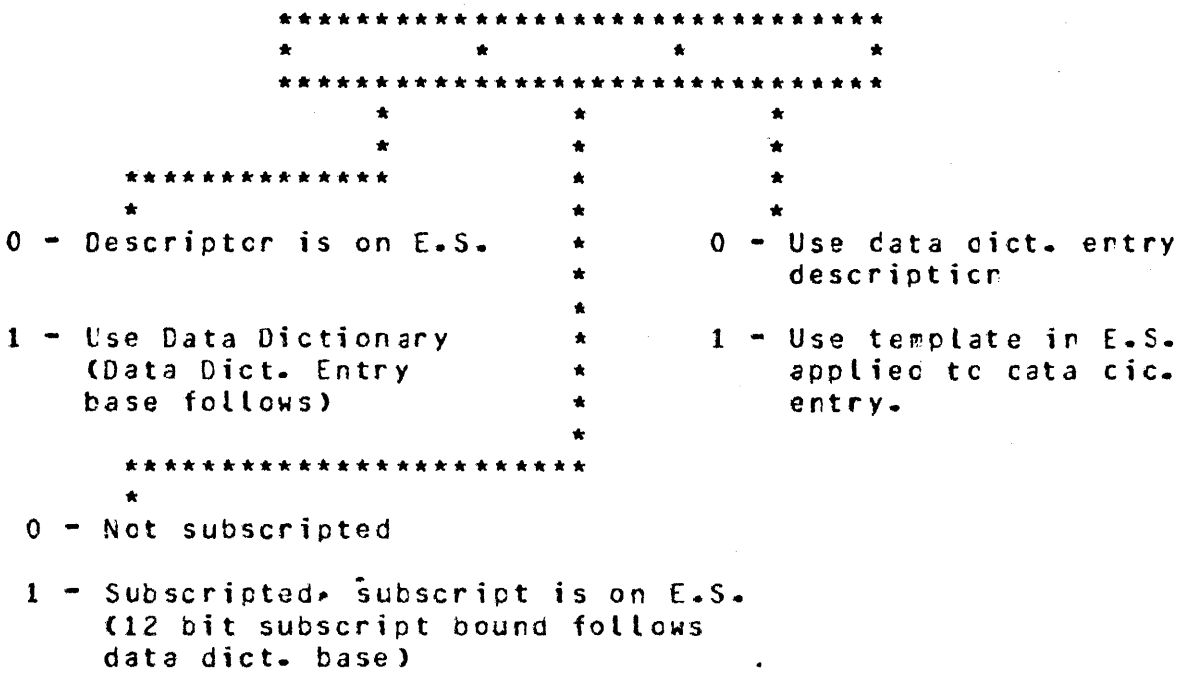
Syntax: XFRM <Destination type> [<DES.DATA DICT.BASE>
[<DEST.SUBSCRIPT.BOUND>]] <Source Type>
[<SOURCE DATA DICT.BASE> [<Source Subscript Bound>]]

NOTE: This operator is scheduled to be removed from the S-machine.

Format:



TYPE FIELD



Function:

a. This operator transfers information between:

1. Messages (or parts of messages)

2. Data contained within Base-Limit and messages or parts of messages.

b. The source and destination fields may be of the following types:

TYPE	MEANS OF DESCRIPTION
1. Base-Limit data	Descriptor on EVALUATION STACK
2. Message	Data dictionary entry number
3. Message array element	Data dictionary entry number and subscript on the EVALUATION STACK, is added to the data dictionary entry number
4. Part of message or message array element	The template on the EVALUATION STACK is to be applied to the message which is described by the data dictionary entry. This is obtained the same as 2 or 3.

c. The type field of the destination field is used to obtain the description of the destination field.

d. The type field of the source field is used to obtain the description of the source field.

e. Data is transferred from the source field to the destination field as in a character to character store operation:

ALPHABETIC INDEX