Description of Oregon SNOBOL5 Source code for 64 bit Intel CPU

Written by Viktors Berstis

CONTENTS

1. Introduction

The Oregon SNOBOL5 programming language is an upgrade of Minnesota SNOBOL4. Both are based on SNOBOL4 developed at Bell Laboratories. SNOBOL5 is not to be confused with SL5 (Snobol Language 5 presumably) which eventually evolved into the ICON programming language. There are other implementations, particularly Catspaw's SNOBOL4+, SPITBOL and Budne's CS4. References to these can be found at the end of this document. Oregon SNOBOL5 is named as it is because Viktors Berstis, its author now resides in Oregon rather than Minnesota and the "5" is to distinguish it from the various other flavors.

The SNOBOL programming language is implemented in macro-assembly language called SIL (SNOBOL4 Implementation Language). This macro language is largely machine-independent and is designed so that it can be implemented on a variety of computers. By implementing the macro language, and using the SNOBOL4 system already written in the macro language, one obtains a version of SNOBOL that is largely source-language compatible with other versions implemented in the same way. Nearly all the logic of the SNOBOL5 language resides in the program written in the macro language. Thus if the macro language is implemented properly, the resulting implementation of SNOBOL is essentially the same as other SNOBOL5 implementations.

Oregon SNOBOL5 was implemented using the SIL source, but with some modified and additional macros, some of which are more extensive than the original set. The macros are implemented as Intel assembler instructions and programs. Both Windows and Linux platforms are supported as compatably as possible. Due to IBM PC1 memory limitations, Minnesota SNOBOL4 was doubly interpreted to save memory space. SNOBOL5 runs much faster as a result and various limitations are vastly increased. Strings can have extremely long lengths because of 64 bit addressing, integers are 64 bits rather than 32 bits, double precision floating point is standard, and there are other extensions to the language.

Because SNOBOL5 evolved from the old Minnesota SNOBOL4, all of the tools to create SNOBOL5 used are the same or have been modified as needed. A SNOBOL program is used to expand the SIL macros. The Microsoft ML64 assembler and linker is used to assemble the code. Linux "objcopy" is used to create the Linux executable from the Microsoft output. For the LINUX version, only system calls are used (to date ####). A minimum of Windows system libraries are used. Oregon SNOBOL5 consists of three major parts, the SIL executable code, the SIL data declarations and support routines.

This manual describes the SIL macro language and contains information about its implementation for the Intel 64 bit processors. Information given here is related to the old Version 3.11 of the SIL source, which has been modified in various ways to meet the needs of this implementation. Some references to the old IBM 360 implementation are included for comparison purposes. The bulk of this description is directly taken from "Implementing SNOBOL4 in SIL; Version 3.11" by Ralph E. Griswold, technical report S4D58 from the University of Arizona. Although much of the description of SIL in here is independent of the target machine, some is specifically related to the Oregon SNOBOL5 implementation. Section 2 describes environmental considerations. Section 3 describes the representation of data and the storage layout. Syntax tables and character graphics are described in Section 4. Section 5 explains the method used to describe the macro operations. Section 6 is a list of all macro operations with a description of how to implement each one. Section 7 contains miscellaneous implementation notes. Section 8 describes each of the Oregon SNOBOL5 source files, assembly procedures. acknowledgements and additional documentation.

2. Environmental Considerations

2.1 Requirements

In order to process the source for SNOBOL5, the following software and equipment is required:

A WINDOWS 7 or later computer
Microsoft Assembler ml64.exe and link.exe (from Microsoft Visual Studio)
Oregon SNOBOL5
A text editor
A Linux machine (I used Ubuntu)

Another version of SNOBOL (including Minnesota SNOBOL4 on Windows XP or older system) can be used to create the initial version of SNOBOL5 if you have not downloaded an executable version of SNOBOL5. This solves the "Catch 22" problem.

2.2 Input and Output

SNOBOL4 was originally designed to perform all input and output through FORTRAN IV routines. Minnesota SNOBOL4 implemented the FORTRAN formatting facility. However, few people used the FORTRAN formatting facility so this has been eliminated from SNOBOL5. Formats specified in the SNOBOL5 programs are ignored.

2.3 Storage Requirements

The default work space size allocated for SNOBOL5 is about 8 megabytes. A run time parameter (--work) can change this value. Allocated storage is referred to in machine-independent data units called descriptors and specifiers that occupy 16 and 32 bytes respectively.

2.4 Other Considerations

SNOBOL5 makes few other demands on its operating system environment. Oregon SNOBOL5 has a keyword (&PARM) which retrieves the command line used to invoke SNOBOL5. Time and date information is used by SNOBOL5, but it is not essential. Various names in the original SIL code have been renamed to avoid reserved words in the ML64.EXE assembler. Examples are PTR->PTRF, FNC->FNF ...

3. Representation of Data

There are a few basic types of data used in the SNOBOL5 system, and a number of aggregates of the basic types. The basic types of data are:

descriptors
specifiers
character strings
syntax table entries

3.1 Descriptors

Descriptors are used to represent all pointers, integers, and real numbers. A descriptor may be thought of as the basic "word" of SNOBOL5. Descriptors consist of three fixed-length fields:

address
flag
value

The size and position of these fields is determined from the data they must represent and the way that they are used in the various operations. The following paragraphs describe some specific requirements.

3.1.1 Address Field

The address field of a descriptor must be large enough to address any descriptor, specifier, or program instruction within the SNOBOL5 system. (Descriptors do not have to address individual characters of strings. See Section 3.2.) The address field must also be large enough to contain any address, integer or real number (including sign) that is to be used in a SNOBOL5 program. The address field is the most frequently used field of a descriptor and is used frequently for addressing and integer arithmetic and it should be positioned so that these operations can be performed efficiently. On some systems address arithmetic may need to be computed differently than integer arithmetic and thus there are alternate macros for these situations.

In SNOBOL5 this field is an 8 byte (64bit) field at the start of descriptor. All eight bytes are used for integer and floating point values. No descriptor can be allocated at address zero, since this is used as a null pointer. Also, user programs will not be able to directly use all of the work space since SNOBOL5 uses some of it for its internal operation. Addresses of SIL instructions are simply their addresses in memory.

3.1.2 Flag Field

The flag field is used to represent the states of a number of disjoint conditions and is treated as a set of bits that are individually tested, turned on, and turned off. Six flag bits used in SNOBOL5: (TTL, STTL, FNF, MARK, PTRF, and VISITED). The last byte (offset 15) of a descriptor is used for the flag field in Oregon SNOBOL5.

3.1.3 Value Field

The value field is used to represent a number of internal quantities that are represented as unsigned integers (magnitudes). These quantities include the encoded representation of source-language data types, the length of strings, and the size (in address units) of various data aggregates. The value field need not be as large as the address field, but it must be large enough to represent the size of the largest data aggregate that can be formed. In Oregon SNOBOL5, 7 bytes following the address field (offset 8) form the 56 bit value field.

3.2 Specifiers

Specifiers are used to refer to character strings. Almost all operations performed on character strings are handled through operations on specifiers. All specifiers are the same size and have five fields:

address
flag
value
offset
length

Specifiers and descriptors may be stored in the same area indiscriminately, and are indistinguishable to many processes in the SNOBOL5 system. As a result, specifiers are composed of two descriptors. One descriptor is used in the standard way to provide the address, flag, and value fields. The other descriptor is used in a nonstandard way. Its address field is used to represent the offset of an individual character from the address given in the specifier's address field. The value field of this other descriptor is used for the length. The flag field (offset 31) in the second descriptor of a specifier must remain zero because the garbage collection routine would otherwise fail to work properly. Thus the length field is the same size as the value field.

3.3 Character Strings

Character strings are represented in packed format, as many characters per descriptor as possible. Storage of character strings in SNOBOL5 dynamic storage is always in storage units that are multiples of descriptors.

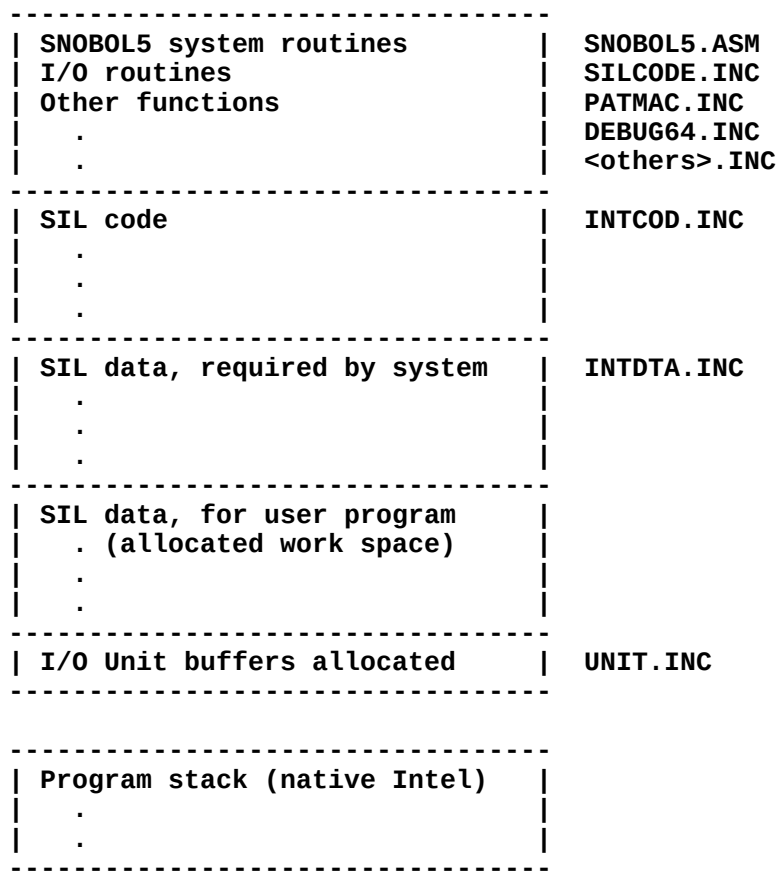
3.4 Syntax Table Entries

Syntax tables are necessarily somewhat machine dependent. Consequently, implementation of these tables is done individually for each machine. They encode a state machines to recognize various syntactic items. A description of the table requirements is given in a following section.

The original SIL code implemented its program stack within its work space. This stack grew with increasing addresses as descriptor and specifiers were pushed onto the stack. However, SNOBOL5 uses the native Intel processor stack instead. The addresses decrease as items are pushed onto this stack. This stack is separate from the allocated SNOBOL5 work space (inlike in the original SIL implementations). Since this stack is in a separate part of memory, the SIL code was modified to handle the differences in stack implementation and the garbage collector was altered to account for this. The garbage collector in SNOBOL5 was implemented with assembly code rather than via SIL macros, primarily to handle the native stack, eliminate some limits and to perform faster than the SIL implementation.

3.5 Oregon SNOBOL5 Storage Layout

The following diagram approximately illustrates the storage layout of Oregon SNOBOL5.



The first part is native assembly code which received the run time parameters, allocates the work space, initializes various data areas and executes the SIL code. Included are various include files which have various support routines, including those for I/O and other functions. Of particular interst might be the PATMAC and DEBUG64 includes. They can be used somewhat independently in other assembler programs. PATMAC implements SNOBOL style pattern matching functions which are used to parse the command line parameters, for example. DEBUG64 is particularly useful while implementing SNOBOL5 as a debug tool. This eliminated the need to use a commercial or other debug package.

The next section (INTCOD.INC) is the expansion of the SIL macro code into assembly code. This is performed by the CRTMAIN.SNO program which generates assembly code suitable as input to Microsofts ML64.EXE assembler. OPTIMIZE.SNO makes a pass of the assembly code to change the machine instructions such that the XPTR descriptor, used frequently in the SIL code, is implemented in registers R14 and R15 instead of memory. This can be disabled via a setting in CRTMAIN.SNO.

The next sections consists of the SIL data workspace required by the system. Part of this data is in INTDTA.INC generated by the SIL macros and the rest is alocated work space.

When an I/O unit is specifed, an additional allocation is made to handle it. The file name, state, attributes and buffers are stored there. See the UNIT.INC file for more information.

Finally, the native program stack is about 8 megabytes and is used by the SIL code as well as any system calls for I/O etc. The size of this is specified by a parameter to the Microsoft LINK.EXE command.

Differences in the Windows versus Linux implementation are handled by the "linuxenvironment" definition specified on the assembler run. For Linux, the resulting SNOBOL5.EXE file is converted to a Linux executable using the "objcopy" utility.

4. Syntax Tables and Character Graphics

4.1 Characters

The SNOBOL language permits the use of any character that can be represented on a particular machine. There are certain characters that have syntactic significance in the source language. The codes, graphics, and internal representations vary from machine to machine. For each machine, representations are chosen for each of the syntactically significant characters. Such characters and sets of characters are given descriptive names to avoid dependence on a particular machine. In the list that follows, ASCII graphics are used as a point of reference.

function	name	graphics
ALPHANUMERIC	digit and letter	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz 0123456789
AT	operator	@
BLANK	separator and operator	blank and tab
BREAK	dot and underscore	.
CMT	comment line	* or # or /
CNT	continue line	+.
COLON	goto designator and dimension separator	:
COMMA	argument separator	,
CTL	control line	-
DOLLAR	operator	$
DOT	operator	.
DQUOTE	literal delimiter	"
EOS	statement terminator	;
EQUAL	assignment	=
FGOSYM	failure goto designator	F or f
FLTENOT	float e notation	E or e
KEYSYM	operator	&
LEFTBR	reference and goto delimiter	<[
LEFTPAREN	expression delimiter	(
LETTER	letter	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
MINUS	operator	-
NOTSYM	operator	~
NUMBER	digit	0123456789
ORSYM	operator	|
PERCENT	operator	%
PLUS	operator	+
POUND	operator	#
QUESYM	operator	?
RAISE	operator	^! or **
RIGHTBR	reference and goto delimiter	>]
RIGHTPAREN	expression delimiter	)
SGOSYM	success goto designator	S or s
SLASH	operator	/
SQUOTE	literal delimiter	'
STAR	operator	*
TERMINATOR	expression terminator	;)>,] blank and tab

4.2 Syntax Tables

The lexical syntax of the SNOBOL language is analyzed using the operation STREAM which is driven from syntax tables. The syntax tables provide a representation of a finite state machine used during lexical analysis.

In a syntax table there is an entry for each character at a position corresponding to the numerical value of the internal encoding of that character. The syntax table entry specifies the action to be taken if that character is encountered. The actions are:

1. CONTIN, indicating that the current syntax table is to be used for processing the next character.

2. GOTO(TABLE), indicating that TABLE is to be used for processing the next character.

3. STOP, indicating that STREAM should terminate with the last character examined to be included in the accepted string.

4. STOPSH, indicating the STREAM should terminate with the last character examined not to be included in the string accepted.

5. ERROR, indicating that STREAM should terminate with an error indication.

6. PUT(ADDRESS), indicating that ADDRESS is to be placed in the address field of the descriptor STYPE.

The classes of characters for which actions are to be taken are given in FOR designations. CONTIN and GOTO(TABLE) provide information about the next table to use and are typically represented by addresses in syntax table entries. STOP, STOPSH, and ERROR are type indicators used to stop the streaming process.

SNABTB is used in pattern matching for ANY(CS), BREAK(CS), NOTANY(CS), and SPAN(CS). SNABTB is modified during execution by the macros CLERTB and PLUGTB. The other syntax tables are not modified.

The syntax tables for Oregon SNOBOL5 are generated in the form of intel code using macros which correspond to the syntax table entries. See the source file STREAM.INC which implements the STREAM macro.

```
BEGIN BIOPTB
FOR(PLUS) PUT(ADDFN) GOTO(TBLKTB)
FOR(MINUS) PUT(SUBFN) GOTO(TBLKTB)
FOR(DOT) PUT(NAMFN) GOTO(TBLKTB)
FOR(DOLLAR) PUT(DOLFN) GOTO(TBLKTB)
FOR(STAR) PUT(MPYFN) GOTO(STARTB)
FOR(SLASH) PUT(DIVFN) GOTO(TBLKTB)
FOR(AT) PUT(BIATFN) GOTO(TBLKTB)
FOR(POUND) PUT(BIPDFN) GOTO(TBLKTB)
FOR(PERCENT) PUT(BIPRFN) GOTO(TBLKTB)
FOR(RAISE) PUT(EXPFN) GOTO(TBLKTB)
FOR(ORSYM) PUT(ORFN) GOTO(TBLKTB)
FOR(KEYSYM) PUT(BIAMFN) GOTO(TBLKTB)
FOR(NOTSYM) PUT(BINGFN) GOTO(TBLKTB)
FOR(QUESYM) PUT(BIQSFN) GOTO(TBLKTB)
ELSE ERROR
END BIOPTB


BEGIN CARDTB
FOR(CMT) PUT(CMTTYP) STOPSH
FOR(CTL) PUT(CTLTYP) STOPSH
FOR(CNT) PUT(CNTTYP) STOPSH
ELSE PUT(NEWTYP) STOPSH
END CARDTB


BEGIN DQLITB
FOR(DQUOTE) STOP
ELSE CONTIN
END DQLITB


BEGIN ELEMTB
FOR(NUMBER) PUT(ILITYP) GOTO(INTGTB)
FOR(LETTER) PUT(VARTYP) GOTO(VARTB)
FOR(SQUOTE) PUT(QLITYP) GOTO(SQLITB)
FOR(DQUOTE) PUT(QLITYP) GOTO(DQLITB)
FOR(LEFTPAREN) PUT(NSTTYP) STOP
ELSE ERROR
END ELEMTB


BEGIN EOSTB
FOR(EOS) STOP
ELSE CONTIN
END EOSTB


BEGIN FLITB
FOR(NUMBER) CONTIN
FOR(FLTENOT) GOTO(FLITC)
FOR(TERMINATOR) STOPSH
ELSE ERROR
END FLITB
```

```
BEGIN FLITC
FOR(PLUS) GOTO FLITD
FOR(MINUS) GOTO FLITD
FOR(NUMBER) GOTO FLITD
ELSE ERROR
END FLITC


BEGIN FLITD
FOR(NUMBER) CONTIN
FOR(TERMINATOR) STOPSH
ELSE ERROR
END FLITD


BEGIN FRWDTB
FOR(BLANK) CONTIN
FOR(EQUAL) PUT(EQTYP) STOP
FOR(RIGHTPAREN) PUT(RPTYP) STOP
FOR(RIGHTBR) PUT(RBTYP) STOP
FOR(COMMA) PUT(CMATYP) STOP
FOR(COLON) PUT(CLNTYP) STOP
FOR(EOS) PUT(EOSTYP) STOP
ELSE PUT(NBTYP) STOPSH
END FRWDTB


BEGIN GOTFTB
FOR(LEFTPAREN) PUT(FGOTYP) STOP
FOR(LEFTBR) PUT(FTOTYP) STOP
ELSE ERROR
END GOTFTB


BEGIN GOTOTB
FOR(SGOSYM) GOTO(GOTSTB)
FOR(FGOSYM) GOTO(GOTFTB)
FOR(LEFTPAREN) PUT(UGOTYP) STOP
FOR(LEFTBR) PUT(UTOTYP) STOP
ELSE ERROR
END GOTOTB


BEGIN GOTSTB
FOR(LEFTPAREN) PUT(SGOTYP) STOP
FOR(LEFTBR) PUT(STOTYP) STOP
ELSE ERROR
END GOTSTB


BEGIN IBLKTB
FOR(BLANK) GOTO(FRWDTB)
FOR(EOS) PUT(EOSTYP) STOP
ELSE ERROR
END IBLKTB


BEGIN INTGTB
FOR(NUMBER) CONTIN
FOR(TERMINATOR) PUT(ILITYP) STOPSH
FOR(DOT) PUT(FLITYP) GOTO(FLITB)
FOR(FLTENOT) PUT(FLITYP) GOTO(FLITC)
ELSE ERROR
END INTGTB
```

```
BEGIN LBLTB
FOR(ALPHANUMERIC) GOTO(LBLXTB)
FOR(BLANK,EOS) STOPSH
ELSE ERROR
END LBLTB


BEGIN LBLXTB
FOR(BLANK,EOS) STOPSH
ELSE CONTIN
END LBLXTB


BEGIN NBLKTB
FOR(TERMINATOR) ERROR
ELSE STOPSH
END NBLKTB


BEGIN NUMBTB
FOR(NUMBER) GOTO(NUMCTB)
FOR(PLUS,MINUS) GOTO(NUMCTB)
FOR(COMMA) PUT(CMATYP) STOPSH
FOR(COLON) PUT(DIMTYP) STOPSH
ELSE ERROR
END NUMBTB


BEGIN NUMCTB
FOR(NUMBER) CONTIN
FOR(COMMA) PUT(CMATYP) STOPSH
FOR(COLON) PUT(DIMTYP) STOPSH
ELSE ERROR
END NUMCTB


BEGIN SNABTB
FOR(FGOSYM) STOP
FOR(SGOSYM) STOPSH
ELSE ERROR
END SNABTB


BEGIN SQLITB
FOR(SQUOTE) STOP
ELSE CONTIN
END SQLITB


BEGIN STARTB
FOR(BLANK) STOP
FOR(STAR) PUT(EXPFN) GOTO(TBLKTB)
ELSE ERROR
END STARTB


BEGIN TBLKTB
FOR(BLANK) STOP
ELSE ERROR
END TBLKTB
```

```
BEGIN UNOPTB
FOR(PLUS) PUT(PLSFN) GOTO(NBLKTB)
FOR(MINUS) PUT(MNSFN) GOTO(NBLKTB)
FOR(DOT) PUT(DOTFN) GOTO(NBLKTB)
FOR(DOLLAR) PUT(INDFN) GOTO(NBLKTB)
FOR(STAR) PUT(STRFN) GOTO(NBLKTB)
FOR(SLASH) PUT(SLHFN) GOTO(NBLKTB)
FOR(PERCENT) PUT(PRFN) GOTO(NBLKTB)
FOR(AT) PUT(ATFN) GOTO(NBLKTB)
FOR(POUND) PUT(PDFN) GOTO(NBLKTB)
FOR(KEYSYM) PUT(KEYFN) GOTO(NBLKTB)
FOR(NOTSYM) PUT(NEGFN) GOTO(NBLKTB)
FOR(ORSYM) PUT(BARFN) GOTO(NBLKTB)
FOR(QUESYM) PUT(QUESFN) GOTO(NBLKTB)
FOR(RAISE) PUT(AROWFN) GOTO(NBLKTB)
ELSE ERROR
END UNOPTB


BEGIN VARATB
FOR(LETTER) GOTO(VARBTB)
FOR(COMMA) PUT(CMATYP) STOPSH
FOR(RIGHTPAREN) PUT(RPTYP) STOPSH
ELSE ERROR
END VARATB


BEGIN VARBTB
FOR(ALPHANUMERIC,BREAK) CONTIN
FOR(LEFTPAREN) PUT(LPTYP) STOPSH
FOR(COMMA) PUT(CMATYP) STOPSH
FOR(RIGHTPAREN) PUT(RPTYP) STOPSH
ELSE ERROR
END VARBTB


BEGIN VARTB
FOR(ALPHANUMERIC,BREAK) CONTIN
FOR(TERMINATOR) PUT(VARTYP) STOPSH
FOR(LEFTPAREN) PUT(FNCTYP) STOP
FOR(LEFTBR) PUT(ARYTYP) STOP
ELSE ERROR
END VARTB
```

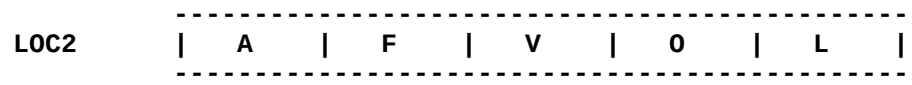
5. Describing the Macros

This section explains the method of describing the macros. The instructions for implementing an operation usually consist of a description of the operation's function, figures indicating data relating to the operation, and programming notes that contain details and references to other relevant information. The figures consist of stylized representations of the various data objects and the fields within them.

5.1 Diagrammatic Representation of Data

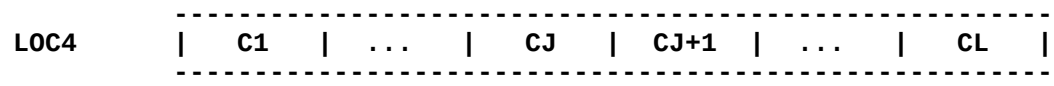
The representation of a descriptor at LOC1 is shown below. A, F, and V indicate the values of the address, flag, and value fields.

The representation of a specifier at LOC2 is shown below. A, F, V, O, and L indicate the values of the address, flag, value, offset, and length fields.



Character strings have two representations depending on how many characters are relevant to the description. The short representation of a string of L characters is shown below. C1 and CL are the first and last characters, respectively. In this representation, the intermediate characters are indicated by dots.

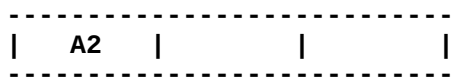
The long representation of a string of L characters at LOC4 is shown below. CJ and CJ+1 are relevant characters in the interior of the string. The long representation is used when such interior characters must be specified.



The representation of a syntax table entry is shown below. A, T, and P indicate values of the next table address, type indicator, and put field as specified by the PUT action.

Various values and expressions may occur in the fields of data objects. Fields are left blank when their value is not used in an operation. In data objects that are changed by an operation, unchanged fields are left blank. For example, if the figure below referred to a descriptor to be changed, the new value of the address

field would be A2, and no other fields would be changed.



Letters are used as abbreviations to differentiate the values that may appear in a field. The seven basic fields are indicated by the letters A, F, V, O, L, T, and P. Numerical suffixes (which may be thought of as subscripts) are used as necessary to distinguish between values of the same type. Thus, for example, A1, A32, and AN might be used to refer to addresses, F1 and F2 to flags, and so on. To make further distinctions where appropriate, I and R are used to indicate integers and real numbers, respectively.

5.2 Branch Points

Program labels are included in the argument lists of many macros. These addresses are points to which control may be transferred, depending on data supplied to the macros. In general, some or all of the branch points may be omitted in a macro call. An omitted branch point signifies that control is to pass to the next macro in line if the condition corresponding to the omitted branch point is satisfied. For example ACOMP is called in the following forms:

```
ACOMP DESCR1,DESCR2,GTLOC,EQLOC,LTLOC
ACOMP DESCR1,DESCR2,GTLOC,EQLOC
ACOMP DESCR1,DESCR2,GTLOC
ACOMP DESCR1,DESCR2,GTLOC,,LTLOC
ACOMP DESCR1,DESCR2,,EQLOC,LTLOC
ACOMP DESCR1,DESCR2,,EQLOC
ACOMP DESCR1,DESCR2,,,LTLOC
```

where GTLOC, EQLOC, and LTLOC are addresses to which ACOMP may branch. ACOMP is not called with all three branch points omitted, since that is not a meaningful operation. Other macros such as SUM (q.v.) are often called with all branch points omitted.

Implementation of the macros must take omission of branch points into consideration. Alternate expansions, conditioned by the omission of branch points, may be used to generate more efficient code.

5.3 Abbreviations

Several abbreviations are used in the descriptions that follow. These are:

1. D is used for the addressing width of a descriptor. In Oregon SNOBOL5, D is 16.

2. S is used for the addressing width of a specifier; S = 2D.

3. CPD is used for the number of characters stored per descriptor. This is 16 for Oregon SNOBOL5.

4. I is used for (signed) integers.

5. R is used for real numbers.

6. E is used for the address width of a syntax table entry.

7. Z is used to indicate the number of the last character in collating sequence. Characters are "numbered" from 0 to Z.

5.4 Data Type Codes

The SNOBOL5 system has data type codes assigned for integers and real numbers, among others. These codes are indicated in the macro descriptions by R and I respectively. These symbols are defined in the SIL source. Some of the codes are machine dependent. See the COPY, PARMS and MDATA macros.

5.5 Programming Notes

Programming notes are provided for some macro operations. The notes are intended to point out special cases, indicate implementation pitfalls, to show changes for Oregon SNOBOL5, and to provide information about conditions that can be used to improve the efficiency of the implementation.

5.6 Macro Prototypes

The following shows an example coding of a SIL macro and the generated expansion:

```
      RTNXA   AEQLC   ABC,YPTR,,THERE         Sample macro invocation

      RTNXA   lea     rax,YPTR
              cmp     qword ptr [ABC],rax
              je      THERE
```

5.7 Format of the SNOBOL5 Source File

One problem in implementing SNOBOL5 for a particular machine involves putting the macro-language program into a form suitable for the assembler for that machine. This typically involves making a number of format changes and correcting a few special cases by hand. It is desirable to perform as many changes as possible by some systematic, mechanical means (preferably with a program) so that new versions of the macro-language program can be converted into the required form easily, thus facilitating the incorporation of updates in the SNOBOL5 language. A systematic, mechanical technique also minimizes random errors inevitably introduced by human interference. Such random errors are particularly dangerous in such an implementation, since most of the logic of the system is at a level divorced from the implementation of the macro language. This section describes the format of the macro-language program in order to make the necessary format changes easier to determine.

The SNOBOL SIL assembly source file consists of text line images, reminiscent of the days of computer punch cards. A sequence number often appears at the end of the line which can be used to find the corresponding line in the old original SIL code. There are two kinds of lines: program text and comments. Comments have an asterisk (*) in column 1 followed by descriptive text. All other lines (about 3/4 of the source) are program text. Program text has a field format as follows:

1. Columns 1 starts the label field. A program label, if present, begins in column 1. All labels begin with a letter, followed by letters or digits and terminate with a blank or tab character (white space). If a program line has no label, column 1 is blank.

2. After the optional label and the following white space is the operation field. Operations consist of letters and numbers and terminate with white space.

3. The next non-blank field is the variable field. A list of operands appears in the variable field. The list consists of items separated by commas. The last item in the list is followed by a blank. If there are no operands, there is only a comma. Items in the operand list may take several forms:

a. Identifiers, which satisfy the requirements of program labels.

b. Integer constants.

c. Arithmetic expressions containing identifiers and constants.

d. Lists of items enclosed in parentheses. Lists are not nested, i.e. lists do not occur as items within lists.

e. Character literals, consisting of characters enclosed in single quotation marks. Quotation marks do not occur within literals, but commas, parentheses, and blanks may. This fact must be taken into account in analyzing the variable field.

f. Nulls, or items of zero length. Nulls represent explicitly omitted arguments to macro operations.

Comments may occur following the blank that terminates the variable field.

The following portion of program is typical.

```
*-------------------------------------------------------------------* 00000807
*                                                                     00000808
*      Block Marking                                                  00000809
*                                                                     00000810
GCM    PROC    ,                  Procedure to mark blocks          00000811
       POP     BK1CL              Restore block to mark from        00000812
       PUSH    ZEROCL             Save end marker                   00000813
GCMA1  GETSIZ  BKDX,BK1CL         Get size of block                 00000814
GCMA2  GETD    DESCL,BK1CL,BKDX   Get descriptor                    00000815
       TESTF   DESCL,PTR,GCMA3    Is it a pointer?                  00000816
       AEQLC   DESCL,0,,GCMA3     Is address zero?                  00000817
       TOP     TOPCL,OFSET,DESCL  Get to title of block pointed to  00000818
       TESTFI  TOPCL,MARK,GCMA4   Is block marked?                  00000819
GCMA3  DECRA   BKDX,DESCR         Decrement offset                  00000820
       AEQLC   BKDX,0,GCMA2       Check for end of block            00000821
       POP     BK1CL              Restore block pushed              00000822
       AEQLC   BK1CL,0,,RTN1      Check for end                     00000823
       SETAV   BKDX,BK1CL         Get size remaining                00000824
       BRANCH  GCMA2              Continue processing               00000825
*_                                                                  00000826
GCMA4  DECRA   BKDX,DESCR         Decrement offset                  00000827
       AEQLC   BKDX,0,,GCMA9      Check for end                     00000828
       SETVA   BK1CL,BKDX         Insert offset                     00000829
       PUSH    BK1CL              Save current block                00000830
GCMA9  MOVD    BK1CL,TOPCL        Set poiner to new block           00000831
       SETFI   BK1CL,MARK         Mark block                        00000832
       TESTFI  BK1CL,STTL,GCMA1   Is it a string?                   00000833
       MOVD    BKDX,TWOCL         Set size of string to 2           00000834
       BRANCH  GCMA2              Join processing                   00000835
```

==========

1. ACOMP (address comparison)

```
-------------------------------------------------------
|          ACOMP     DESCR1,DESCR2,GTLOC,EQLOC,LTLOC |
-------------------------------------------------------
```

ACOMP is used to compare the address fields of two descriptors. The comparison is unsigned arithmetic with A1 and A2 being considered as unsigned addresses. See ICOMP for the signed version for integers. If A1 > A2, transfer is to GTLOC. If A1 = A2, transfer is to EQLOC. If A1 < A2, transfer is to LTLOC.

Data Input to ACOMP

```
            ----------------------------
DESCR1      |   A1   |        |        |
            ----------------------------

            ----------------------------
DESCR2      |   A2   |        |        |
            ----------------------------
```

Programming Notes:

1. A1 and A2 may be relocatable addresses.

==========

2. ACOMPC (address comparison with constant)

```
-------------------------------------------------
|          ACOMPC    DESCR,N,GTLOC,EQLOC,LTLOC |
-------------------------------------------------
```

ACOMPC is used to compare the address field of a descriptor to a constant. The comparison is unsigned arithmetic with A being considered as an address. See ICOMPC for the version for a signed comparizon of integers. If A > N, transfer is to GTLOC. If A = N, transfer is to EQLOC. If A < N, transfer is to LTLOC.

Data Input to ACOMPC

```
            ----------------------------
DESCR       |   A    |        |        |
            ----------------------------
```

Programming Notes:

1. A may be a relocatable address.

2. N is never negative.

3. N is often 0.

==========

3. ADDLG (add to specifier length)

```
--------------------------------
|          ADDLG     SPEC,DESCR |
--------------------------------
```

ADDLG is used to add an integer to the length of a specifier.

Data Input to ADDLG

```
          ---------------------------------------------
SPEC      |        |        |        |        |   L    |
          ---------------------------------------------

          ----------------------------
DESCR     |   I    |        |        |
          ----------------------------
```

Data Altered by ADDLG

```
          ---------------------------------------------
SPEC      |        |        |        |        |  L+I   |
          ---------------------------------------------
```

Programming Notes:

1. I is always positive.

==========

4. ADDSIB (add sibling to tree node)

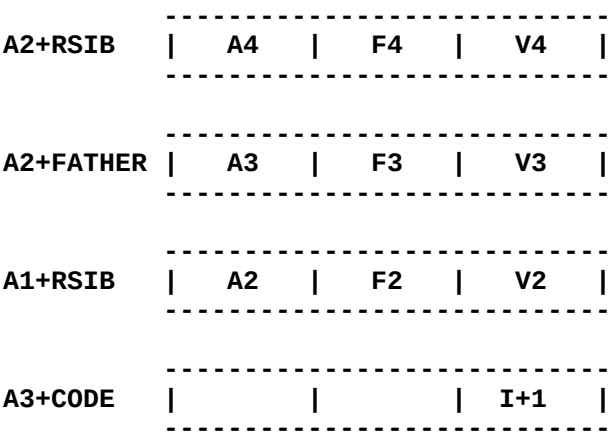
```
-----------------------------------
|          ADDSIB    DESCR1,DESCR2 |
-----------------------------------
```

ADDSIB is used to add a tree node as a sibling to another node.

Data Input to ADDSIB

```
          ----------------------------
DESCR1    |   A1   |        |        |
          ----------------------------

          ----------------------------
DESCR2    |   A2   |   F2   |   V2   |
          ----------------------------

          ----------------------------
A1+FATHER |   A3   |   F3   |   V3   |
          ----------------------------

          ----------------------------
A1+RSIB   |   A4   |   F4   |   V4   |
          ----------------------------

          ----------------------------
A3+CODE   |        |        |   I    |
          ----------------------------
```

Data Altered by ADDSIB



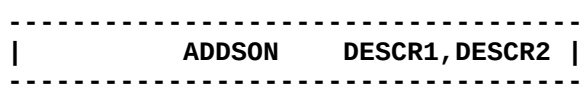
Programming Notes:

1. ADDSIB is only used by compilation procedures.

2. FATHER, RSIB, and CODE are symbols defined in the source program.

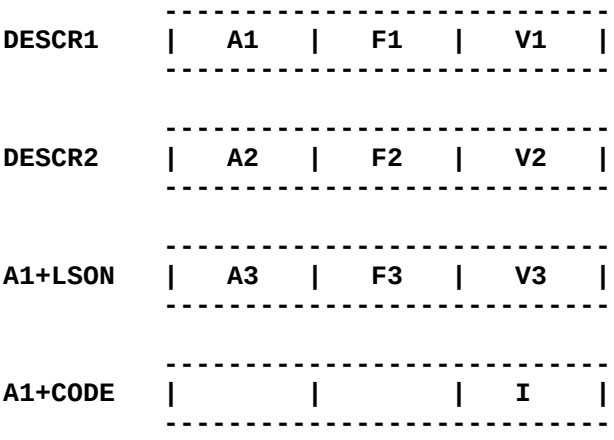
==========

5. ADDSON (add son to tree node)

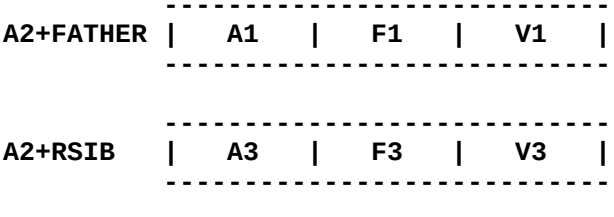


ADDSON is used to add a tree node as a son to another node.

Data Input to ADDSON



Data Altered by ADDSON



.
.

```
                    .
               ---------------------------
     A1+LSON   |   A2   |   F2   |   V2   |
               ---------------------------

               ---------------------------
     A1+CODE   |        |        |  I+1   |
               ---------------------------
```

Programming Notes:

1. ADDSON is only used by compilation procedures.

2. FATHER, LSON,RSIB, and CODE are symbols defined in the source program.

==========

6. ADJUST (compute adjusted address)

```
-----------------------------------------
|          ADJUST    DESCR1,DESCR2,DESCR3 |
-----------------------------------------
```

ADJUST is used to adjust the address field of a descriptor. The only use of this macro is in the garbage collector. Since SNOBOL5 uses a separate assembly routine for garbage collection, this macro is not used.

Data Input to ADJUST

```
               ---------------------------
     DESCR2    |   A2   |        |        |
               ---------------------------

               ---------------------------
     DESCR3    |   A3   |        |        |
               ---------------------------

               ---------------------------
     A2        |   A4   |        |        |
               ---------------------------
```

Data Altered by ADJUST

```
               ---------------------------
     DESCR1    | A3+A4  |        |        |
               ---------------------------
```

Programming Notes:

1. A3 is always an address integer.

==========

7. ADREAL (add real numbers)

```
-----------------------------------------------------
|          ADREAL    DESCR1,DESCR2,DESCR3,FLOC,SLOC |
-----------------------------------------------------
```

ADREAL is used to add two real numbers. If the result is out of the range available for real numbers, transfer is to FLOC. Otherwise transfer is to SLOC.

Data Input to ADREAL

DESCR2	R2	F2	V2
DESCR3	R3		

Data Altered by ADREAL

DESCR1	R2+R3	F2	V2

==========

8. AEQL (addresses equal test)

AEQL	DESCR1,DESCR2,NELOC,EQLOC

AEQL is used to compare the address fields of two descriptors. The comparison is arithmetic with A1 and A2 being considered as signed integers: If A1 = A2, transfer is to EQLOC. Otherwise transfer is to NELOC.

Data Input to AEQL

DESCR1	A1		
DESCR2	A2		

Programming Notes:

1. A1 and A2 may be relocatable addresses.

==========

9. AEQLC (address equal to constant test)

AEQLC	DESCR,N,NELOC,EQLOC

AEQLC is used to compare the address field of a descriptor to a constant. The comparison is unsigned arithmetic with A being considered as an address. See IEQLC for the signed arithmetic version for integers. If A = N, transfer is to EQLOC. Otherwise transfer is to NELOC.

Data Input to AEQLC

DESCR	A		

Programming Notes:

1. A may be a relocatable address.

2. N is never negative.

3. N is often 0.

==========

10. AEQLIC (address equal to constant indirect test)

```
-------------------------------------------
|          AEQLIC    DESCR,N1,N2,NELOC,EQLOC |
-------------------------------------------
```

AEQLIC is used to compare an indirectly specified address field of a descriptor to a constant. The comparison is arithmetic with A1 being considered as a signed integer. If A2 = N2, transfer is to EQLOC. Otherwise transfer is to NELOC.

Data Input to AEQLIC

```
            ----------------------------
DESCR       |   A1   |        |        |
            ----------------------------

            ----------------------------
A1+N1       |   A2   |        |        |
            ----------------------------
```

Programming Notes:

1. A2 may be a relocatable address.

2. N2 is never negative.

3. N1 is always zero.

==========

11. ANDFUN (perform a logical and function)

```
---------------------------------------
|          ANDFUN    SPEC1,SPEC2,SPEC3 |
---------------------------------------
```

ANDFUN is used to logically AND the bytes specified by strings SPEC2 and SPEC3 and place the result in SPEC1. If either SPEC2 or SPEC3 is shorter than the other, then the remaining characters of the shorter string are assumed to be zero bytes. F(Cn) = logical AND of C2n and C3n.

Data Input to ANDFUN

```
            ------------------------------------------------
SPEC1       |   A1   |        |        |   O1   |MAX(L2,L3)|
            ------------------------------------------------

            ------------------------------------------
SPEC2       |   A2   |        |        |   O2   |   L2   |
            ------------------------------------------

            ------------------------------------------
SPEC3       |   A3   |        |        |   O3   |   L3   |
            ------------------------------------------
```

```
               ----------------------------
     A2+O2     |  C21   |  ...   |  C2L2  |
               ----------------------------

               ----------------------------
     A3+O3     |  C31   |  ...   |  C3L2  |
               ----------------------------
```

Data Altered by ANDFUN

```
               ------------------------------------
     A1+O1     | F(C1)  |  ...   |F(C(MAX(L2,L3)))|
               ------------------------------------
```

Programming Notes:

1. L2 and L3 may be zero.

==========

12. APDSP (append specifier)

```
---------------------------------
|          APDSP     SPEC1,SPEC2 |
---------------------------------
```

APDSP is used to append one specified string to another specified string.

Data Input to APDSP

```
               ------------------------------------------------
     SPEC1     |   A1   |        |        |   O1   |   L1   |
               ------------------------------------------------

               ------------------------------------------------
     SPEC2     |   A2   |        |        |   O2   |   L2   |
               ------------------------------------------------

               ----------------------------
     A1+O1     |  C11   |  ...   |  C1L1  |
               ----------------------------

               ----------------------------
     A2+O2     |  C21   |  ...   |  C2L2  |
               ----------------------------
```

Data Altered by APDSP

```
               ------------------------------------------------
     SPEC1     |   A1   |        |        |   O1   | L1+L2  |
               ------------------------------------------------

               ------------------------------------------------------
     A1+O1     |  C11   |  ...   |  C1L1  |  C21   |  ...   |  C2L2  |
               ------------------------------------------------------
```

Programming Notes:

1. If L1 = 0, C21 is placed at A1+O1.

2. The storage following C1L1 is always adequate for C21...C2L2.

==========

13. ARRAY (assemble array of descriptors)

```
-----------------------
| L        ARRAY    N |
-----------------------
```

ARRAY is used to assemble an array of descriptors.

Data Assembled by ARRAY

```
          ----------------------------
L         |   0    |   0    |   0    |
          ----------------------------

                      .
                      .
                      .
          ----------------------------
L+(N-1)*D |   0    |   0    |   0    |
          ----------------------------
```

Programming Notes:

1. All fields of all descriptors assembled by ARRAY must be zero when program execution begins.

==========

14. ATAN (arc tangent)

```
-----------------------------------
|          ATAN      DESCR1,DESCR2 |
-----------------------------------
```

ATAN is used to take the arc tangent of a REAL number. The result is in radians ranging from -PI/2 to +PI/2.

Data Input to ATAN

```
          ----------------------------
DESCR2    |   A    |        |   R    |
          ----------------------------
```

Data Altered by ATAN

```
          ----------------------------
DESCR1    |ATAN(A) |   0    |   R    |
          ----------------------------
```

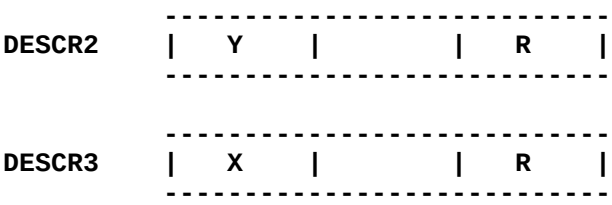
==========

15. ATAN2 (arc tangent)

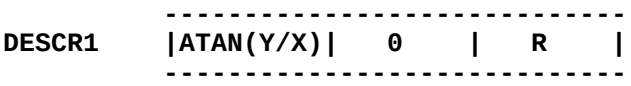
```
-----------------------------------
|          ATAN2     DESCR1,DESCR2 |
-----------------------------------
```

ATAN2 is used to take the arc tangent of a REAL x,y coordinate. The result is in radians ranging from -PI to +PI.

Data Input to ATAN

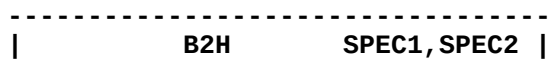


Data Altered by ATAN



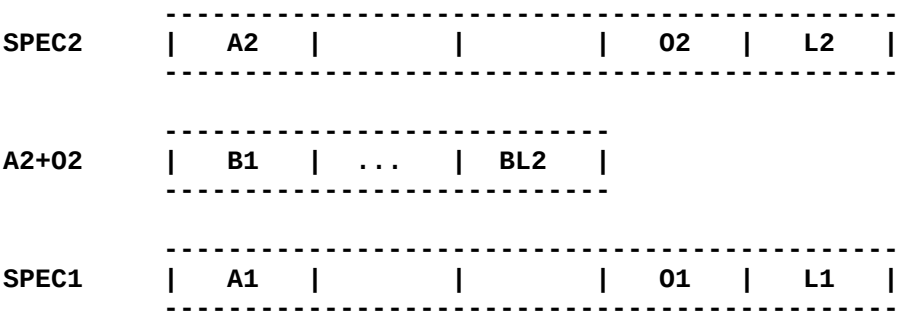
==========

16. B2H (convert bits string to hexadecimal string)



B2H is used to convert a bit string to a hexadecimal character string. The bit string is actually a character string from which the bit values are derived from the low order bit of each character. C1 is formed by concatenating bits from B1 through B4, with B4 as the low order bit. Then this hexadecimal value is converted to one of the appropriate characters: 0123456789ABCDEF. In all cases L1*4 >= L2. If L1*4 is not equal to L2, then the bits specified beyond BL2 should be considered to be zero bits.

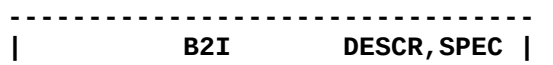
Data Input to B2H



Data Altered by B2H

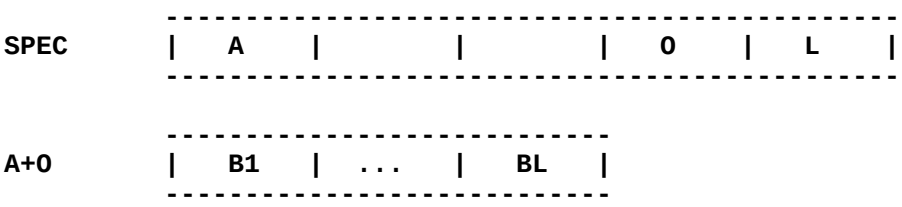
==========

17. B2I (convert bits to an integer)

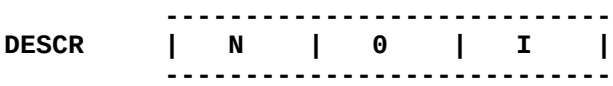


B2I is used to convert a bit string to an integer. The bit string is actually a character string from which the bit values are derived from the low order bit of each character. N is formed by concatenating bits from B1 through BL, with BL as the low order bit. In all cases ALENG*BITSPA >= L. If ALENG*BITSPA is not equal to L, then the bits specified before B1 should be considered to be zero bits, padding N in the high order positions.

Data Input to B2I

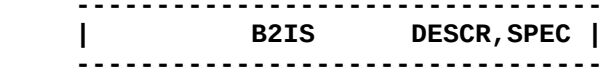


Data Altered by B2I



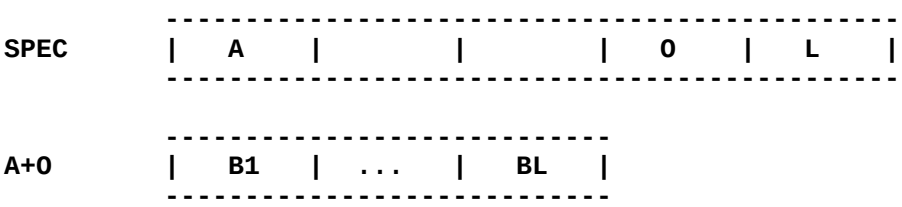
==========

18. B2IS (convert bits to an integer, sign extended)



B2IS is used to convert a bit string to an integer. The bit string is actually a character string from which the bit values are derived from the low order bit of each character. N is formed by concatenating bits from B1 through BL, with BL as the low order bit. In all cases ALENG*BITSPA >= L. If ALENG*BITSPA is not equal to L, then the bits specified before B1 should be considered to be the same as B1, padding N in the high order positions.

Data Input to B2IS



Data Altered by B2IS

==========

19. B2R (convert bits to an real number)

```
---------------------------------
|          B2R       DESCR,SPEC |
---------------------------------
```

B2R is used to convert a bit string to an REAL value. The bit string is actually a character string from which the bit values are derived from the low order bit of each character. N is formed by concatenating bits from B1 through BL, with BL as the low order bit. In all cases ALENG*BITSPA = L.

Data Input to B2R

```
          ---------------------------------------------
SPEC      |   A    |        |        |   O    |   L    |
          ---------------------------------------------

          ----------------------------
A+O       |   B1   |  ...   |   BL   |
          ----------------------------
```

Data Altered by B2R

```
          ----------------------------
DESCR     |   N    |   0    |   R    |
          ----------------------------
```

==========

20. B2S (convert bits to character string)

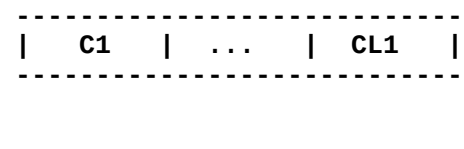
```
----------------------------------
|          B2S       SPEC1,SPEC2 |
----------------------------------
```

B2S is used to convert a bit string to a character string. The bit string is actually a character string from which the bit values are derived from the low order bit of each character. C1 is formed by concatenating bits from B1 through B8, with B8 as the low order bit. In all cases L1*BITSPA >= L2. If L1*BITSPA is not equal to L2, then the bits specified beyond BL2 should be considered to be zero bits.

Data Input to B2S

```
          ---------------------------------------------
SPEC2     |   A2   |        |        |   O2   |   L2   |
          ---------------------------------------------

          ----------------------------
A2+O2     |   B1   |  ...   |  BL2   |
          ----------------------------

          ---------------------------------------------
SPEC1     |   A1   |        |        |   O1   |   L1   |
          ---------------------------------------------
```

Data Altered by B2S

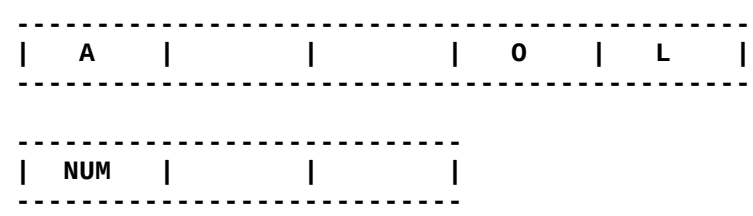


==========

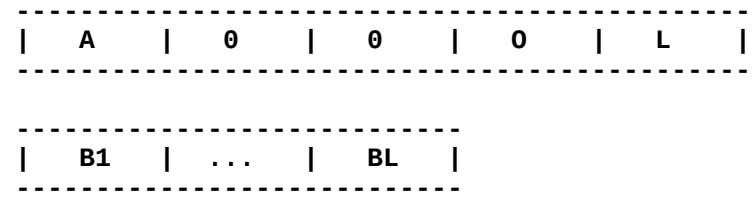
21. BITI (convert numeric to bits)

BITI is used to expand an integer or real into its big endian bit representation. L should be the number of bits in the NUM field of a descriptor. B1 should be the high order bit of NUM. BL should be the low order bit of NUM. NUM is either an integer or real. L is ALENG*BITSPA, 64 for SNOBOL5. Each of the B1...BL should be either the "0" or "1" character. In bit operations, only the low order bit of each result byte will be considered. So in an ASCII system "P" would be equivalent to "0" and "Q" would be equivalent to "1" in the bit string. This is not recommended however.

Data Input to BITI



Data Altered by BITI

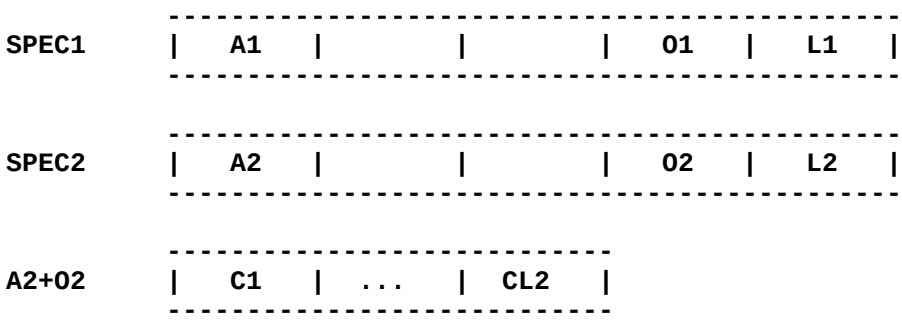


==========

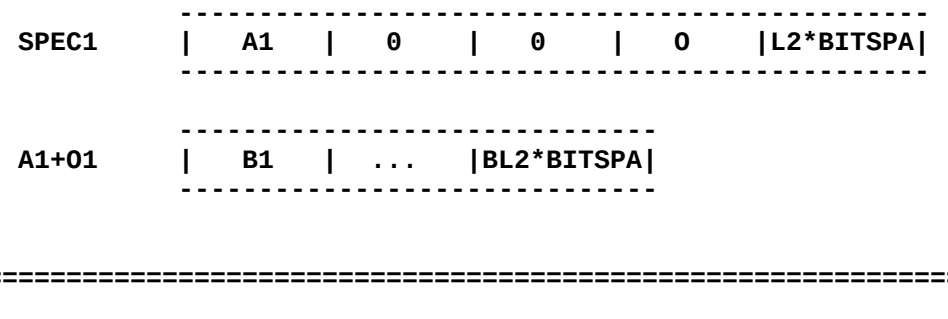
22. BITS (convert string to bits)

BITS is used to expand a string into its bits representation. L1 should be L2 times BITSPA. B1 should be the high order bit of C1. B(BITSPA) should be the low order bit of C1. Each of the B1...BL1 should be either the "0" or "1" character. In bit operations, only the low order bit of each result byte will be considered. So in an ASCII system "P" would be equivalent to "0" and "Q" would be equivalent to "1" in the bit string. This is not recommended however. BITSPA is the number of bits per machine address of storage, usually 8. Thus a bits representation of a character string will take at least 8 times more storage. This approach to bit string is used to eliminate complex bit shifting that would otherwise be required.

Data Input to BITS



Data Altered by BITS



==========

23. BKSIZE (get block size)

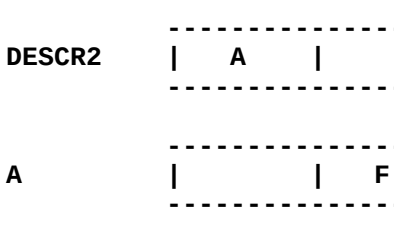
BKSIZE	DESCR1,DESCR2

BKSIZE is used to determine the amount of storage occupied by a block or string structure. This macro is only used in the garbage collector and is not implemented in SNOBOL5 since the garbage collector is a separate assembly routine. The flag field of the descriptor at A distinguishes between string structures and blocks. If F contains the flag STTL, then

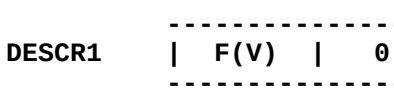
$$F(V)=D*(4+[((V-1)/CPD)+1])=(((V+31)/8)*8)+8$$

where [x] is the integer part of x and CPD is the number of characters stored per descriptor. The constant 4 occurs because there are 4 descriptors (including the title) in a string structure in addition to the string itself. The expression in brackets represents the number of descriptors required for a string of V characters. If F does not contain the flag STTL, then F(V) = V+D.

Data Input to BKSIZE



Data Altered by BKSIZE



==========

24. BKSPCE (backspace record)

```
----------------------------
|          BKSPCE    DESCR |
----------------------------
```

BKSPCE is used to back space one record on the file associated with unit reference number I.

Data Input to BKSPCE

DESCR	I		

Programming Notes:

1. Refer to Section 2.1 for a discussion of unit reference numbers.

2. Not implemented in Oregon SNOBOL5.

==========

25. BRANCH (branch to program location)

```
------------------------------
|          BRANCH    LOC,PROC |
------------------------------
```

BRANCH is used to alter the flow of program control by branching to LOC. If PROC is given, it is the procedure in which LOC occurs. If PROC is omitted, LOC is in the current procedure. PROC is ignored in SNOBOL5.

==========

26. BRANIC (branch indirect with offset constant)

```
------------------------------
|          BRANIC    DESCR,N |
------------------------------
```

BRANIC is used to alter the flow of program control by branching indirectly to the operation at code address LOC.

Data Input to BRANIC

DESCR	A		
A+N	LOC		

Programming Notes:

1. N is always zero

==========

27. BUFFER (assemble buffer of blank characters)

```
-----------------------
| LOC      BUFFER    N |
-----------------------
```

BUFFER is used to assemble a string of N blank characters.

Data Assembled by BUFFER

Programming Notes:

1. All characters of the string assembled by BUFFER must be blank (not zero) when program execution begins.

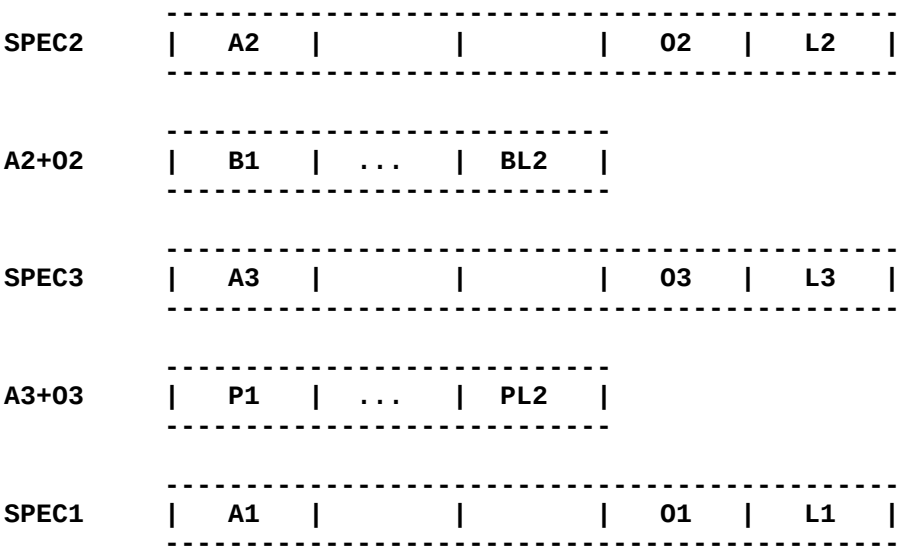
==========

28. CENTER (centers a string by padding on the left and right)

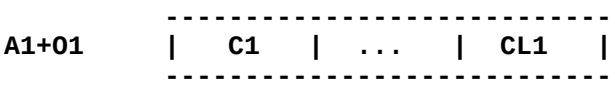
```
---------------------------------------
|          CENTER    SPEC1,SPEC2,SPEC3 |
---------------------------------------
```

CENTER is used to pad a string on the left and right to center it. The pad character is P1 unless L3 is zero, in which case the pad character is a blank. L1 >= L2 in all cases. If an unequal number of pad characters are to be inserted on the left and right, then the right shall have the extra pad character. If L1=L2, then there is no padding and the source string B1...BL2 is copied to C1...CL1.

Data Input to CENTER



Data Altered by CENTER



==========

29. CHKVAL (check value)

```
-----------------------------------------------------------
|          CHKVAL    DESCR1,DESCR2,SPEC,GTLOC,EQLOC,LTLOC |
-----------------------------------------------------------
```

CHKVAL is used to compare an integer to the length of a specifier plus another integer. If L+I2 > I1, transfer is to GTLOC. If L+I2 = I1, transfer is to EQLOC. If L+I2 < I1, transfer is to LTLOC.

Data Input to CHKVAL

```
           ---------------------------------------------
SPEC       |        |        |        |        |   L    |
           ---------------------------------------------

           ----------------------------
DESCR1     |   I1   |        |        |
           ----------------------------

           ----------------------------
DESCR2     |   I2   |        |        |
           ----------------------------
```

Programming Notes:

1. I1, I2, and L are always positive integers.

2. CHKVAL is used only in pattern matching.

==========

30. CLERTB (clear syntax table)

```
--------------------------------
|          CLERTB    TABLE,KEY |
--------------------------------
```

CLERTB is used to set the indicator fields of all entries of a syntax table to a constant. KEY may be one of four values:

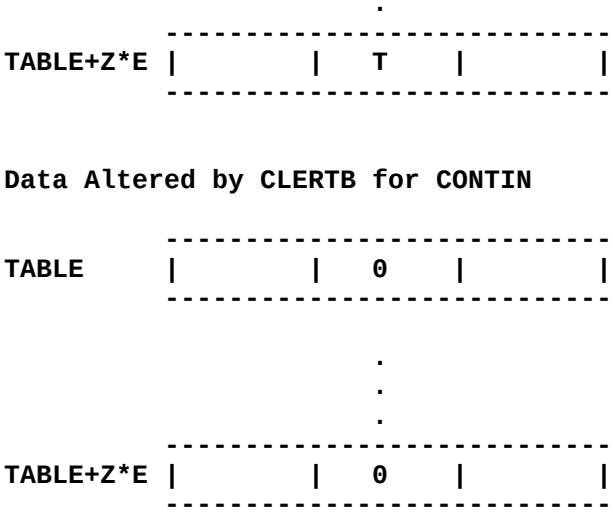
CONTIN = 1
ERROR = 2
STOP = 3
STOPSH = 4

The indicator field of each entry of TABLE is set to T where T is the indicator that corresponds to the value of KEY. The TABLE can be implemented as just an array of bytes outside of the work space. If the Flag fields of descriptors are used, one must be careful to not have the values stored there turning on the TTL, STTL or PRTF flags. E is one fewer than the number of possible characters storable in a byte. For SNOBOL5 this E=255 and Z=1. TABLE is always "SNABTB".

Data Altered by CLERTB for ERROR, STOP, or STOPSH

```
           ----------------------------
TABLE      |        |   T    |        |
           ----------------------------
```

.
.



Programming Notes:

1. See Section 4.2.

==========

31. COPY (copy file into assembly)

COPY is used to copy a file of machine-dependent data into the source program. COPY occurs three times in the assembly:

```
COPY    MDATA
COPY    MLINK
COPY    PARMS
```

MLINK and PARMS are copied at the beginning of the SNOBOL5 assembly. MDATA is copied in the data region.

MDATA is a file of machine-dependent data. It contains data used in the implementation of the macros and for strings that depend on the character set of an individual machine or that represent other problems that prevent a machine-independent representation. These are:

1. ALPHA, a string that consists of all characters arranged in the order of their internal numerical representation (collating sequence).

2. AMPST, a string consisting of a single ampersand, or whatever character is used to represent the keyword operator in the source language.

3. COLSTR, a string of two characters consisting of a colon followed by a blank.

4. QTSTR, a string consisting of a single quotation mark, or whatever character is used to represent a quotation mark in the source language.

These strings of characters are pointed to by the specifiers ALPHSP, AMPSP, COLSP, and QTSP respectively.

MLINK is a file of entry points and external symbol names that describe linkages used to access machine-language subroutines and I/O packages.

PARMS is a file of machine-dependent constants (equivalences). It contains constants used in the implementation of the macros and definitions of symbols. These are:

1. ALPHSZ, the number of characters in the character set for the machine. (ALPHSZ is 256 for the IBM PC.)

2. CPA, the number of characters per machine addressing unit. (CPA is 1 for the IBM PC, i.e., one character per byte.)

3. DESCR, the address width of a descriptor.

4. FNF, a flag used to identify function descriptors.

5. MARK, a flag used to identify descriptors that are marked titles during Garbage Collection.

6. VISITED, a flag used to identify descriptors that have been visited during Garbage Collection.

7. PTRF, a flag used to identify descriptors pointing into SNOBOL5 dynamic storage.

8. SIZLIM, the value of the largest integer that can be stored in the value field of a descriptor.

9. SPEC, the address width of a specifier.

10. STTL, a flag used to identify descriptors that are titles of string structures.

11. TTL, a flag used to identify descriptors that are titles of blocks.

12. UNITI, the number of the standard input unit. UNITI is 5 for the Oregon SNOBOL5 implementation.

13. UNITO, the number of the standard print output unit. UNITO is 6 for the Oregon SNOBOL5 implementation.

14. UNITP, the number of the standard punch output unit. UNITP is 7 for the Oregon SNOBOL5 implementation.

CSTACK and OSTACK, the current end old stack pointers, respectively, should be defined in one of the COPY files. These pointers may either be in registers or in the address fields of descriptors, depending on how the stack management macros are implemented (see PUSH and RCALL, e.g.). If these pointers are implemented as registers, they should be defined in PARMS. If they are implemented in storage locations, they should be defined in MDATA.
Programming Notes:

1. COPY may be implemented in a variety of ways. COPY may, for example, simply expand into the data required, depending on the value of its argument as given above.

2. Any of the COPY segments can be used to incorporate other machine-dependent data.

==========

32. CPYPAT (copy pattern)

```
-------------------------------------------------------------
|          CPYPAT    DESCR1,DESCR2,DESCR3,DESCR4,DESCR5,DESCR6 |
-------------------------------------------------------------
```

CPYPAT is used to copy a pattern. First set

```
R1 = A1
R2 = A2
R3 = A6
```

where R1, R2, and R3 are temporary locations. Sections of the pattern are copied for successive values of R1 and R2. After copying each section, set

```
R3 = R3-(1+V7)*D
```

Then set

```
R1 = R1+(1+V7)*D
R2 = R2+(1+V7)*D
```

If R3 > 0, continue, copying the next section. Otherwise the operation is complete. The final value of R1 is inserted in the address field of DESCR1.

The functions F1 and F2 are defined as follows:

```
F1(X) = 0 if X = 0
F1(X) = X+A4 otherwise

F2(X) = A5 if X = 0
F2(X) = X+A4 otherwise
```

Initial Data Input to CPYPAT

DESCR1	A1		
DESCR2	A2		
DESCR3	A3		
DESCR4	A4		
DESCR5	A5		
DESCR6	A6		

Data Input to CPYPAT for Successive Values of R2

R2+D	A7	F7	V7
R2+2D	A8	0	V8

R2+3D	A9	0	V9

Data Altered by CPYPAT for Successive Values of R1

R1+D	A7	F7	V7
R1+2D	F1(A8)	0	F2(V8)
R1+3D	A9+A3	0	V9+A3

Additional Data Input for Successive Values of R2 if V7 = 3

R2+4D	A10	F10	V10

Additional Data Altered for Successive Values of R1 if V7 = 3

R1+4D	A10	F10	V10

Data Altered when Copying is Complete

DESCR1	R1		

==========

33. COS (cosine)

COS	DESCR1,DESCR2

COS is used to take the cosine of an angle in radians.

Data Input to COS

DESCR2	A		R

Data Altered by COS

DESCR1	ATAN(A)	0	R

==========

34. DATE (get date)

```
--------------------------
|          DATE      SPEC |
--------------------------
```

DATE is used to obtain the current date. A character representation of the current date is placed in BUFFER.

Data Altered by DATE

```
                ---------------------------------------------
SPEC            | BUFFER |   0    |   0    |   0    |   L    |
                ---------------------------------------------

                ----------------------------
BUFFER          |   C1   |  ...   |   CL   |
                ----------------------------
```

Programming Notes:

1. The choice of representation for the date is not important so far as the source language is concerned.

04-01-81 12:30:29.123

is returned by Oregon SNOBOL5, which includes the time. The accuracy may be less than shown by the number of digits.

2. BUFFER is local to DATE and its old contents may be overwritten by a subsequent use of DATE.

3. DATE is used only in the SNOBOL5 DATE function.

4. Implementation of DATE, as such, is not essential. In this case, DATE should set the length of SPEC to zero and do nothing else.

==========

35. DECRA (decrement address)

```
------------------------------
|          DECRA     DESCR,N |
------------------------------
```

DECRA is used to decrement the address field of a descriptor. A is considered an unsigned address. See DECRI for a signed integer version.

Data Input to DECRA

```
                ----------------------------
DESCR           |   A    |        |        |
                ----------------------------
```

Data Altered by DECRA

```
                ----------------------------
DESCR           |  A-N   |        |        |
                ----------------------------
```

Programming Notes:

1. A maybe a relocatable address.

2. N is always positive.

3. N is often 1 or D.

==========

36. DECRI (decrement integer)

```
------------------------------
|          DECRI     DESCR,N |
------------------------------
```

DECRI is used to decrement the integer in the address field of a descriptor. A is considered to be a signed integer. See DECRA for an address computation version.

Data Input to DECRI

```
          ----------------------------
DESCR     |   I    |        |        |
          ----------------------------
```

Data Altered by DECRI

```
          ----------------------------
DESCR     |  I-N   |        |        |
          ----------------------------
```

Programming Notes:

1. N is often 1 or D.

2. A-N may be negative. numbers.

==========

37. DEQL (descriptor equal test)

```
-------------------------------------------------
|          DEQL      DESCR1,DESCR2,NELOC,EQLOC |
-------------------------------------------------
```

DEQL is used to compare two descriptors. If A1 = A2, F1 = F2, and V1 = V2, transfer is to EQLOC. Otherwise transfer is to NELOC.

Data Input to DEQL

```
          ----------------------------
DESCR1    |   A1   |   F1   |   V1   |
          ----------------------------

          ----------------------------
DESCR2    |   A2   |   F2   |   V2   |
          ----------------------------
```

Programming Notes:

1. All fields of the two descriptors must be identical for transfer to EQLOC.

==========

38. DESCR (assemble descriptor)

```
----------------------------
| LOC      DESCR     A,F,V |
----------------------------
```

DESCR assembles a descriptor with specified address, flag, and value fields.

Data Assembled by DESCR

LOC	A	F	V

Programming Notes:

1. Any or all of A, F, and V may be omitted. A zero field must be assembled when the corresponding argument is omitted.

==========

39. DHERE (define location in the data area)

```
-----------------------
| LOC      DHERE      |
-----------------------
```

DHERE is used to establish the equivalence of LOC as the location of the next data address. It only applies to the data declarations area and not instructions in contrast to LHERE.

Programming Notes:

1. DHERE is equivalent to the familiar EQU *. Similarly

```
      LOC     DHERE
              STRING  'hello'
```

is equivalent to

```
      LOC     STRING  'hello'
```

==========

40. DIVIDE (divide integers)

```
-----------------------------------------------------
|          DIVIDE    DESCR1,DESCR2,DESCR3,FLOC,SLOC |
-----------------------------------------------------
```

DIVIDE is used to divide one integer by another. Any remainder is discarded. That is, the result is truncated, not rounded. If I = 0, transfer is to FLOC. Otherwise transfer is to SLOC.

Data Input to DIVIDE

DESCR2	A	F	V

```
          ----------------------------
DESCR3    |   I    |        |        |
          ----------------------------
```

Data Altered by DIVIDE

```
          ----------------------------
DESCR1    |  A/I   |   F    |   V    |
          ----------------------------
```

Programming Notes:

1. A may be a relocatable address.

==========

41. DVREAL (divide real numbers)

```
-----------------------------------------------------
|          DVREAL    DESCR1,DESCR2,DESCR3,FLOC,SLOC |
-----------------------------------------------------
```

DVREAL is used to divide one real number by another. If R3 = 0 or the result is out of the range available for real numbers, transfer is to FLOC. Otherwise transfer is to SLOC.

Data Input to DVREAL

```
          ----------------------------
DESCR2    |   R2   |   F2   |   V2   |
          ----------------------------

          ----------------------------
DESCR3    |   R3   |        |        |
          ----------------------------
```

Data Altered by DVREAL

```
          ----------------------------
DESCR1    | R2/R3  |   F2   |   V2   |
          ----------------------------
```

Programming Notes:

1. In addition to use in source-language arithmetic, DVREAL is used in the computation of statistics published at the end of a SNOBOL5 run.

==========

42. END (end assembly)

```
-----------------------
|          END        |
-----------------------
```

END is used to terminate assembly of the SNOBOL5 system. It occurs only once and is the last line of the assembly.

==========

43. ENDEX (end execution of SNOBOL5 run)

```
---------------------------
|          ENDEX     DESCR |
---------------------------
```

ENDEX is used to terminate execution of a SNOBOL5 run. ENDEX is the last instruction executed and is responsible for returning properly to the environment that initiated the SNOBOL5 run. If I is nonzero, a postmortem dump of user memory should be given. This is not done in SNOBOL5. The value of &ABEND is returned as a return code of the SNOBOL5 run. There may be limits on what the maximum return value can be, depending on the operating system. ENDEX also closes any open I/O units and deallocates all storage allocated by SNOBOL5.

Data Input to ENDEX

```
             ----------------------------
DESCR        |   I    |        |        |
             ----------------------------
```

Programming Notes:

1. If a dump is not given, the keyword &ABEND will not have its specified effect. Nothing else will be affected.

2. On the IBM System/360, if I is nonzero, an abend dump is given with a user code of I.

==========

44. ENFILE (write end of file)

```
----------------------------
|          ENFILE    DESCR |
----------------------------
```

ENFILE is used to write an end-of-file on (close) the file associated with unit reference number I.

Data Input to ENFILE

```
             ----------------------------
DESCR        |   I    |        |        |
             ----------------------------
```

Programming Notes:

1. Refer to Section 2.1 for a discussion of unit reference numbers.

==========

45. EQU (define symbol equivalence)

```
------------------------
| SYMBOL   EQU       N |
------------------------
```

EQU is used to assign, at assembly time, the value of N to SYMBOL.

==========

46. EQUD (define symbol equivalence)

```
-------------------------
| SYMBOL   EQUD      LOC |
-------------------------
```

EQUD is similar to EQU, but is used in the data area to assign a value to the SYMBOL from an expression LOC.

==========

47. EXPINT (exponentiate integers)

```
-----------------------------------------------------
|          EXPINT    DESCR1,DESCR2,DESCR3,FLOC,SLOC |
-----------------------------------------------------
```

EXPINT is used to raise an integer to an integer power. If I1 = 0 and I2 is not positive, or if the result is out of the range available for integers, transfer is to FLOC. Otherwise transfer is to SLOC.

Data Input to EXPINT

```
           ----------------------------
DESCR2     |   I1   |   F    |   V    |
           ----------------------------

           ----------------------------
DESCR3     |   I2   |        |        |
           ----------------------------
```

Data Altered by EXPINT

```
           ----------------------------
DESCR1     | I1**I2 |   F    |   V    |
           ----------------------------
```

==========

48. EXREAL (exponentiate real numbers)

```
-----------------------------------------------------
|          EXREAL    DESCR1,DESCR2,DESCR3,FLOC,SLOC |
-----------------------------------------------------
```

EXREAL is used to raise a real number to a real power. If the result is not a real number or is out of the range available for real numbers, transfer is to FLOC. Otherwise transfer is to SLOC.

Data Input to EXREAL

```
           ----------------------------
DESCR2     |   R1   |   F    |   V    |
           ----------------------------

           ----------------------------
DESCR3     |   R2   |        |        |
           ----------------------------
```

Data Altered by EXREAL

DESCR1	R1**R2	F	V

==========

49. FILNAM (set file name for I/O unit number)

FILNAM	DESCR,SPEC,SPEC2,FLOC,SLOC

FILNAM is used to set the file name for an I/O unit number. The file name (C1...CL) should follow the rules for names in the system. The attributes string (C21...C2L2) is a string of optional file modifiers. If there is a problem parsing or setting the file name, transfer is to FLOC. Otherwise transfer is to SLOC.

Data Input to FILNAM

DESCR	I					
SPEC	A			O	L	
A+O	C1	...	CL			
SPEC2	A2			O2	L2	
A2+O2	C21	...	C2L2			

Programming Notes:

1. This is a new macro for Oregon SNOBOL5.

==========

50. FORMAT (assemble format string)

LOC	FORMAT	'C1...CL'

FORMAT is used to assemble the characters of a format.

Data Assembled by FORMAT

LOC	C1	...	CL

Programming Notes:

1. The characters assembled by FORMAT are treated as an "undigested" format by the I/O routines.

2. In Oregon SNOBOL5, the format is terminated with a zero byte. Substitution values are indicated by a backslash (\).

\#	substitute integer value here
\F	substitute float value here
\^L	replace with form feed character
\^M	replace with carriage return character
\^J	replace with line feed character

==========

51. FSHRTN (foreshorten specifier)

FSHRTN	SPEC,N

FSHRTN is used to exclude initial characters from a string specification.

Data Input to FSHRTN

SPEC				O	L

Data Altered by FSHRTN

SPEC				O+N	L-N

Programming Notes:

1. L-N is never negative.

==========

52. FUNC (Extra function)

D####	D####

is used

Data Input to D####

D####	I		

Programming Notes:

==========

53. GETAC (get address with offset constant)

```
-------------------------------------
|          GETAC     DESCR1,DESCR2,N |
-------------------------------------
```

GETAC is used to get an address field with an offset constant.

Data Input to GETAC

DESCR2	A2			
A2+N	A			

Data Altered by GETAC

DESCR1	A			

Programming Notes:

1. N may be negative.

==========

54. GETBAL (get parenthesis balanced string)

```
-----------------------------------------
|          GETBAL    SPEC,DESCR,FLOC,SLOC |
-----------------------------------------
```

GETBAL is used to get the specification of a balanced substring. The string starting at CL+1 and ending at CL+N is examined to determine the shortest balanced substring CL+1,...,CL+J. J is determined according to the following rules:

If CL+1 is not a parenthesis, then J = 1.

If CL+1 is a left parenthesis, then J is the least integer such that CL+1...CL+J is balanced with respect to parentheses in the usual algebraic sense.

If CL+1 is a right parenthesis, or if no such balanced string exists, transfer is to FLOC. Otherwise SPEC is modified as indicated and transfer is to SLOC.

Data Input to GETBAL

SPEC	A			O	L	
DESCR	N					

A+O	C1	...	CL	CL+1	...	CL+N	

Data Altered by GETBAL

SPEC	A			O	L+J

==========

55. GETD (get descriptor)

```
-----------------------------------------
|          GETD      DESCR1,DESCR2,DESCR3 |
-----------------------------------------
```

GETD is used to get a descriptor.

Data Input to GETD

DESCR2	A2		
DESCR3	A3		
A2+A3	A	F	V

Data Altered by GETD

DESCR1	A	F	V

==========

56. GETDC (get descriptor with offset constant)

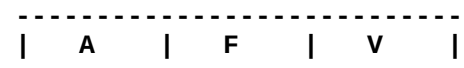
```
--------------------------------------
|          GETDC     DESCR1,DESCR2,N |
--------------------------------------
```

GETDC is used to get a descriptor with an offset constant.

Data Input to GETDC

DESCR2	A2		
A2+N	A	F	V

Data Altered by GETDC

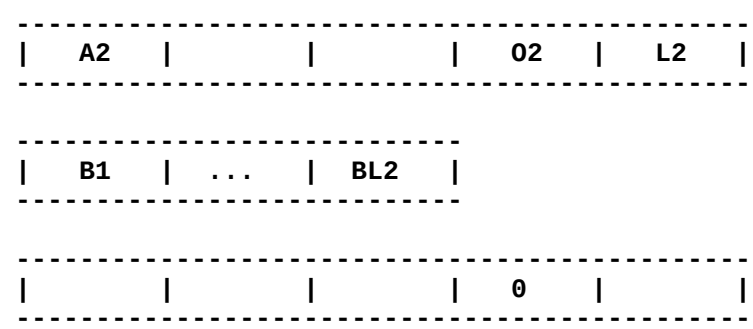


==========

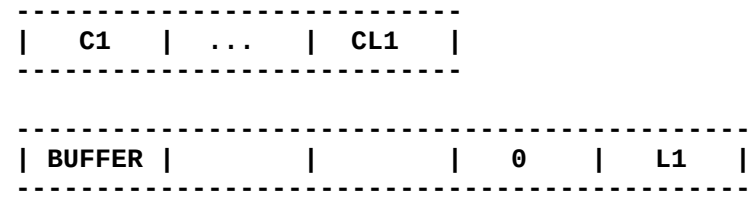
57. GETENV (retrieves an environment variable's value)

GETENV is used to retrieve the value of an environment variable. The name of the environment variable is B1...B(L2). The value is placed in a BUFFER area. The address of the buffer is placed in the address field for specifier 1. The length of the data is plaed in specifier 1 as L1. If the environment variable named was not found, then L1 is set to zero.

Data Input to GETENV

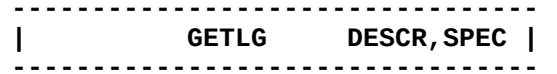


Data Altered by GETENV



==========

58. GETLG (get length of specifier)



GETLG is used to get the length of a specifier.

Data Input to GETLG

Data Altered by GETLG

DESCR	L	0	0

==========

59. GETLTH (get length for string structure)

```
-------------------------------------
|          GETLTH    DESCR1,DESCR2 |
-------------------------------------
```

GETLTH is used to determine the amount of storage required for a string structure. The amount of storage is given by the formula

F(L)=D*(3+[((L-1)/CPD)+1])
For SNOBOL5 this is:
F(L)=D*(3+[((L-1)/8)+1])=(L+63)&FFFFFFFFFFFFFFF0h

where [x] is the integer part of x and CPD is the numbers of characters stored per descriptor. The constant 3 accounts for the three descriptors in a string structure in addition to the string itself. The expression in brackets represents the number of descriptors required for a string of L characters.

Data Input to GETLTH

DESCR2	L		

Data Altered by GETLTH

DESCR1	F(L)	0	0

==========

60. GETSIZ (get size)

```
-------------------------------------
|          GETSIZ    DESCR1,DESCR2 |
-------------------------------------
```

GETSIZ is used to get the size from the value field of a title descriptor.

Data Input to GETSIZ

DESCR2	A		
A			V

Data Altered by GETSIZ

DESCR1	V	0	0

==========

61. GETSPC (get specifier with constant offset)

```
-----------------------------------
|          GETSPC    SPEC,DESCR,N |
-----------------------------------
```

GETSPC is used to get a specifier.

Data Input to GETSPC

DESCR	A1					
A1+N	A	F	V	O	L	

Data Altered by GETSPC

SPEC	A	F	V	O	L	

==========

62. GETSTD (get descriptor with constant offset from stack)

```
--------------------------------------
|          GETSTD    DESCR1,DESCR2,N |
--------------------------------------
```

GETSTD is used to get a descriptor from the SIL stack, in case the stack is implemented differently on the particular system. Otherwise it is the same as GETDC.

Data Input to GETSTD

DESCR2	A2		
A2+N	A	F	V

Data Altered by GETSTD

DESCR1	A	F	V

==========

63. H2B (convert hexadecimal string to a bit string)

```
---------------------------------
|          H2B       SPEC1,SPEC2 |
---------------------------------
```

H2B is used to convert a hexadecimal string to a bit character string. The hexadecimal source string consists of the characters: 0123456789ABCDEFabcdef. If the hex character is not one listed, then it is considered to be zero. Each hex character converts to a four character string of zero and one characters representing the hex value. The high order bit comes first. For example if B1 is "e", then C1 is "1", C2 is "1", C3 is "1" and C4 is "0". L2*4 will always equal L1. considered to be zero bits.

Data Input to H2B

```
          ---------------------------------------------
SPEC2     |   A2   |        |        |   O2   |   L2   |
          ---------------------------------------------

          ----------------------------
A2+O2     |   B1   |  ...   |  BL2   |
          ----------------------------

          ---------------------------------------------
SPEC1     |   A1   |        |        |   O1   |   L1   |
          ---------------------------------------------
```

Data Altered by H2B

```
          ----------------------------
A1+O1     |   C1   |  ...   |  CL1   |
          ----------------------------
```

==========

64. H2I (convert hex digits to an integer)

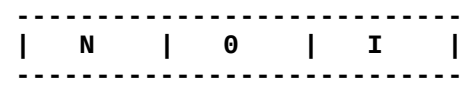
```
---------------------------------
|          H2I       DESCR,SPEC |
---------------------------------
```

H2I is used to convert a hex digit string to an integer. The hex characters can be any of 0123456789abcdefABCDEF. Other characters are treated as zero hex digits. N is formed by concatenating hex digits from B1 through BL, with BL as the low order hex digit. If L is less than ALENG*2, then zero hex digits are padded on the left in the high order positions. In all cases ALENG*2 >= L.

Data Input to H2I

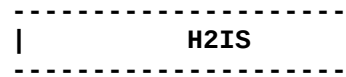
```
          ---------------------------------------------
SPEC      |   A    |        |        |   O    |   L    |
          ---------------------------------------------

          ----------------------------
A+O       |   B1   |  ...   |   BL   |
          ----------------------------
```

Data Altered by H2I



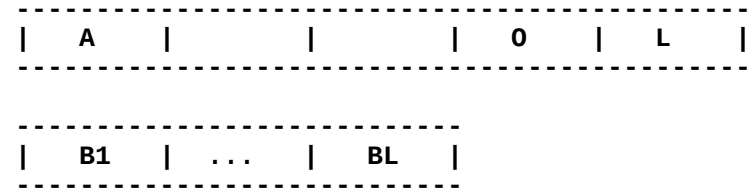
==========

65. H2IS (convert hex digits to an integer, sign extended)

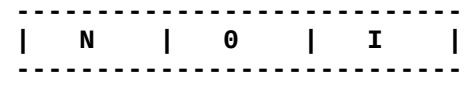


H2IS is used to convert a bit string to an integer. The hex characters can be any of 0123456789abcdefABCDEF. Other characters are treated as zero hex digits. N is formed by concatenating hex digits from B1 through BL, with BL as the low order hex digit. If L is less than ALENG*2, then the result is padded on on the left with the high order bit of the first hex digit (B1). In all cases ALENG*2 >= L. If ALENG*BITSPA is not equal to L, then the bits specified before B1 should be considered to be the same as B1, padding N in the high order positions.

Data Input to H2IS

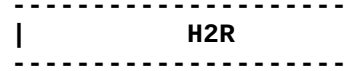


Data Altered by H2IS



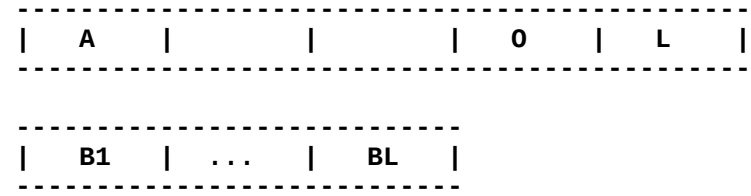
==========

66. H2R (convert hex digits to an real number)



H2R is used to convert a 16 hexadecimal digit string in double precision format to an REAL value. N is formed by concatenating hex digits from B1 through BL, with BL as the low order hex digit. In all cases ALENG*2 = L (16).

Data Input to H2R



Data Altered by H2R

DESCR	N	0	R

==========

67. H2S (convert hexadecimal string to a bit string)

H2S	SPEC1,SPEC2

H2S is used to convert a hexadecimal string to a character string. The hexadecimal source string consists of the characters: 0123456789ABCDEFabcdef. If the hex character is not one listed, then it is considered to be zero. Each pair of hex characters convert to a character in the result string. The high order hex digit comes first. (L2+1)/2 will always equal L1. If L2 is odd, a trailing zero hex digit is assumed.

Data Input to H2S

SPEC2	A2			O2	L2

A2+O2	B1	...	BL2

SPEC1	A1			O1	L1

Data Altered by H2S

A1+O1	C1	...	CL1

==========

68. HS2R (convert 8 hex digits to an real number)

HS2R	DESCR,SPEC

HS2R is used to convert a 8 hexadecimal digit string in single precision to an REAL value. N is formed by concatenating hex digits from B1 through BL, with BL as the low order hex digit. In all cases ALENG*2 = L (16).

Data Input to HS2R

SPEC	A			O	L

```
            ----------------------------
A+O         |   B1   |  ...   |   BL   |
            ----------------------------
```

Data Altered by HS2R

```
            ----------------------------
DESCR       |   N    |   0    |   R    |
            ----------------------------
```

==========

69. HX2R (convert 20 hex digits to an real number)

```
---------------------------------
|          HX2R      DESCR,SPEC |
---------------------------------
```

HX2R is used to convert a 20 hexadecimal digit string in double extended precision to an REAL value. N is formed by concatenating hex digits from B1 through BL, with BL as the low order hex digit. In all cases ALENG*2 = L (16).

Data Input to HX2R

```
            ------------------------------------------------
SPEC        |   A    |        |        |   O    |   L    |
            ------------------------------------------------

            ----------------------------
A+O         |   B1   |  ...   |   BL   |
            ----------------------------
```

Data Altered by HX2R

```
            ----------------------------
DESCR       |   N    |   0    |   R    |
            ----------------------------
```

==========

70. HEXI (convert numeric to hex digits)

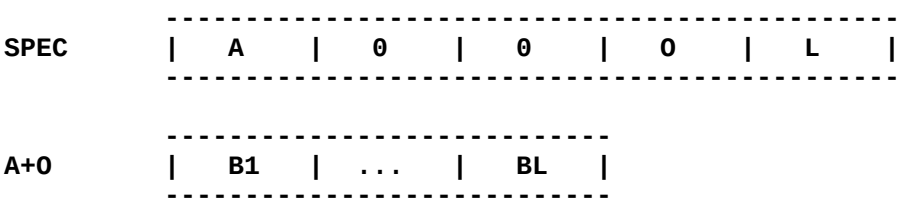
```
---------------------------------
|          HEXI      SPEC,DESCR |
---------------------------------
```

HEXI is used to expand an integer or real into its big endian representation in hexadecimal digits. L should be the number of hex digits in the NUM field of a descriptor. B1 should be the high order hex digit of NUM. BL should be the low order hex digit of NUM. NUM is either an integer or real. L is ALENG*BITSPA/4, 16 for SNOBOL5. Each of the B1...BL should be either the "0" to "F" characters.

Data Input to HEXI

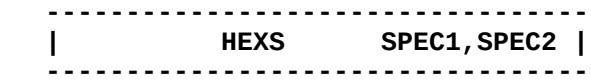
```
            ------------------------------------------------
SPEC        |   A    |        |        |   O    |   L    |
            ------------------------------------------------

            ----------------------------
DESCR       |  NUM   |        |        |
            ----------------------------
```

Data Altered by HEXI



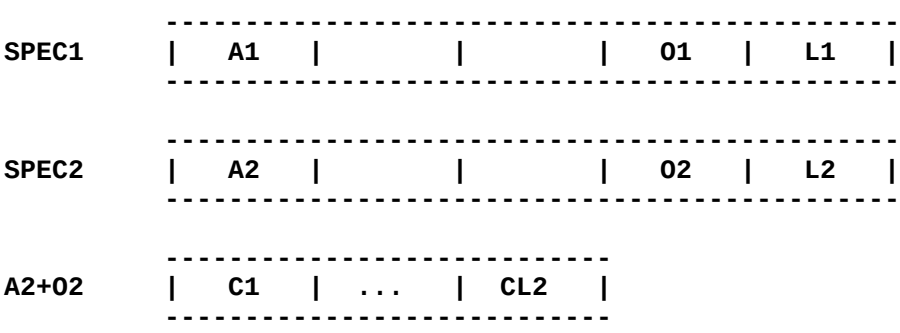
==========

71. HEXS (convert string to hex digits)

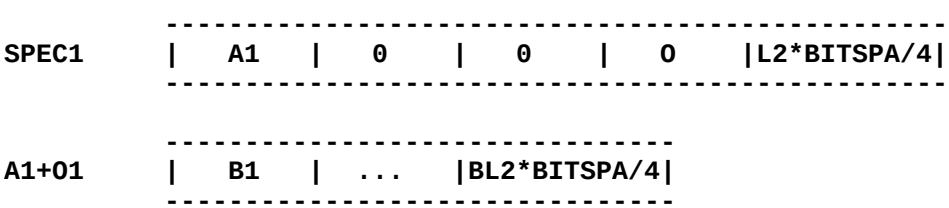


HEXS is used to expand a string into its hexadecimal digit representation. L1 should be L2*BITSPA/4. B1 should be the high order hex digit of C1. B(2) should be the low order hex digit of C1. Each of the B1...BL1 should be "0" to "F" characters. BITSPA is the number of bits per machine address of storage, usually 8. Thus a hex digits representation of a character string will take at least 2 times more storage.

Data Input to HEXS



Data Altered by HEXS

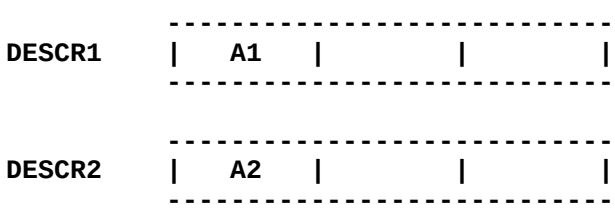


==========

72. ICOMP (integer comparison)

ICOMP is used to compare the integers in the address fields of two descriptors. The comparison is arithmetic with A1 and A2 being considered as signed integers. See ACOMP for the address comparison version. If A1 > A2, transfer is to GTLOC. If A1 = A2, transfer is to EQLOC. If A1 < A2, transfer is to LTLOC.

Data Input to ICOMP

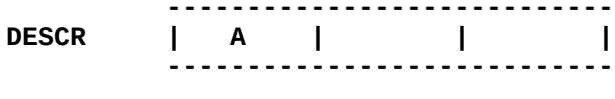


==========

73. ICOMPC (integer comparison with constant)

ICOMPC is used to compare the address field of a descriptor to a constant. The comparison is arithmetic with A being considered as a signed integer. See ACOMPC for the version for an unsigned comparizon of addresses. If A > N, transfer is to GTLOC. If A = N, transfer is to EQLOC. If A < N, transfer is to LTLOC.

Data Input to ACOMPC



Programming Notes:

1. N is never negative.

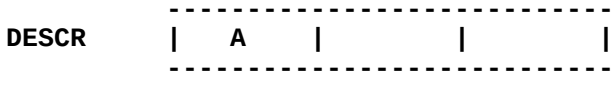
2. N is often 0.

==========

74. IEQLC (integer equal to constant test)

IEQLC is used to compare the integer in the address field of a descriptor to a constant. The comparison is arithmetic with A being considered as a signed integer. See AEQLC for the unsigned arithmetic version for addresses. If A = N, transfer is to EQLOC. Otherwise transfer is to NELOC.

Data Input to AEQLC



Programming Notes:

1. N is never negative.

2. N is often 0.

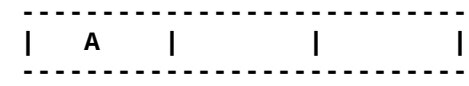
==========

75. INCRA (increment address)

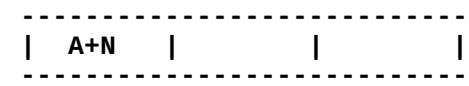
```
-----------------------------
|          INCRA     DESCR,N |
-----------------------------
```

INCRA is used to increment the unsigned address field of a descriptor. See INCRI for the signed integer version.

Data Input to INCRA



Data Altered by INCRA



Programming Notes:

1. A may be a relocatable address.

2. A is never negative.

3. N is always positive.

4. N is often 1 or D.

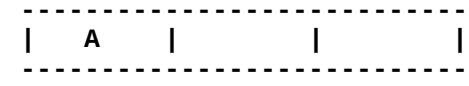
==========

76. INCRI (increment integer)

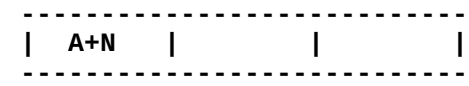
```
-----------------------------
|          INCRI     DESCR,N |
-----------------------------
```

INCRA is used to increment the signed integer in the address field of a descriptor. See INCRA for the unsigned address version.

Data Input to INCRI



Data Altered by INCRI



Programming Notes:

1. N is always positive.

2. N is often 1 or D.

==========

77. INCRV (increment value field)

```
------------------------------
|          INCRV     DESCR,N |
------------------------------
```

INCRV is used to increment the value field of a descriptor. I is considered as an unsigned (nonnegative) integer.

Data Input to INCRV

```
           ----------------------------
DESCR      |        |        |   I    |
           ----------------------------
```

Data Altered by INCRV

```
           ----------------------------
DESCR      |        |        |  I+N   |
           ----------------------------
```

Programming Notes:

1. N is always positive.

2. N is often 1.

==========

78. INCSP (increment stack pointer)

```
------------------------------
|          INCRSP    DESCR,N |
------------------------------
```

INCRSP is used to increment the unsigned address field of a descriptor which is pointing to an entry within the SIL stack. If pushing the stack increases this address, the action is the same as INCRA. However if the stack decreases this address, the action should be to decrement the address.

Data Input to INCSP

```
           ----------------------------
DESCR      |   A    |        |        |
           ----------------------------
```

Data Altered by INCSP

```
           ----------------------------
DESCR      |  A+N   |        |        |
           ----------------------------
```

Programming Notes:

1. A may be a relocatable address.

2. A is never negative.

3. N is always positive.

4. N is often 1 or D.

==========

79. INIT (initialize SNOBOL5 run)

```
----------------------
|          INIT      |
----------------------
```

INIT is used to initialize a SNOBOL5 run. INIT is the first instruction executed and is responsible for performing any initialization necessary. The operation is machine and system dependent. Typically, INIT sets program masks and the values of certain registers.

In addition to any initialization required for a particular system and machine, INIT also performs the following initialization for the SNOBOL5 system. Dynamic storage is initialized. The address fields of FRSGPT and HDSGPT are set to point to the first descriptor in dynamic storage. The address field of TLSGP1 is set to the first descriptor past the end of dynamic storage. Space for dynamic storage may be preallocated or obtained from the operating system by INIT. The timer is initialized for subsequent use by the MSTIME macro (q.v.).

==========

80. INSERT (insert node in tree)

```
-------------------------------------
|          INSERT    DESCR1,DESCR2 |
-------------------------------------
```

INSERT is used to insert a tree node above another node.

Data Input to INSERT

	A1	F1	V1
DESCR1			
DESCR2	A2	F2	V2
A1+FATHER	A3	F3	V3
A3+LSON	A4		
A2+CODE			I

Data Altered by INSERT

	A2	F2	V2
A1+FATHER			

```
                ----------------------------
     A4+RSIB    |   A2   |   F2   |   V2   |
                ----------------------------

                ----------------------------
     A2+FATHER  |   A3   |   F3   |   V3   |
                ----------------------------

                ----------------------------
     A2+LSON    |   A1   |   F1   |   V1   |
                ----------------------------

                ----------------------------
     A2+CODE    |        |        |  I+1   |
                ----------------------------
```

Programming Notes:

1. Since the fields of the descriptor at A1+FATHER are used in the data to be altered, care should be taken not to modify this descriptor until its former values have been used.

2. INSERT is only used by compilation procedures.

3. FATHER, LSON, RSIB, and CODE are symbols defined in the source program.

==========

81. INTRL (convert integer to real number)

```
-------------------------------------
|          INTRL     DESCR1,DESCR2 |
-------------------------------------
```

INTRL is used to convert a (signed) integer to a real number. R(I) is the real number corresponding to I.

Data Input to INTRL

```
                ----------------------------
     DESCR2     |   I    |        |        |
                ----------------------------
```

Data Altered by INTRL

```
                ----------------------------
     DESCR1     |  R(I)  |   0    |   R    |
                ----------------------------
```

Programming Notes:

1. R is a symbol defined in the source program and is the code for the real data type.

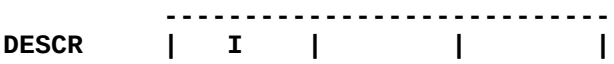
==========

82. INTSPC (convert integer to specifier)

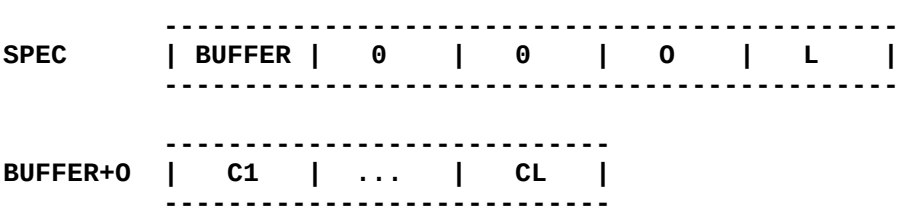
```
---------------------------------
|          INTSPC    SPEC,DESCR |
---------------------------------
```

INTSPC is used to convert a (signed) integer to a specified string.

Data Input to INTSPC



Data Altered by INTSPC



Programming Notes:

1. C1...CL should be a "normalized" string corresponding to the integer A. That is, it should contain no leading zeroes and should begin with a minus sign if A is negative.

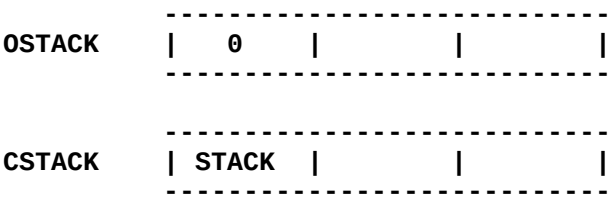
2. BUFFER is local to INTSPC and its contents may be overwritten by a subsequent use of INTSPC.

==========

83. ISTACK (initialize stack)

ISTACK is used to initialize the system stack. Note: this is assuming the stack grows to upward addresses. SNOBOL5 uses the native Intel stack which grows to downward addresses. The various stack macros need to be adjusted to handle this.

Data Altered by ISTACK

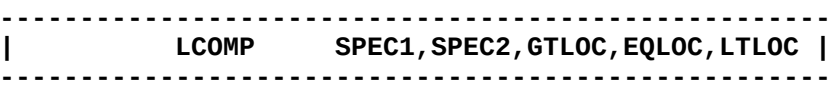


Programming Notes:

1. STACK is a program symbol whose value is the address of the first descriptor of the system stack.

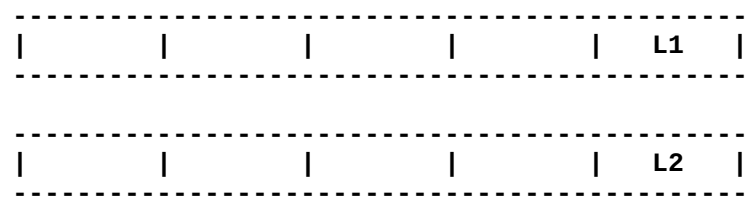
==========

84. LCOMP (length comparison)



LCOMP is used to compare the lengths of two specifiers. If L1 > L2, transfer is to GTLOC. If L1 = L2, transfer is to EQLOC. If L1 < L2, transfer is to LTLOC.

Data Input to LCOMP



==========

85. LEQLC (length equal to constant test)

```
----------------------------------------
|          LEQLC     SPEC,N,NELOC,EQLOC |
----------------------------------------
```

LEQLC is used to compare the length of a specifier to a constant. If L = N, transfer is to EQLOC. Otherwise transfer is to NELOC.

Data Input to LEQLC

Programming Notes:

1. L and N are never negative.

==========

86. LEXCMP (lexical comparison of strings)

```
-----------------------------------------------------
|          LEXCMP    SPEC1,SPEC2,GTLOC,EQLOC,LTLOC |
-----------------------------------------------------
```

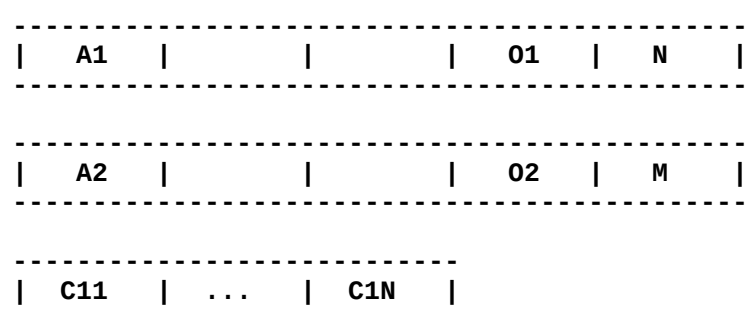
LEXCMP is used to compare two strings lexicographically (i.e. according to their alphabetical ordering).

If C11...C1N1 > C21...C2M, transfer is to GTLOC.

If C11...C1N1 = C21...C2M, transfer is to EQLOC.

If C11...C1N1 < C21...C2M, transfer is to LTLOC.

Data Input to LEXCMP



```
              ---------------------------
     A2+O2    |  C21   |  ...   |  C2M   |
              ---------------------------
```

Programming Notes:

1. The lexicographical ordering is machine dependent and is determined by the numerical order of the internal representation of the characters for a particular machine.

2. A string that is an initial substring of another string is lexicographically less than that string. That is ABC is less than ABCA.

3. The null (zero-length) string is lexicographically less than any other string.

4. Two strings are equal if and only if they are of the same length and are identical character by character.

5. By far the most frequent use of LEXCMP is to determine whether two strings are the same or different. In these cases GTLOC and LTLOC will specify the same location or both be omitted. Because of the frequency of such use, it is desirable to handle this case specially, since a test for equality usually can be performed more efficiently than the general test.

==========

87. LHERE (define location here)

```
-----------------------
| LOC      LHERE      |
-----------------------
```

LHERE is used to establish the equivalence of LOC as the location of the next program instruction. It only applies to instructions and not data declarations in contrast to DHERE.

Programming Notes:

1. LHERE is equivalent to the familiar EQU *. Similarly

```
      LOC     LHERE
              OP
```

is equivalent to

```
      LOC     OP
```

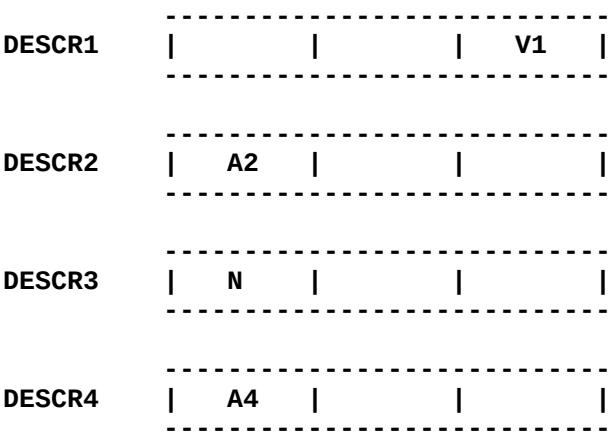
==========

88. LINK (link to external function)

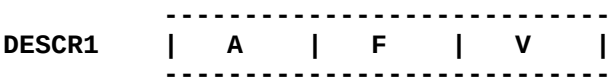
```
-------------------------------------------------------------
|          LINK      DESCR1,DESCR2,DESCR3,DESCR4,FLOC,SLOC |
-------------------------------------------------------------
```

LINK is used to link to an external function. A2 is a pointer to an argument list of N descriptors. A4 is the address of the external function to be called. V1 is the date type expected for the resulting value. The returned value is placed in DESCR1. If the external function signals failure, transfer is to FLOC. Otherwise transfer is to SLOC. SNOBOL5 does not implement this.

Data Input to LINK



Data Altered by LINK



Programming Notes:

1. LINK is a system-dependent operation.

2. LINK need not be implemented if LOAD is not. In this case, LINK should branch to INTR10.

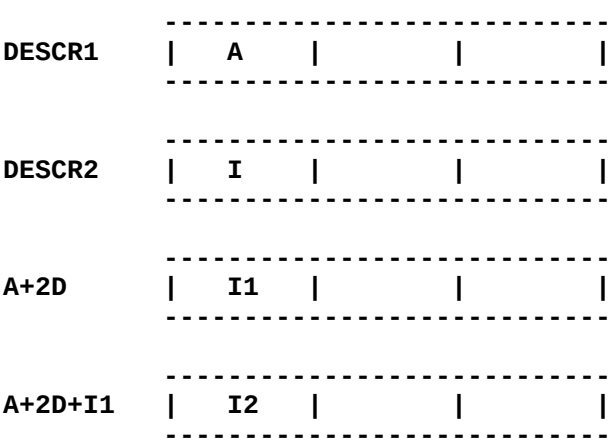
3. LINK is not implemented in Oregon SNOBOL5.

==========

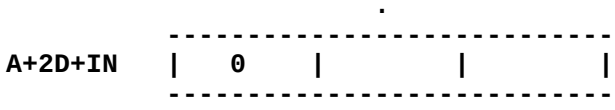
89. LINKOR (link 'or' fields of pattern nodes)

LINKOR links through "or" (alternative) fields of pattern nodes until the end, indicated by a zero field, is reached. This zero field is replaced by I.

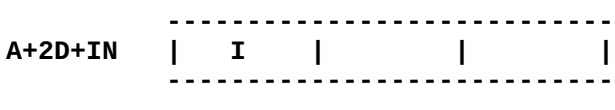
Data Input to LINKOR



.
.



Data Altered by LINKOR



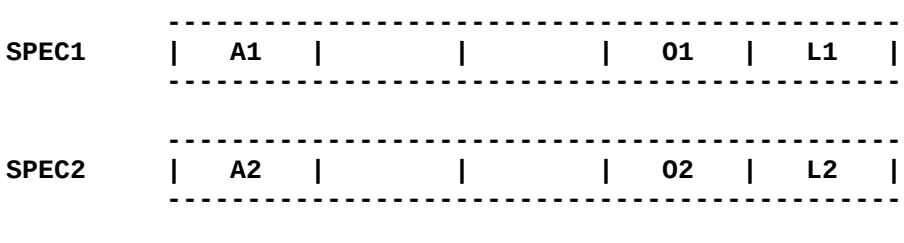
==========

90. LOAD (load external function)

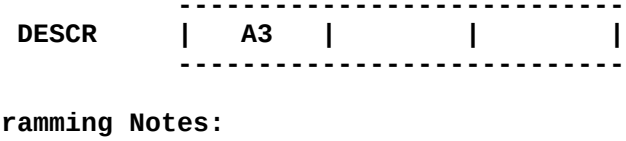
```
-------------------------------------------------
|          LOAD      DESCR,SPEC1,SPEC2,FLOC,SLOC |
-------------------------------------------------
```

LOAD is used to load an external function. C11...C1L1 is the name of the external function to be loaded from a library. C21...C2L2 is the name of the library. A3 is the address of the entry point. If the external function is loaded, transfer is to SLOC. Otherwise transfer is to FLOC. SNOBOL5 does not implement this.

Data Input to LOAD



Data Altered by LOAD



Programming Notes:

1. LOAD is a system-dependent operation.

2. LOAD need not be implemented as such. If it is not, the built-in function LOAD will not be available, and an error comment should be generated by branching to UNDF.

3. On the IBM System/360, LOAD uses the OS macro LOAD to bring an external function from the library whose DDNAME is specified by C21...C2L2.

4. LOAD is not implemented in Oregon SNOBOL5.

==========

91. LOBFUN (show low order bits as characters 0 and 1)

```
---------------------------------
|          LOBFUN    SPEC1,SPEC2 |
---------------------------------
```

LOBFUN is used to make a bit string visually more easily viewable. The low order bits of the characters C21...C2L are examined and those characters are changed to either "0" or "1". F(Cn) = "0" if the low order bit of C2n is zero. F(Cn) = "1" if the low order bit of C2n is one.

Data Input to NOTFUN

```
          ---------------------------------------------
SPEC1     |   A1   |        |        |   O1   |   L    |
          ---------------------------------------------

          ---------------------------------------------
SPEC2     |   A2   |        |        |   O2   |   L    |
          ---------------------------------------------

          ----------------------------
A2+O2     |  C21   |  ...   |  C2L   |
          ----------------------------
```

Data Altered by NOTFUN

```
          ----------------------------
A1+O1     | F(C1)  |  ...   | F(CL)  |
          ----------------------------
```

Programming Notes:

1. L may be zero.

==========

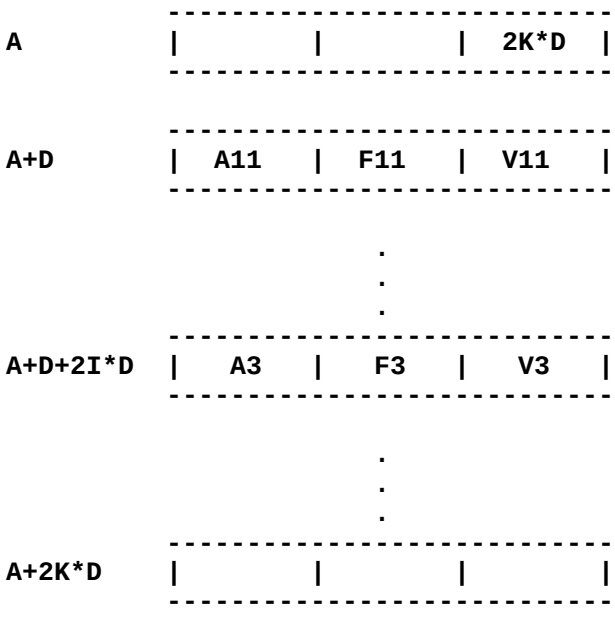
92. LOCAPT (locate attribute pair by type)

```
-----------------------------------------------------
|          LOCAPT    DESCR1,DESCR2,DESCR3,FLOC,SLOC |
-----------------------------------------------------
```

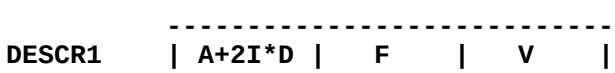
LOCAPT is used to locate the "type" descriptor of a descriptor pair on an attribute list. Descriptors on an attribute list are in "type-value" pairs. Odd-numbered descriptors are "type" descriptors. The list starting at A+D is searched, comparing descriptors at A+D, A+3D, ... for the first descriptor which is equal to DESCR3. If a descriptor equal to DESCR3 is not found, transfer is to FLOC. Otherwise transfer is to SLOC.

Data Input to LOCAPT

```
          ----------------------------
DESCR2    |   A    |   F    |   V    |
          ----------------------------

          ----------------------------
DESCR3    |   A3   |   F3   |   V3   |
          ----------------------------
```



Data Altered by LOCAPT



Programming Notes:

1. Note that the address of DESCR1 is set to one descriptor less then the descriptor that is located.

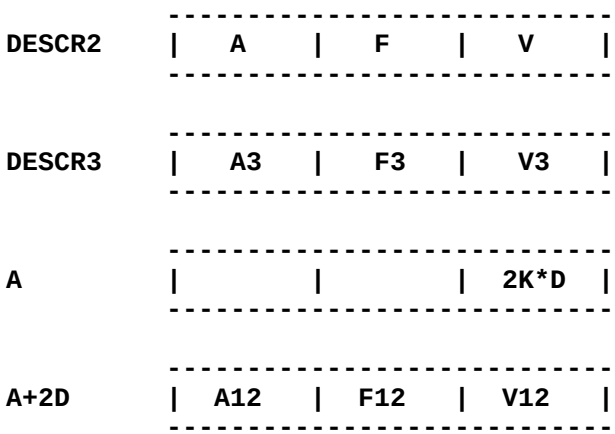
==========

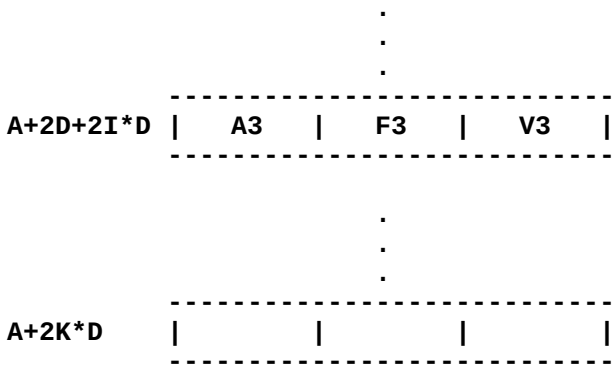
93. LOCAPV (locate attribute pair by value)

```
-------------------------------------------------------
|          LOCAPV    DESCR1,DESCR2,DESCR3,FLOC,SLOC |
-------------------------------------------------------
```

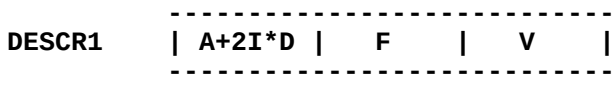
LOCAPV is used to locate the "value" descriptor of a descriptor pair on an attribute list. Descriptors on an attribute list are in "type-value" pairs. Even-numbered descriptors are "value" descriptors. The list starting at A+D is searched, comparing descriptors at A+2D, A+4D, ... for the first descriptor which is equal to DESCR3. If a descriptor equal to DESCR3 is not found, transfer is to FLOC. Otherwise transfer is to SLOC.

Data Input to LOCAPV





Data Altered by LOCAPV



Programming Notes:

1. Note that the address of DESCR1 is set to two descriptors less than the descriptor that is located.

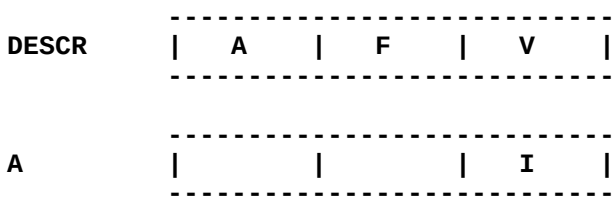
2. The S/370 implementation sets F and V of DESCR1 to zero.

==========

94. LOCSP (locate specifier to string)

LOCSP is used to obtain a specifier to a string given in a string structure. CPD is the number of characters per descriptor.

Data Input to LOCSP



Data Altered by LOCSP if A is not equal to zero (null pointer)

Data Altered by LOCSP if A is equal to zero

Programming Notes:

1. If A = zero, the value of DESCR represents the null (zero-length) string and is handled as a special case as indicated. The other fields of SPEC are unchanged in this case.

==========

95. LOG (natural logarithm)

```
-----------------------------------
|          LOG       DESCR1,DESCR2 |
-----------------------------------
```

LOG is used to compute the natural logarithm of A.

Data Input to LOG

```
           ----------------------------
DESCR2     |   A    |        |   R    |
           ----------------------------
```

Data Altered by LOG

```
           ----------------------------
DESCR1     | LOG(A) |   0    |   R    |
           ----------------------------
```

==========

96. LOG2 (base 2 logarithm)

```
-----------------------------------
|          LOG2      DESCR1,DESCR2 |
-----------------------------------
```

LOG2 is used to compute the base 2 logarithm of A.

Data Input to LOG2

```
           ----------------------------
DESCR2     |   A    |        |   R    |
           ----------------------------
```

Data Altered by LOG2

```
           ----------------------------
DESCR1     |LOG2(A) |   0    |   R    |
           ----------------------------
```

==========

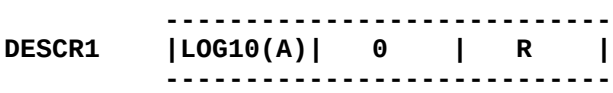
97. LOG10 (base 10 logarithm)

```
-----------------------------------
|          LOG10     DESCR1,DESCR2 |
-----------------------------------
```

LOG10 is used to compute the base 10 logarithm of A.

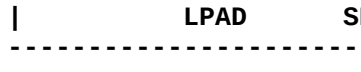
Data Input to LOG10

Data Altered by LOG10



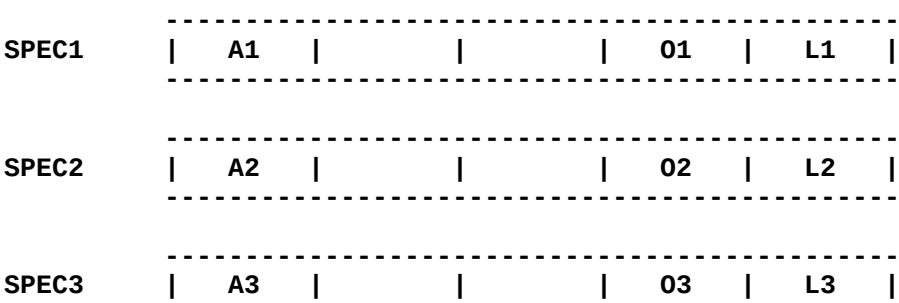
==========

98. LPAD (pad a string on the left with blanks or other character)

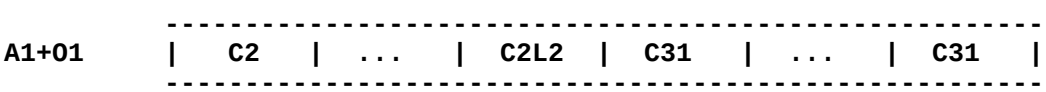


LPAD is used to pad the string specified by SPEC2 on the left side with the first character of the string specified by SPEC3. If L3 is zero or if SPEC3 was omitted, then a blank character is used for padding. If L1 <= L2, the the string is simply copied without padding.

Data Input to LPAD



Data Altered by LPAD



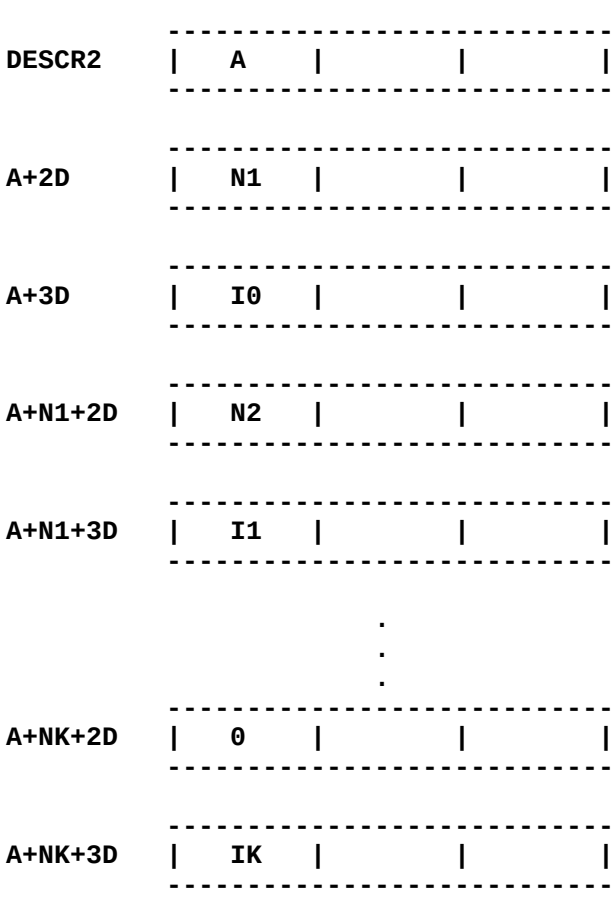
==========

99. LVALUE (get least length value)

LVALUE is used to get the least value of address fields in a chain of pattern nodes. The address field of DESCR1 is set to I where

$$I = \min(I0,\ldots,IK)$$

Data Input to LVALUE



Data Altered by LVALUE

DESCR1	I	0	0

Programming Notes:

1. I0,...,IK are all nonnegative.

2. A is never zero, but N1 may be.

---

100. MAKNOD (make pattern node)

```
-------------------------------------------------------------------
|          MAKNOD    DESCR1,DESCR2,DESCR3,DESCR4,DESCR5,DESCR6 |
-------------------------------------------------------------------
```

MAKNOD is used to make a node for a pattern. DESCR6 may be omitted. If it is, one less descriptor is modified, but the two forms are otherwise the same.

Data Input to MAKNOD

DESCR2	A2	F2	V2
DESCR3	A3		
DESCR4	A4		
DESCR5	A5	F5	V5

Additional Data Input if DESCR6 is Given

DESCR6	A6	F6	V6

Data Altered by MAKNOD

DESCR1	A2	F2	V2
A2+D	A5	F5	V5
A2+2D	A4		
A2+3D	A3		

Additional Data Altered if DESCR6 is Given

A2+4D	A6	F6	V6

Programming Notes:

1. As indicated, there are two forms of MAKNOD. If DESCR6 is given, an additional descriptor if modified, but otherwise the two forms are the same.

2. DESCR1 must be changed last, since DESCR6 may be the same descriptor as DESCR1.

3. MAKNOD is used only for constructing patterns.

==========

101. MNREAL (minus real number)

```
-----------------------------------
|          MNREAL    DESCR1,DESCR2 |
-----------------------------------
```

MNREAL is used to change the sign of a real number.

Data Input to MNREAL

	R	F	V
DESCR2			

Data Altered by MNREAL

	-R	F	V
DESCR1			

Programming Notes:

1. R may be negative.

==========

102. MNSINT (minus integer)

```
-----------------------------------------------
|          MNSINT    DESCR1,DESCR2,FLOC,SLOC |
-----------------------------------------------
```

MNSINT is used to change the sign of an integer. If -I exceeds the maximum integer, transfer is to FLOC. Otherwise transfer is to SLOC. Note that with two's complement integers, the magnitude of the maximum negative number can be one more than that for the positive. When the maximum negative number is inverted, it may cause a branch to FLOC, depending on the implementation.

Data Input to MNSINT

	I	F	V
DESCR2			

Data Altered by MNSINT

	-I	F	V
DESCR1			

Programming Notes:

1. I may be negative.

==========

103. MOVA (move address)

```
-----------------------------------
|          MOVA      DESCR1,DESCR2 |
-----------------------------------
```

MOVA is used to move an address field from one descriptor to another.

Data Input to MOVA

```
           ----------------------------
DESCR2     |   A    |        |        |
           ----------------------------
```

Data Altered by MOVA

```
           ----------------------------
DESCR1     |   A    |        |        |
           ----------------------------
```

==========

104. MOVBLK (move block of descriptors)

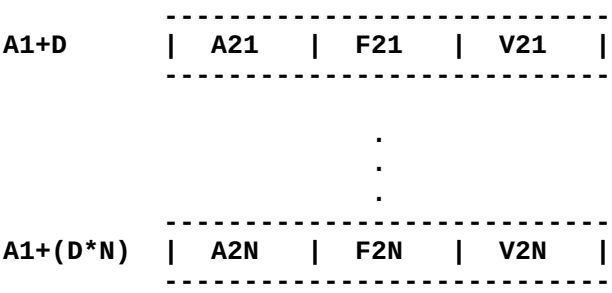
```
-------------------------------------------
|          MOVBLK    DESCR1,DESCR2,DESCR3 |
-------------------------------------------
```

MOVBLK is used to move (copy) a block of descriptors.

Data Input to MOVBLK

```
           ----------------------------
DESCR1     |   A1   |        |        |
           ----------------------------

           ----------------------------
DESCR2     |   A2   |        |        |
           ----------------------------

           ----------------------------
DESCR3     |  D*N   |        |        |
           ----------------------------

           ----------------------------
A2+D       |  A21   |  F21   |  V21   |
           ----------------------------

                       .
                       .
                       .
           ----------------------------
A2+(D*N)   |  A2N   |  F2N   |  V2N   |
           ----------------------------
```

Data Altered by MOVBLK



Programming Notes:

1. Note that the descriptor at A1 is not altered.

2. The area into which the move is made may overlap the area from which the move is made. This only occurs when A1 is less than A2. Care must be taken to handle this case correctly.

==========

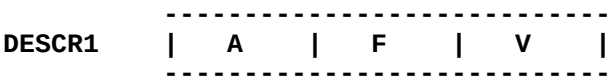
105. MOVD (move descriptor)

```
-----------------------------------
|          MOVD      DESCR1,DESCR2 |
-----------------------------------
```

MOVD is used to move (copy) a descriptor from one location to another.

Data Input to MOVD

Data Altered by MOVD



==========

106. MOVDIC (move descriptor indirect with constant offset)

```
-----------------------------------------
|          MOVDIC    DESCR1,N1,DESCR2,N2 |
-----------------------------------------
```

MOVDIC is used to move a descriptor that is indirectly specified with an offset constant.

Data Input to MOVDIC

```
            ----------------------------
DESCR2      |   A2   |        |        |
            ----------------------------

            ----------------------------
A2+N2       |   A    |   F    |   V    |
            ----------------------------
```

Data Altered by MOVDIC

```
            ----------------------------
A1+N1       |   A    |   F    |   V    |
            ----------------------------
```

==========

107. MOVSTD (move a block of descriptors from the SIL stack)

```
-----------------------------------------
|          MOVSTD    DESCR1,DESCR2,DESCR3 |
-----------------------------------------
```

MOVSTD is used to move (copy) a block of descriptors from the SIL stack. If the SIL stack address increases with a push, then this macro can be the same as MOVBLK. However, if the stack address decreases with a push, this macro must be implemented to account for that.

Data Input to MOVSTD

```
            ----------------------------
DESCR1      |   A1   |        |        |
            ----------------------------

            ----------------------------
DESCR2      |   A2   |        |        |
            ----------------------------

            ----------------------------
DESCR3      |  D*N   |        |        |
            ----------------------------

            ----------------------------
A2          |  A21   |  F21   |  V21   |
            ----------------------------

                        .
                        .
                        .
            ----------------------------
A2-(D*(N-1))|  A2N   |  F2N   |  V2N   |
            ----------------------------
```

Data Altered by MOVSTD

```
            ----------------------------
A1+D        |  A21   |  F21   |  V21   |
            ----------------------------

                        .
                        .
                        .
```

```
              ----------------------------
     A1+(D*N) |  A2N   |  F2N   |  V2N   |
              ----------------------------
```

Programming Notes:

1. Note that the descriptor at A1 is not altered.

2. The area into which the move is made may overlap the area from which the move is made. This only occurs when A1 is less than A2. Care must be taken to handle this case correctly.

==========

108. MOVV (move value field)

```
-----------------------------------
|          MOVV      DESCR1,DESCR2 |
-----------------------------------
```

MOVV is used to move a value field from one descriptor to another.

Data Input to MOVV

```
              ----------------------------
     DESCR2   |        |        |   V    |
              ----------------------------
```

Data Altered by MOVV

```
              ----------------------------
     DESCR1   |        |        |   V    |
              ----------------------------
```

==========

109. MOVV0 (move value field and set flag field to zero)

```
-----------------------------------
|          MOVV0     DESCR1,DESCR2 |
-----------------------------------
```

MOVV0 is used to move a value field from one descriptor to another and set the destination flag field to zero.

Data Input to MOVV0

```
              ----------------------------
     DESCR2   |        |        |   V    |
              ----------------------------
```

Data Altered by MOVV0

```
              ----------------------------
     DESCR1   |        |   0    |   V    |
              ----------------------------
```

==========

110. MPREAL (multiply real numbers)

```
-------------------------------------------------
|          MPREAL    DESCR1,DESCR2,DESCR3,FLOC,SLOC |
-------------------------------------------------
```

MPREAL is used to multiply two real numbers. If the result is out of the range available for real numbers, transfer is to FLOC. Otherwise transfer is to SLOC.

Data Input to MPREAL

DESCR2	R2	F2	V2
DESCR3	R3		

Data Altered by MPREAL

DESCR1	R2*R3	F2	V2

==========

111. MSTIME (get millisecond time)

```
----------------------------
|          MSTIME    DESCR |
----------------------------
```

MSTIME is used to get the millisecond time.

Data Altered by MSTIME

DESCR	TIME	0	0

Programming Notes:

1. The origin with respect to which the time is obtained is not important. The SNOBOL5 system deals only with differences in times.

2. In IBM System/360 the units were milliseconds, but accuracy is not critical. SNOBOL5 uses units of nanoseconds instead of milliseconds and the accuracy is not guaranteed.

3. MSTIME is used in program tracing, the SNOBOL5 TIME function, and in statistics printed upon termination of a SNOBOL5 run.

4. It is not critically important that MSTIME be implemented as such. If it is not, the address field of DESCR should be set to zero also.

==========

112. MULT (multiply integers)

```
-----------------------------------------------------
|          MULT      DESCR1,DESCR2,DESCR3,FLOC,SLOC |
-----------------------------------------------------
```

MULT is used to multiply two integers. In the event of overflow, transfer is to FLOC. Otherwise, transfer is to SLOC.

Data Input to MULT

```
           ----------------------------
DESCR2     |   I2   |   F2   |   V2   |
           ----------------------------

           ----------------------------
DESCR3     |   I3   |        |        |
           ----------------------------
```

Data Altered by MULT

```
           ----------------------------
DESCR1     | I2*I3  |   F2   |   V2   |
           ----------------------------
```

Programming Notes:

1. The test for success and failure is used in only two calls of this macro. Hence the code to make the check is not needed in most cases.

2. DESCR1 and DESCR2 are often the same.

==========

113. MULTA (multiply an address)

```
-----------------------------------------
|          MULTA     DESCR1,DESCR2,DESCR3 |
-----------------------------------------
```

MULT is used to multiply an address by an integer.

Data Input to MULTA

```
           ----------------------------
DESCR2     |   A2   |        |        |
           ----------------------------

           ----------------------------
DESCR3     |   I3   |        |        |
           ----------------------------
```

Data Altered by MULTA

```
           ----------------------------
DESCR1     | A2*I3  |        |        |
           ----------------------------
```

==========

114. MULTC (multiply address by constant)

```
-------------------------------------
|          MULTC     DESCR1,DESCR2,N |
-------------------------------------
```

MULTC is used to multiply an integer by a constant.

Data Input to MULTC

```
          ----------------------------
DESCR2    |   I    |        |        |
          ----------------------------
```

Data Altered by MULTC

```
          ----------------------------
DESCR1    |  I*N   |   0    |   0    |
          ----------------------------
```

Programming Notes:

1. I*N never exceeds the range available for integers.

2. DESCR1 and DESCR2 are often the same.

3. N is often D, which typically may be implemented by a shift, or simply by no operation if D is 1 for a particular machine.

==========

115. NANDFUN (perform a logical nand function)

```
----------------------------------------
|          NANDFUN   SPEC1,SPEC2,SPEC3 |
----------------------------------------
```

NANDFUN is used to logically NAND the bytes specified by strings SPEC2 and SPEC3 and place the result in SPEC1. If either SPEC2 or SPEC3 is shorter than the other, then the remaining characters of the shorter string are assumed to be zero bytes. F(Cn) = logical NAND of C2n and C3n.

Data Input to NANDFUN

```
          ------------------------------------------------
SPEC1     |   A1   |        |        |   O1   |MAX(L2,L3)|
          ------------------------------------------------

          ---------------------------------------------
SPEC2     |   A2   |        |        |   O2   |   L2   |
          ---------------------------------------------

          ---------------------------------------------
SPEC3     |   A3   |        |        |   O3   |   L3   |
          ---------------------------------------------

          ----------------------------
A2+O2     |  C21   |  ...   |  C2L2  |
          ----------------------------
```

```
          ----------------------------
A3+O3     |  C31   |  ...   |  C3L2  |
          ----------------------------
```

Data Altered by NANDFUN

```
          -------------------------------------
A1+O1     | F(C1)  |  ...   |F(C(MAX(L2,L3)))|
          -------------------------------------
```

Programming Notes:

1. L2 and L3 may be zero.

==========

116. NORFUN (perform a logical nor function)

```
----------------------------------------
|          NORFUN    SPEC1,SPEC2,SPEC3 |
----------------------------------------
```

NORFUN is used to logically NOR the bytes specified by strings SPEC2 and SPEC3 and place the result in SPEC1. If either SPEC2 or SPEC3 is shorter than the other, then the remaining characters of the shorter string are assumed to be zero bytes. F(Cn) = logical NOR of C2n and C3n.

Data Input to NORFUN

```
          ------------------------------------------------
SPEC1     |   A1   |        |        |   O1   |MAX(L2,L3)|
          ------------------------------------------------

          ---------------------------------------------
SPEC2     |   A2   |        |        |   O2   |   L2   |
          ---------------------------------------------

          ---------------------------------------------
SPEC3     |   A3   |        |        |   O3   |   L3   |
          ---------------------------------------------

          ----------------------------
A2+O2     |  C21   |  ...   |  C2L2  |
          ----------------------------

          ----------------------------
A3+O3     |  C31   |  ...   |  C3L2  |
          ----------------------------
```

Data Altered by NORFUN

```
          -------------------------------------
A1+O1     | F(C1)  |  ...   |F(C(MAX(L2,L3)))|
          -------------------------------------
```

Programming Notes:

1. L2 and L3 may be zero.

==========

117. NOTFUN (perform a logical negation function)

```
---------------------------------
|          NOTFUN    SPEC1,SPEC2 |
---------------------------------
```

NOTFUN is used to logically invert (negate) the bytes specified by strings SPEC2 and place the result in SPEC1. F(Cn) = logical NOT of C2n.

Data Input to NOTFUN

```
          ------------------------------------------------
SPEC1     |   A1   |        |        |   O1   |   L    |
          ------------------------------------------------

          ------------------------------------------------
SPEC2     |   A2   |        |        |   O2   |   L    |
          ------------------------------------------------

          ----------------------------
A2+O2     |  C21   |  ...   |  C2L   |
          ----------------------------
```

Data Altered by NOTFUN

```
          ----------------------------
A1+O1     | F(C1)  |  ...   | F(CL)  |
          ----------------------------
```

Programming Notes:

1. L may be zero.

==========

118. ORDVST (order variable storage)

```
-----------------------
|          ORDVST     |
-----------------------
```

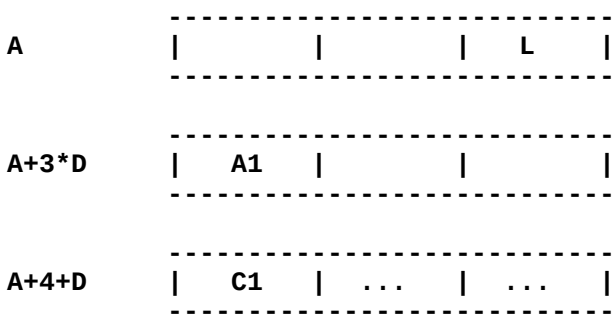
ORDVST is used to alphabetically order variables in SNOBOL5 dynamic storage. Variables are organized in a number of bins, each bin containing a linked list of variables as shown below. OBEND = OBSTRT+(OBSIZ-1)*D, where OBSIZ is the number of bins and is defined in the source program.

Bins of Variables

```
          ----------------------------
OBSTRT    |   A1   |        |        |
          ----------------------------

          ----------------------------
OBSTRT+D  |   A2   |        |        |
          ----------------------------

                      .
                      .
                      .
```

The addresses A1, A2, ..., AN point to the first variable in each bin. A zero value for any of these addresses indicates there are no variables in that bin. Within each bin, variables are linked together.

Relevant Parts of a Variable



L is the length of the string. The string itself begins at A+4*D and occupies as many descriptor locations as are necessary. A1 is a link to the next variable in the bin. A zero value of A1 indicates the end of the chain for that bin.
Programming Notes:

1. ORDVST is used only in ordering variables for a programmer-requested postmortem dump of variable storage. ORDVST need not be implemented as such, but may simply perform no operation. In this case, the postmortem dump will not be alphabetized, but will be otherwise correct.

2. If ORDVST is implemented, it is easiest to put all variables in one long chain starting at OBSTRT. The address fields of the descriptors OBSTRT+D,...,OBSTRT+(OBSIZ-1)*D should then be set to zero.

3. Since dynamic storage may contain many variables, some care must be taken to assure that the sorting procedure is not excessively slow. Variables whose values are the null string (zero address field and value field containing the program symbol S) should be omitted from the sort.

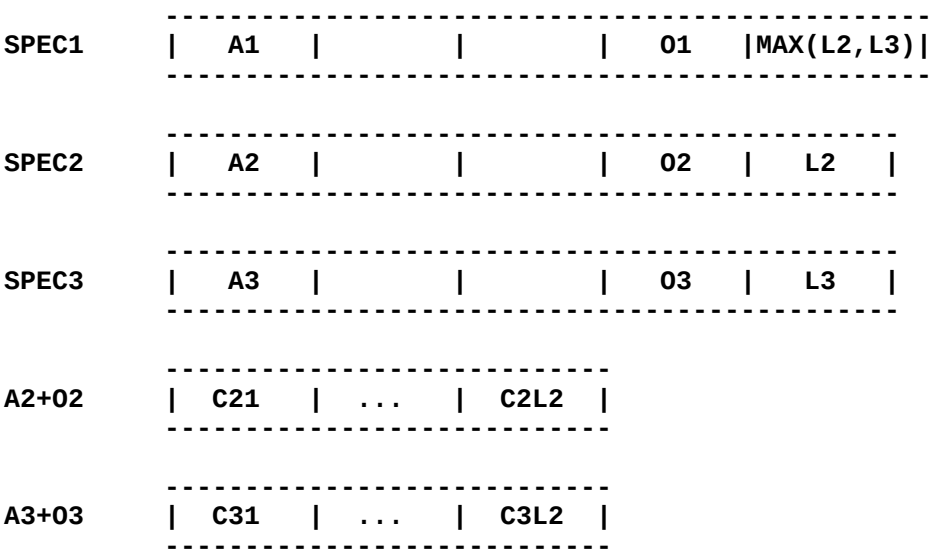
4. Since any character may appear in a string, the value of L must be used to determine the length of the string in a variable -- characters following the string in the last descriptor are undefined.

==========

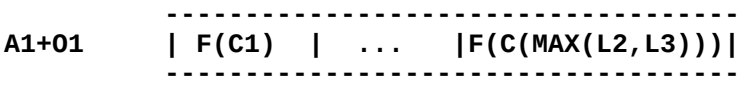
119. ORFUN (perform a logical OR function)

ORFUN is used to logically OR the bytes specified by strings SPEC2 and SPEC3 and place the result in SPEC1. If either SPEC2 or SPEC3 is shorter than the other, then the remaining characters of the shorter string are assumed to be zero bytes. F(Cn) = logical OR of C2n and C3n.

Data Input to ORFUN



Data Altered by ORFUN

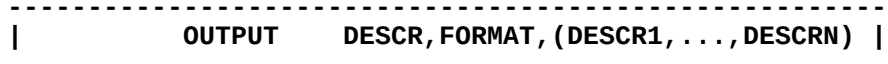


Programming Notes:

1. L2 OR L3 may be zero.

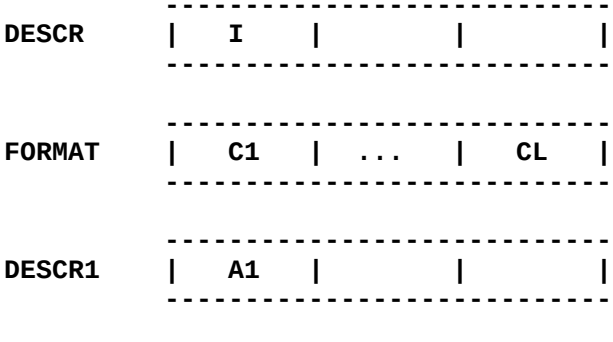
==========

120. OUTPUT (output record)



OUTPUT is used to output a list of items according to FORMAT. The output is put on the file associated with unit reference number I. The format C1...CL may specify literals and the conversion of integers and real numbers given in the address fields A1,...,AN. OUTPUT is used for statistics and informational messages.

Data Input to OUTPUT



.
.
.

DESCRN	AN		

Programming Notes:

1. The FORMAT field is a template in Oregon SNOBOL5. See the SIL source FORMAT statements for examples.

==========

121. PLUGTB (plug syntax table)

PLUGTB TABLE,KEY,SPEC

PLUGTB is used to set selected indicator fields in the entries of a syntax table to a constant. KEY may be one of four values:

CONTIN
ERROR
STOP
STOPSH

The indicator fields of entries corresponding to C1,...,CL are set to T where T is the indicator that corresponds to the value of KEY.

Data Input to PLUGTB

SPEC	A			O	L

A+O	C1	...	CL

Data Altered by PLUGTB for ERROR, STOP, or STOPSH

TABLE+E*C1		T	
	.		
	.		
	.		
TABLE+E*CL		T	

Data Altered by PLUGTB for CONTIN

TABLE+E*C1	TABLE	0	
	.		
	.		
	.		
TABLE+E*CL	TABLE	0	

Programming Notes:

1. See Section 4.2.

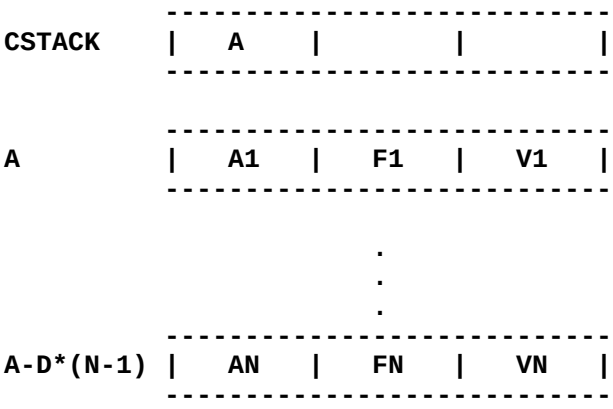
==========

122. POP (pop descriptors from stack)

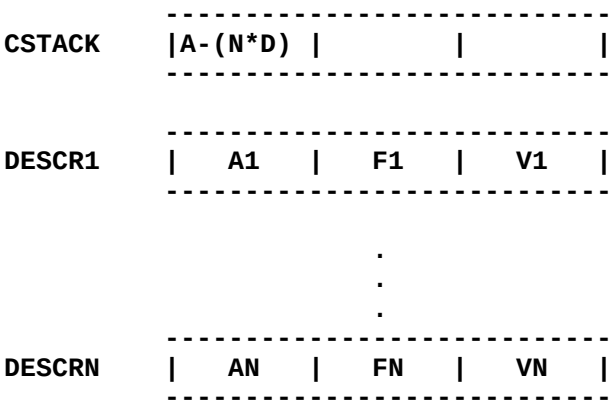
```
-----------------------------------------
|          POP       (DESCR1,...,DESCRN) |
-----------------------------------------
```

POP is used to pop a list of descriptors off the system stack.

Data Input to POP



Data Altered by POP



Programming Notes:

1. If A-(N*D) < STACK, stack underflow occurs. This condition indicates a programming error in the implementation of the macro language. An appropriate diagnostic message indicating an error may be obtained by transferring to the program location INTR10 if the condition is detected.

1. N is always less than 9.

==========

123. PROC (procedure entry)

```
---------------------------
| LOC1     PROC      LOC2 |
---------------------------
```

PROC is used to identify a procedure entry point. LOC2 may be omitted, in which case LOC1 is the primary procedure entry point. If LOC2 is given, LOC1 is a secondary entry point in the procedure with primary entry point LOC2.

Programming Notes:

1. Procedure entry points are referred to by RCALL, BRANIC, and BRANCH (in its two-argument form).

2. In most implementations, PROC has no functional use and may be implemented as LHERE. For machines that have a severely limited program basing range (such as the IBM System/360), PROC may be used to perform required basing operations.

==========

124. PSTACK (post stack position)

```
----------------------------
|          PSTACK    DESCR |
----------------------------
```

PSTACK is used to post the current stack position.

Data Input to PSTACK

CSTACK	A		

Data Altered by PSTACK

DESCR	A-D	0	0

Programming Notes:

1. The A field of DESCR is set to A instead of A-D for a download growing stack as in SNOBOL5.

==========

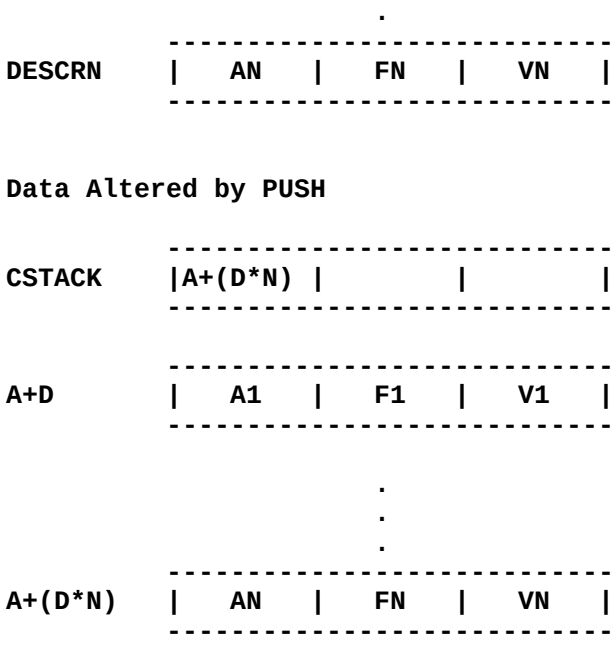
125. PUSH (push descriptors onto stack)

```
-----------------------------------------
|          PUSH      (DESCR1,...,DESCRN) |
-----------------------------------------
```

PUSH is used to push a list of descriptors onto the system stack. The items in the list are processed left to right.

Data Input to PUSH

CSTACK	A		
DESCR1	A1	F1	V1
.			
.			



Programming Notes:

1. If A+(D*N) > STACK+(STSIZE*D), stack overflow occurs. Transfer should be made to the program location OVER, which will result in an appropriate error termination. With a download growing stack this test must be adjusted accordingly.

2. N is always less than 9.

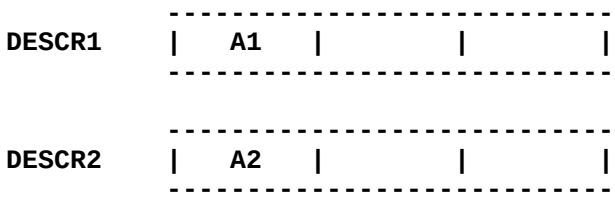
==========

126. PUTAC (put address with offset constant)

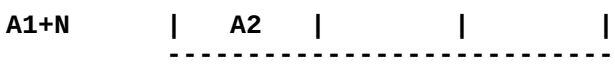
```
-------------------------------------
|          PUTAC     DESCR1,N,DESCR2 |
-------------------------------------
```

PUTAC is used to put an address field into a descriptor located at a constant offset.

Data Input to PUTAC



Data Altered by PUTAC



==========

127. PUTD (put descriptor)

```
-----------------------------------------
|          PUTD      DESCR1,DESCR2,DESCR3 |
-----------------------------------------
```

PUTD is used to put a descriptor.

Data Input to PUTD

```
            ----------------------------
DESCR1      |   A1   |        |        |
            ----------------------------

            ----------------------------
DESCR2      |   A2   |        |        |
            ----------------------------

            ----------------------------
DESCR3      |   A    |   F    |   V    |
            ----------------------------
```

Data Altered by PUTD

```
            ----------------------------
A1+A2       |   A    |   F    |   V    |
            ----------------------------
```

==========

128. PUTDC (put descriptor with constant offset)

```
--------------------------------------
|          PUTDC     DESCR1,N,DESCR2 |
--------------------------------------
```

PUTDC is used to put a descriptor at a location with a constant offset.

Data Input to PUTDC

```
            ----------------------------
DESCR1      |   A1   |        |        |
            ----------------------------

            ----------------------------
DESCR2      |   A    |   F    |   V    |
            ----------------------------
```

Data Altered by PUTDC

```
            ----------------------------
A1+N        |   A    |   F    |   V    |
            ----------------------------
```

==========

129. PUTLG (put specifier length)

```
--------------------------------
|          PUTLG     SPEC,DESCR |
--------------------------------
```

PUTLG is used to put a length into a specifier.

Data Input to PUTLG

DESCR	I			

Data Altered by PUTLG

SPEC					I

Programming Notes:

1. I is always nonnegative.

==========

130. PUTSPC (put specifier with offset constant)

```
-----------------------------------
|          PUTSPC    DESCR,N,SPEC |
-----------------------------------
```

PUTSPC is used to put a specifier.

Data Input to PUTSPC

DESCR	A1					
SPEC	A	F	V	O	L	

Data Altered by PUTSPC

A1+N	A	F	V	O	L	

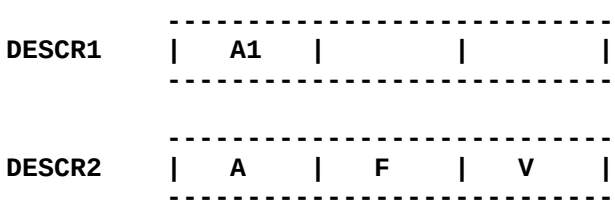
==========

131. PUTSTD (put descriptor with constant offset into SIL stack)

```
-------------------------------------
|          PUTSTD    DESCR1,N,DESCR2 |
-------------------------------------
```

PUTSTD is used to put a descriptor at a location, in the SIL stack, with a constant offset. This is similar to PUTDC which does not put the descriptor into the SIL stack.

Data Input to PUTSTD



Data Altered by PUTSTD

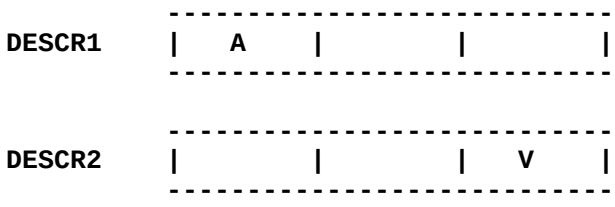
==========

132. PUTVC (put value field with offset constant)

```
--------------------------------------
|          PUTVC     DESCR1,N,DESCR2 |
--------------------------------------
```

PUTVC is used to put a value field into a descriptor at a location with a constant offset.

Data Input to PUTVC



Data Altered by PUTVC

==========

133. R2HS (convert double real to 8 hex digit single precision string)

```
---------------------------------
|          R2HS      SPEC,DESCR |
---------------------------------
```

R2HS is used to convert a double precision real into the big endian representation single precision real as 8 hexadecimal digits. NUM is the double precision real number input. L should be 8. B1 should be the high order hex digit of the result. B8 should be the low order hex digit. Each of the B1...B8 should be either the "0" to "F" characters.

Data Input to R2HS

SPEC	A			O	L

DESCR	NUM

Data Altered by R2HS

SPEC	A	0	0	O	L	

A+O	B1	...	B8

==========

134. R2HX (convert double real to 20 hex digit extended double precision string)

R2HX	SPEC,DESCR

R2HX is used to convert a double precision real into the big endian representation extended double precision real as 20 hexadecimal digits. NUM is the double precision real number input. L should be 20. B1 should be the high order hex digit of the result. B20 should be the low order hex digit. Each of the B1...B20 should be either the "0" to "F" characters.

Data Input to R2HX

SPEC	A			O	L

DESCR	NUM

Data Altered by R2HX

SPEC	A	0	0	O	L	

A+O	B1	...	B20

==========

135. RANDOM (pseudo random number generator)

```
-----------------------------------
|          RANDOM    DESCR1,DESCR2 |
-----------------------------------
```

RANDOM is used to generate a random integer N. If X is not zero, it should be used to set the random number seed.

Data Input to RANDOM

```
          ----------------------------
DESCR2    |   X    |        |        |
          ----------------------------
```

Data Altered by RANDOM

```
          ----------------------------
DESCR1    |   N    |   0    |   I    |
          ----------------------------
```

==========

136. RCALL (recursive call)

```
-------------------------------------------------------------------
|          RCALL     DESCR,PROC,(DESCR1,...,DESCRN),(LOC1,...,LOCM) |
-------------------------------------------------------------------
```

RCALL is used to perform a recursive call. DESCR is the descriptor that receives the value upon return from the call. PROC is the procedure being called. DESCR1,...,DESCRN are descriptors whose values are passed to PROC. LOC1,...,LOCM are locations to transfer to upon return according to the return exit signaled. The old stack pointer (A0) is saved on the stack, the current stack pointer becomes the old stack pointer, and a new current stack pointer is generated as indicated. The return location LOC is saved on the stack so that the return can be properly made. The values of the arguments DESCR1,...,DESCRN are placed on the stack and processed right to left. Note that their order is the opposite of the order that would be obtained by using PUSH.

At the return location LOC, code similar to that shown should be assembled. OP represents an instruction that stores the value returned by PROC in DESCR.

Data Input to RCALL

```
          ----------------------------
CSTACK    |   A    |        |        |
          ----------------------------

          ----------------------------
OSTACK    |   A0   |        |        |
          ----------------------------

          ----------------------------
DESCR1    |   A1   |   F1   |   V1   |
          ----------------------------

                    .
                    .
                    .
```

```
          ----------------------------
DESCRN    |   AN   |   FN   |   VN   |
          ----------------------------
```

Data Altered by RCALL

```
          ----------------------------
A+D       |   A0   |   0    |   0    |
          ----------------------------

          ----------------------------
A+2D      |  LOC   |   0    |   0    |
          ----------------------------

          ----------------------------
A+3D      |   AN   |   FN   |   VN   |
          ----------------------------

                      .
                      .
                      .
          ----------------------------
A+D*(2+N) |   A1   |   F1   |   V1   |
          ----------------------------

          ------------------------------
CSTACK    |A+(2+N)*D|        |        |
          ------------------------------

          ----------------------------
OSTACK    |   A    |        |        |
          ----------------------------
```

Return Code at LOC

```
 LOC      OP      DESCR1
          BRANCH  LOC1
             .
             .
             .
          BRANCH  LOCM
```

Programming Notes:

1. RCALL and RRTURN are used in combination, and their relation to each other must be thoroughly understood in order to implement them correctly.

2. Ordinarily OP is an instruction to store the value returned by RRTURN.

3. DESCR sometimes is omitted. In this case, any value returned by RRTURN is ignored and OP should perform no operation.

4. (DESCR1,...,DESCRN) sometimes is entirely omitted. In this case N should be taken to be zero in interpreting the figures.

5. Any of the locations LOC1,...,LOCM may be omitted. As in the case of operations with omitted conditional branches, control then passes to the operation following the RCALL.

6. The return indicated by RRTURN may be M+1, in which case control is passed to the operation following the RCALL.

7. The return indicated by RRTURN is never greater than M+1.

8. RCALL typically must save program state information. On the IBM System/360, this consists of the location LOC and a base register for the procedure containing the RCALL. This information is pushed onto the stack. In pushing information onto the stack, care must be taken to observe the rules concerning the use of descriptors. The rest of the SNOBOL5 system treats the stack as descriptors, and the flag fields of descriptors used to save program state information must be set to zero.

==========

137. RCOMP (real comparison)

```
-------------------------------------------------------------
|          RCOMP     DESCR1,DESCR2,GTLOC,EQLOC,LTLOC,NANLOC |
-------------------------------------------------------------
```

RCOMP is used to compare two real numbers. If R1 > R2, transfer is to GTLOC. If R1 = R2, transfer is to GTLOC. If R1 < R2, transfer is to LTLOC. If R1 or R2 is NaN (IEEE Not a Number), transfer is to NANLOC.

Data Input to RCOMP

```
           ----------------------------
DESCR1     |   R1   |        |        |
           ----------------------------

           ----------------------------
DESCR2     |   R2   |        |        |
           ----------------------------
```

Programming Notes:

1. Only Minnesota SNOBOL4 and Oregon SNOBOL5 have and use the NANLOC operand.

==========

138. REALST (convert real number to string)

```
---------------------------------
|          REALST    SPEC,DESCR |
---------------------------------
```

REALST is used to convert a real number into a specified string.

Data Input to REALST

```
           ----------------------------
DESCR      |   R    |        |        |
           ----------------------------
```

Data Altered by REALST

```
           ---------------------------------------------
SPEC       | BUFFER |   0    |   0    |   0    |   L    |
           ---------------------------------------------

           ----------------------------
BUFFER     |   C1   |  ...   |   CL   |
           ----------------------------
```

Programming Notes:

1. C1...CL should represent the real number R in the SNOBOL5 fashion, containing a decimal point and having at least one digit before the decimal point, zeroes being added as necessary. If R is negative, the string should begin with a minus sign. If e notation exponent form is implemented, it should also be supported in SPREAL and the STREAM syntax tables (which it curently is). See &FLTDEC and &FLTSIG keywords.

2. The number of digits (and hence the size of BUFFER) required is machine dependent and depends on the range available for real numbers.

3. BUFFER is local to REALST and its contents may be overwritten by a subsequent use of REALST.

4. Infinity is converted to the string "INFINITY". NaN's are converted to the string "NAN".

==========

139. REMSP (specify remaining string)

```
---------------------------------------
|          REMSP     SPEC1,SPEC2,SPEC3 |
---------------------------------------
```

REMSP is used to obtain a remainder specifier resulting from the deletion of a specified length at the beginning.

Data Input to REMSP

	A2	F2	V2	O2	L2
SPEC2					
SPEC3					L3

Data Altered by REMSP

	A2	F2	V2	O2+L3	L2-L3
SPEC1					

Programming Notes:

1. SPEC1 and SPEC3 may be the same.

2. L2-L3 is never negative.

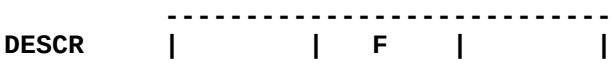
==========

140. RESETF (reset flag)

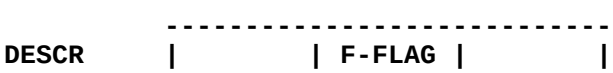
```
---------------------------------
|          RESETF    DESCR,FLAG |
---------------------------------
```

RESETF is used to reset (delete) a flag from a descriptor.

Data Input to RESETF



Data Altered by RESETF



Programming Notes:

1. Only FLAG is removed from the flags in F. Any other flags are left unchanged.

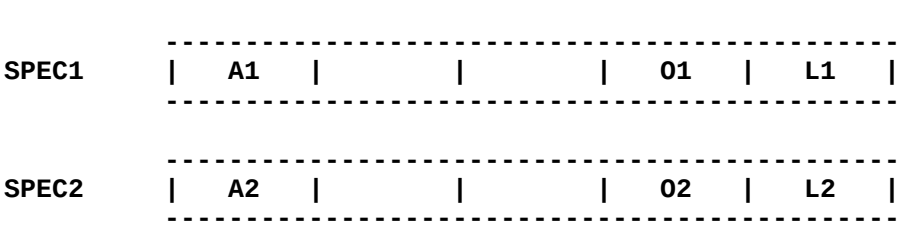
2. If F does not contain FLAG, no data is altered.

==========

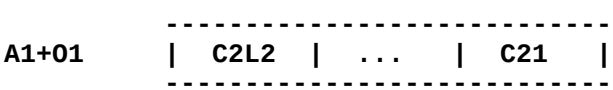
141. REVERSE (reverse the order of characters in a string)

REVERSE is used to reverse the order of characters in string specified by SPEC2 and place the result in SPEC1. L1 and L2 must be equal.

Data Input to REVERSE

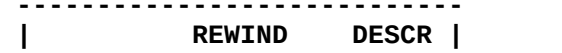


Data Altered by REVERSE



==========

142. REWIND (rewind file)



REWIND is used to rewind the file associated with the unit reference number I.

Data Input to REWIND

DESCR	I		

Programming Notes:

1. Refer to Section 2.1 for a discussion of unit reference numbers.

==========

143. RLINT (convert real number to integer)

```
-----------------------------------------------
|          RLINT     DESCR1,DESCR2,FLOC,SLOC |
-----------------------------------------------
```

RLINT is used to convert a real number to an integer. If the magnitude of R exceeds the magnitude of the largest integer, transfer is to FLOC. Otherwise transfer is to SLOC.

Data Input to RLINT

DESCR2	R		

Data Altered by RLINT

DESCR1	I(R)	0	I

Programming Notes:

1. I(R) is the integer equivalent of the real number R.

2. The fractional part of R is discarded.

3. I is a symbol defined in the source program and is the code for the integer data type.

==========

144. RPAD (pad a string on the right with blanks or other character)

```
---------------------------------------
|          RPAD      SPEC1,SPEC2,SPEC3 |
---------------------------------------
```

RPAD is used to pad the string specified by SPEC2 on the right side with the first character of the string specified by SPEC3. If L3 is zero or if SPEC3 was omitted, then a blank character is used for padding. If L1 <= L2, the the string is simply copied without padding.

Data Input to RPAD

SPEC1	A1			O1	L1

```
          ---------------------------------------------
SPEC2     |   A2   |        |        |   O2   |   L2   |
          ---------------------------------------------

          ---------------------------------------------
SPEC3     |   A3   |        |        |   O3   |   L3   |
          ---------------------------------------------

          ----------------------------
A2+O2     |   C2   |  ...   |  C2L2  |
          ----------------------------

          ----------------------------
A3+O3     |  C31   |  ...   |  C3L3  |
          ----------------------------
```

Data Altered by RPAD

```
          ------------------------------------------------------------
A1+O1     |   C2   |  ...   |  C2L2  |  C31   |  ...   |  C31   |
          ------------------------------------------------------------
```

==========

145. RPLACE (replace characters)

```
----------------------------------------
|          RPLACE    SPEC1,SPEC2,SPEC3 |
----------------------------------------
```

RPLACE is used to replace characters in a string. SPEC2 specifies a set of characters to be replaced. SPEC3 specifies the replacement to be made for the characters specified by SPEC2. The replacement is described by the following rules.
For I = 1,...,L

F(CI) = CI if CI not= C2J for any J (1 <= J <= L2)
F(CI) = C3J if CI = C2J for some J (1 <= J <= L2)

Data Input to RPLACE

```
          ---------------------------------------------
SPEC1     |   A1   |        |        |   O1   |   L    |
          ---------------------------------------------

          ---------------------------------------------
SPEC2     |   A2   |        |        |   O2   |   L2   |
          ---------------------------------------------

          ---------------------------------------------
SPEC3     |   A3   |        |        |   O3   |   L2   |
          ---------------------------------------------

                              .
                              .
                              .
          ----------------------------
A1+O1     |   C1   |  ...   |   CL   |
          ----------------------------

          ----------------------------
A2+O2     |  C21   |  ...   |  C2L2  |
          ----------------------------
```

```
          ----------------------------
A3+O3     |  C31   |  ...   |  C3L2  |
          ----------------------------
```

Data Altered by RPLACE

```
          ----------------------------
A1+O1     | F(C1)  |  ...   | F(CL)  |
          ----------------------------
```

Programming Notes:

1. L may be zero.

2. If there are duplicate characters in C21...C2L2, replacement should be made corresponding to the last instance of the character. That is, if

C2I = C2J = ... = C2K (I < J < K)

then

F(CI) = C3K

3. RPLACE is used only in the SNOBOL5 REPLACE function. It is not essential that RPLACE be implemented as such. If it is not, RPLACE should transfer to UNDF to provide an appropriate error comment.

==========

146. RRTURN (recursive return)

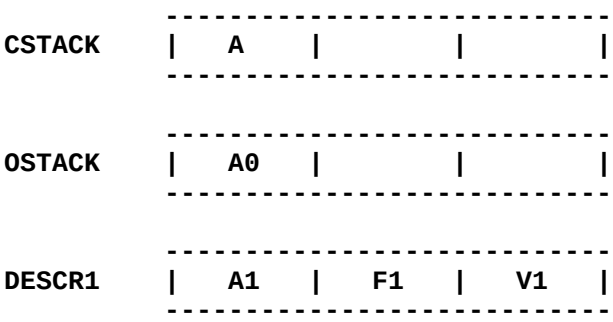
```
------------------------------
|          RRTURN    DESCR,N |
------------------------------
```

RRTURN is used to return from a recursive call. DESCR is the descriptor whose value is returned. The stack pointers are repositioned as shown. At the location LOC, code similar to that shown is assembled by the RCALL to which return is to be made. OP represents an instruction that is used by RRTURN to return the value of DESCR. Control is transferred to LOCn corresponding to the number N given in the RRTURN.

Data Input to RRTURN

```
          ----------------------------
OSTACK    |   A    |        |        |
          ----------------------------

          ----------------------------
A+D       |   A0   |        |        |
          ----------------------------

          ----------------------------
A+2D      |  LOC   |        |        |
          ----------------------------

          ----------------------------
DESCR     |   A1   |   F1   |   V1   |
          ----------------------------
```

Data Altered by RRTURN



Return Code at LOC

```
 LOC     OP      DESCR1
         BRANCH  LOC1
            .
            .
            .
         BRANCH  LOCM
```

Programming Notes:

1. RCALL and RRTURN are used in combination, and their relation to each other must be thoroughly understood.

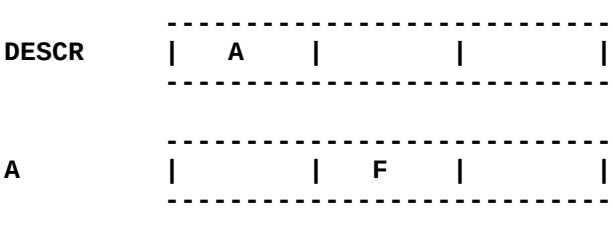
2. DESCR may be omitted. In this case, OP should not be executed. In Oregon SNOBOL5, this is indicated by a zero address for the first operand (DESCR) of the instruction.

==========

147. RSETFI (reset flag indirect)

RSETFI is used to reset (delete) a flag from a descriptor that is specified indirectly.

Data Input to RSETFI



Data Altered by RSETFI

Programming Notes:

1. Only FLAG is removed from the flags in F. Any other flags are left unchanged.

2. If F does not contain FLAG, no data is altered.

3. This macro is only used in garbage collection.

==========

148. SBREAL (subtract real numbers)

```
-----------------------------------------------------
|          SBREAL    DESCR1,DESCR2,DESCR3,FLOC,SLOC |
-----------------------------------------------------
```

SBREAL is used to subtract one real number from another. If the result is out of the range available for real numbers, transfer is to FLOC. Otherwise transfer is to SLOC.

Data Input to SBREAL

```
          ----------------------------
DESCR2    |   R2   |   F2   |   V2   |
          ----------------------------

          ----------------------------
DESCR3    |   R3   |        |        |
          ----------------------------
```

Data Altered by SBREAL

```
          ----------------------------
DESCR1    | R2-R3  |   F2   |   V2   |
          ----------------------------
```

==========

149. SEEK (SEEK to particular place in a file)

```
-------------------------------------------------------------
|          SEEK      DESCR1,DESCR2,DESCR3,DESCR4,FLOC,SLOC |
-------------------------------------------------------------
```

SEEK is used to position the next file read or write to a particular position in the file. UNIT specifies the I/O unit number this applies to. TYPE is either 0 for absolute seek, 1 for relative to current position seek, or 2 for relative to the end of the file seek. OFFSET is the position or offset of the seek according to TYPE. N receives the new absolute offset in the file.

Data Input to SEEK

```
          ----------------------------
DESCR1    |  UNIT  |        |        |
          ----------------------------

          ----------------------------
DESCR2    | OFFSET |        |        |
          ----------------------------

          ----------------------------
DESCR3    |  TYPE  |        |        |
          ----------------------------
```

Data Altered by SEEK

DESCR4	N		I

==========

150. SELBRA (select branch point)

```
-----------------------------------------
|          SELBRA    DESCR,(LOC1,...,LOCN) |
-----------------------------------------
```

SELBRA is used to alter the flow of program control by selecting a location from a list and branching to it. Transfer is to LOCI corresponding to I.

Data Input to SELBRA

DESCR	I		

Programming Notes:

1. Any of the locations may be omitted. As in the case of operations with omitted conditional branches, control then passes to the operation following SELBRA.

2. If I = N+1, control is passed to the operation following SELBRA.

3. I is always in the range 1 <= I <= N+1. For debugging purposes, it may be useful to verify that I is within this range.

==========

151. SETAA (set address filed to a constant address)

```
------------------------------
|          SETAA     DESCR,A |
------------------------------
```

SETAA is used to set the address field of a descriptor to a constant. This macro is similar to SETAC.

Data Altered by SETAA

DESCR	A		

Programming Notes:

1. N is a relocatable address.

==========

152. SETAC (set address to constant)

```
------------------------------
|          SETAC     DESCR,N |
------------------------------
```

SETAC is used to set the address field of a descriptor to a constant.

Data Altered by SETAC

DESCR	N		

Programming Notes:

1. N may be a relocatable address.

2. N is often 0, 1, or D.

3. N is never negative.

==========

153. SETAV (set address from value field)

```
-----------------------------------
|          SETAV     DESCR1,DESCR2 |
-----------------------------------
```

SETAV sets the address field of one descriptor from the value field of another and zero the flag and value fields of the target descriptor.

Data Input to SETAV

DESCR2		V	

Data Altered by SETAV

DESCR1	V	0	0

==========

154. SETAVO (set address from value field)

```
-----------------------------------
|          SETAVO    DESCR1,DESCR2 |
-----------------------------------
```

SETAVO sets the address field of one descriptor from the value field of another.

Data Input to SETAVO

DESCR2		V	

Data Altered by SETAVO

DESCR1	V		

==========

155. SETF (set flag)

```
---------------------------------
|          SETF      DESCR,FLAG |
---------------------------------
```

SETF is used to set (add) a flag in the flag field of DESCR.

Data Input to SETF

```
            ----------------------------
DESCR       |        |   F    |        |
            ----------------------------
```

Data Altered by SETF

```
            ----------------------------
DESCR       |        | F+FLAG |        |
            ----------------------------
```

Programming Notes:

1. FLAG is added to the flags already present in F. The other flags are left unchanged.

2. If F already contains FLAG, no data is altered.

==========

156. SETFI (set flag indirect)

```
---------------------------------
|          SETFI     DESCR,FLAG |
---------------------------------
```

SETFI is used to set (add) a flag in the flag field of a descriptor specified indirectly.

Data Input to SETFI

```
            ----------------------------
DESCR       |   A    |        |        |
            ----------------------------

            ----------------------------
A           |        |   F    |        |
            ----------------------------
```

Data Altered by SETFI

```
            ----------------------------
A           |        | F+FLAG |        |
            ----------------------------
```

Programming Notes:

1. FLAG is added to the flags already present in F. The other flags are left unchanged.

2. If F already contains FLAG, no data is altered.

==========

157. SETLC (set length of specifier to constant)

```
----------------------------
|          SETLC     SPEC,N |
----------------------------
```

SETLC is used to set the length of a specifier to a constant.

Data Altered by SETLC

SPEC					N

Programming Notes:

1. N is never negative.

2. N is often 0.

==========

158. SETSIZ (set size)

```
-----------------------------------
|          SETSIZ    DESCR1,DESCR2 |
-----------------------------------
```

SETSIZ is used to set the size into the value field of a title descriptor.

Data Input to SETSIZ

DESCR1	A		
DESCR2	I		

Data Altered by SETSIZ

A			I	

Programming Notes:

1. I is always positive and small enough to fit into the value field.

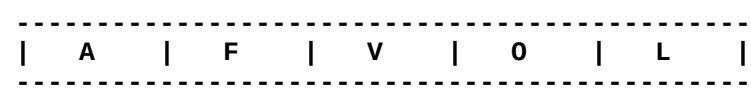
==========

159. SETSP (set specifier)

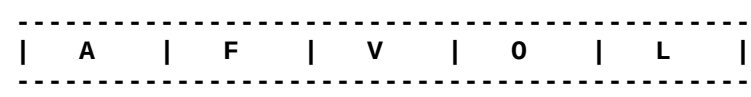
```
---------------------------------
|          SETSP     SPEC1,SPEC2 |
---------------------------------
```

SETSP is used to set one specifier equal to another.

Data Input to SETSP



Data Altered by SETSP



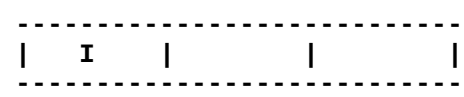
==========

160. SETVA (set value field from address)

```
-----------------------------------
|          SETVA     DESCR1,DESCR2 |
-----------------------------------
```

SETVA is used to set the value field of one descriptor from the address field of another.

Data Input to SETVA



Data Altered by SETVA

Programming Notes:

1. I is always positive and small enough to fit into the value field.

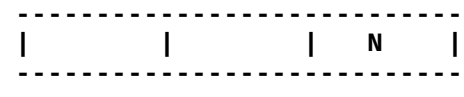
==========

161. SETVC (set value to constant)

```
------------------------------
|          SETVC     DESCR,N |
------------------------------
```

SETVC is used to set the value field of a descriptor to a constant.

Data Altered by SETVC



Programming Notes:

1. N is always positive and small enough to fit into the value field.

==========

162. SHORTN (shorten specifier)

```
----------------------------
|          SHORTN    SPEC,N |
----------------------------
```

SHORTN is used to shorten the specification of a string.

Data Input to SHORTN

SPEC					**L**

Data Altered by SHORTN

SPEC					**L-N**

Programming Notes:

1. L-N is never negative.

==========

163. SIN (sine)

```
-------------------------------------
|          SIN       DESCR1,DESCR2 |
-------------------------------------
```

SIN is used to take the sine of an angle in radians.

Data Input to SIN

DESCR2	**A**		**R**

Data Altered by SIN

DESCR1	**SIN(A)**	**0**	**R**

==========

164. SORT (sort an array in place)

```
-----------------------------------------
|          SORT      DESCR1,DESCR2,FLOC |
-----------------------------------------
```

SORT is used to sort a one or two dimensional array pointed to by A. N specifies which column to sort on. Numerics sort as higher than any numeric (INTEGER or REAL). If the items are not strings or numerics, the comparison is only on type. The sign of N specifies the direction of the sort (ascending or descending). See

SORT.INC for details about the array structure at A. The sort should be stable so that successive sorts don't destroy the order of prior ones. That is, entries with equal values in the sort column should not have their order altered. If any of the conditions are not met, execution transfers to FLOC.

Data Input to SQRT

DESCR1	A		
DESCR2	N		

Data Altered by SORT

Array located at A.

==========

165. SPCINT (convert specifier to integer)

SPCINT	DESCR,SPEC,FLOC,SLOC

SPCINT is used to convert a specified string to a integer. I(S) is a signed integer resulting from the conversion of the string C1...CL. If C1...CL does not represent an integer or if the integer it represents is too large to fit the address field, transfer is to FLOC. Otherwise transfer is to SLOC. All characters C1 to CL must be part of the integer, no extra blanks, for example, allowed.

Data Input to SPCINT

SPEC	A			O	L
A+O	C1	...	CL		

Data Altered by SPCINT

DESCR	I(S)	0	I

Programming Notes:

1. I is a symbol defined in the source program and is the code for the integer data type.

2. C1...CL may begin with a sign (plus or minus) and may contain indefinite number of leading zeros. Consequently the value of L itself does not determine whether the integer represented is too large to fit into an address field.

3. A sign alone is not a valid integer.

4. If L = 0, I(S) should be the integer 0.

5. DESCR must not be altered if taking failure exit.

==========

166. SPEC (assemble specifier)

```
--------------------------------
| LOC      SPEC      A,F,V,O,L |
--------------------------------
```

SPEC is used to assemble a specifier.

Data Assembled by SPEC

```
          ---------------------------------------------
LOC       |   A    |   F    |   V    |   O    |   L    |
          ---------------------------------------------
```

==========

167. SPOP (pop specifier from stack)

```
----------------------------------------
|          SPOP      (SPEC1,...,SPECN) |
----------------------------------------
```

SPOP is used to pop a list of specifiers from the system stack.

Data Input to SPOP

```
          ----------------------------
CSTACK    |   A    |        |        |
          ----------------------------

          ---------------------------------------------
A+D-S      |   A1   |   F1   |   V1   |   O1   |   L1   |
          ---------------------------------------------

                               .
                               .
                               .
          ---------------------------------------------
A+D-(N*S) |   AN   |   FN   |   VN   |   ON   |   LN   |
          ---------------------------------------------
```

Data Altered by SPOP

```
          ----------------------------
CSTACK    |A-(N*S) |        |        |
          ----------------------------

          ---------------------------------------------
SPEC1      |   A1   |   F1   |   V1   |   O1   |   L1   |
          ---------------------------------------------

                               .
                               .
                               .
          ---------------------------------------------
SPECN      |   AN   |   FN   |   VN   |   ON   |   LN   |
          ---------------------------------------------
```

Programming Notes:

1. If A-(N*S) < STACK, stack underflow occurs. This condition indicates a programming error in the implementation of the macro language. An appropriate error termination for this error may be obtained by transferring to the program location INTR10 if the condition is detected.

2.N is always less than 5.

==========

168. SPREAL (convert specified string to real number)

```
-----------------------------------------
|          SPREAL    DESCR,SPEC,FLOC,SLOC |
-----------------------------------------
```

SPREAL is used to convert a specified string into a real number. R(S) is a signed real number resulting from the conversion of the string S = C1. If C1...CL does not represent a real number, or if the real number it represents is out of the range available for real numbers, transfer is to FLOC. All of the characters must be part of the number, no extra blanks or other characters allowed. Otherwise transfer is to SLOC. If e notation exponent form is implemented, it should also be supported in REALST and the STREAM syntax tables (in which it curently is). See &FLTDEC and &FLTSIG keywords.

Data Input to SPREAL

SPEC	A			O	L

A+O	C1	...	CL

Data Altered by SPREAL

DESCR	R(S)	0	R

Programming Notes:

1. R is a symbol defined in the source program and is the code for the real data type.

2. C1,...,CL may begin with a sign (plus or minus) and may contain an indefinite number of leading zeros. C1,...,CL will contain a decimal point if it represents a real number, and have at least one digit before the decimal point.

3. If L = 0, R(S) should be the real number 0.0.

4. If C1,...,CL is the string 'NAN' or 'INFINITY', then it is converted to the appropriate IEEE float form.

5. DESCR must not be altered on failure exit.

==========

169. SPUSH (push specifiers onto stack)

```
---------------------------------------
|          SPUSH     (SPEC1,...,SPECN) |
---------------------------------------
```

SPUSH is used to push a list of specifiers onto the system stack.

Data Input to SPUSH

```
            ----------------------------
CSTACK      |   A    |        |        |
            ----------------------------

            ---------------------------------------------
SPEC1       |   A1   |   F1   |   V1   |   O1   |   L1   |
            ---------------------------------------------

                                .
                                .
                                .
            ---------------------------------------------
SPECN       |   AN   |   FN   |   VN   |   ON   |   LN   |
            ---------------------------------------------
```

Data Altered by SPUSH

```
            ----------------------------
CSTACK      |A+(S*N) |        |        |
            ----------------------------

            ---------------------------------------------
A+D         |   A1   |   F1   |   V1   |   O1   |   L1   |
            ---------------------------------------------

                                .
                                .
                                .
            ---------------------------------------------
A+D+S*N-S   |   AN   |   FN   |   VN   |   ON   |   LN   |
            ---------------------------------------------
```

Programming Notes:

1. If A+(S*N) > STACK+(STSIZE*D), stack overflow occurs, transfer should be made to the program location OVER, which will result in an appropriate error termination.

2. N is always less than 5.

==========

170. SQRT (square root)

```
-----------------------------------
|          SQRT      DESCR1,DESCR2 |
-----------------------------------
```

SQRT is used to take the square root of a REAL number.

Data Input to SQRT

DESCR2	A		R

Data Altered by SQRT

DESCR1	SQRT(A)	0	R

==========

171. STPRNT (string print)

STPRNT DESCR1,DESCR2,SPEC

STPRNT is used to print a string. The string C11...C1L is printed on the file associated with unit reference number I. C21...C2M is the output format. J is an integer specifying a condition signaled by the output routine.

Data Input to STPRNT

DESCR2	A				
A+D	I				
A+2D	A2				
A2			M		
A2+4D	C21	...	C2M		
SPEC	A1			O1	L
A1+O1	C11	...	C1L		

Data Altered by STPRNT

DESCR1	J		

Programming Notes:

1. The format C21...C2M is a FORTRAN IV format in "undigested" form. See FORMAT.

2. Both C11...C1L and C21...C2M begin at descriptor boundaries.

3. The condition J set in the address field of DESCR1 is not used.

==========

172. STPRNTB (string print both)

```
----------------------------------------
|          STPRNTB   DESCR1,DESCR2,SPEC |
----------------------------------------
```

STPRNTB is the same as STPRNT except it indicates that non-user data is being printed. The implementation may print this data on the console as well as writing to a listing file. Data written with OUTPUT is also in this category.

==========

173. STREAD (string read)

```
-----------------------------------------------
|          STREAD    SPEC,DESCR,EOF,ERROR,SLOC |
-----------------------------------------------
```

STREAD is used to read a string of maximum length L. If the record read is shorter than L, then blank characters are padded to reach length L. The string C1...CL is read from the file associated with unit reference number I. If an end-of-file is encountered, transfer is to EOF. If a reading error occurs (eg reading after EOF), transfer is to ERROR. Otherwise transfer is to SLOC. STREAD is used primarily for reading the initial SNOBOL source statements.

Data Input to STREAD

```
           ----------------------------
DESCR      |   I    |        |        |
           ----------------------------

           ---------------------------------------------
SPEC       |   A    |        |        |   O    |   L    |
           ---------------------------------------------
```

Data Altered by STREAD

```
           ----------------------------
A+O        |   C1   |  ...   |   CL   |
           ----------------------------
```

==========

174. STRDNP (string read, variable length, no pad)

```
-----------------------------------------------
|          STRDNP    SPEC,DESCR,EOF,ERROR,SLOC |
-----------------------------------------------
```

STRDNP is used to read a string of length up to L. The string C1...CN is read from the file associated with unit reference number I. If an end-of-file is encountered, transfer is to EOF. If a reading error occurs (eg reading after EOF), transfer is

to ERROR. Otherwise transfer is to SLOC.

Data Input to STRDNP

DESCR	I		

SPEC	A			O	L

Data Altered by STRDNP

SPEC					N

A+O	C1	...	CN

==========

175. STREAM (stream for token)

STREAM	SPEC1,SPEC2,TABLE,ERROR,RUNOUT,SLOC

STREAM is used to locate a syntactic token at the beginning of the string specified by SPEC2. If there is an I (1 <= I <= L) such that TI is ERROR, STOP, or STOPSH, and J is the least such I, then

a): if TJ is ERROR, transfer is to ERROR.
b): if TJ is STOPSH, transfer is to SLOC.
c): oOtherwise transfer is to RUNOUT.

In the figures that follow, J is the least value of I for which TI is STOP or STOPSH. P is the last value of P (1 <= I <= J) that is nonzero (i.e. for which a PUT is specified in the syntax table description for the tables given). If no PUT is specified, P is zero.

Data Input to STREAM

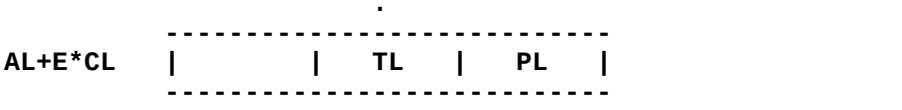
SPEC2	A	F	V	O	L

A+O	C1	...	CJ	CJ+1	...	CL

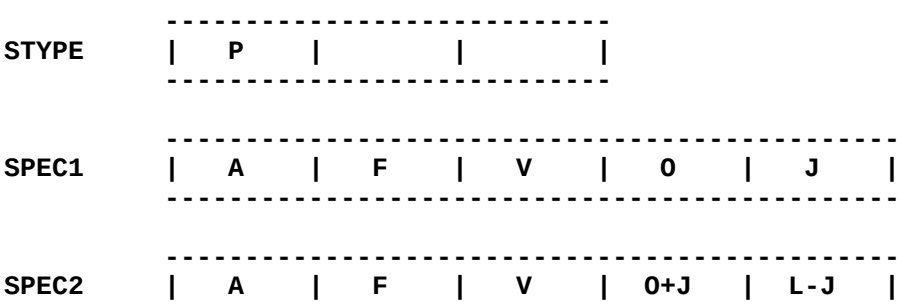
TABLE+E*C1	A2	T1	P1

A2+E*C2	A3	T2	P2

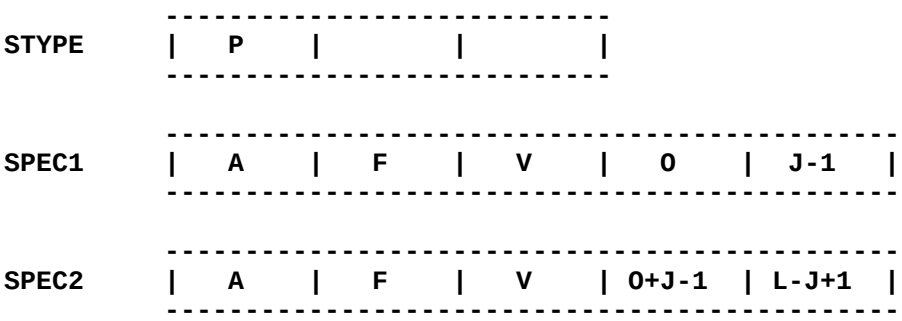
.
.



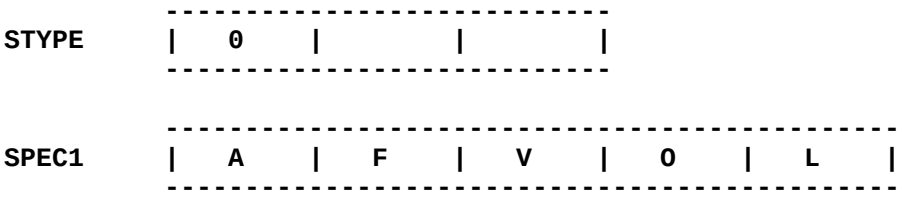
Data Altered by STREAM if Termination is STOP



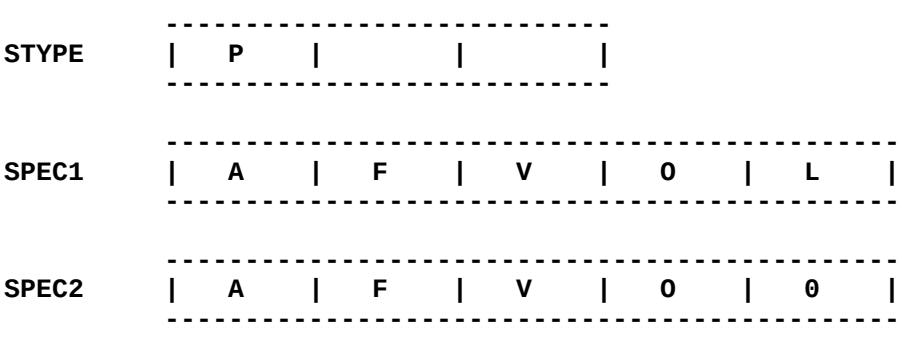
Data Altered by STREAM if Termination is STOPSH



Data Altered by STREAM if Termination is ERROR



Data Altered by STREAM if Termination is RUNOUT



Programming Notes:

1. Termination with STOP or STOPSH may occur on the last character, CL.

2. If L = 0 (i.e. if SPEC2 specifies the null string), RUNOUT occurs. In this case the address field of STYPE should be set to 0.

3. See Section 4.2.

4. Be careful to distinguish between the letter O and the digit 0 above.

==========

176. STRING (assemble specified string)

```
--------------------------------
| LOC      STRING    'C1...CL' |
--------------------------------
```

STRING is used to assemble a string and a specifier to it.

Data Assembled by STRING

```
          ---------------------------------------------
LOC       |   A    |   0    |   0    |   0    |   L    |
          ---------------------------------------------

          ----------------------------
A         |   C1   |  ...   |   CL   |
          ----------------------------
```

Programming Notes:

1. Note that LOC is the location of the specifier, not the string. The string may immediately follow the specifier, or it may be assembled at a remote location.

==========

177. SUBSP (substring specification)

```
-------------------------------------------------
|          SUBSP     SPEC1,SPEC2,SPEC3,FLOC,SLOC |
-------------------------------------------------
```

SUBSP is used to specify an initial substring of a specified string. If L3 >= L2, transfer is to SLOC. Otherwise transfer is to FLOC and SPEC1 is not altered.

Data Input to SUBSP

```
          ---------------------------------------------
SPEC2     |        |        |        |        |   L2   |
          ---------------------------------------------

          ---------------------------------------------
SPEC3     |   A3   |   F3   |   V3   |   O3   |   L3   |
          ---------------------------------------------
```

Data Altered by SUBSP if L3 >= L2

```
          ---------------------------------------------
SPEC1     |   A3   |   F3   |   V3   |   O3   |   L2   |
          ---------------------------------------------
```

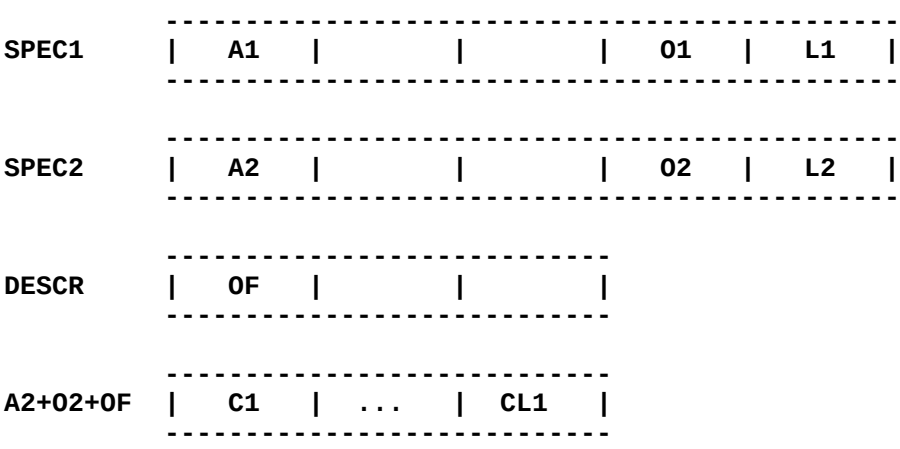
==========

178. SUBSTR (substring)

```
---------------------------------------
|          SUBSTR    SPEC1,SPEC2,DESCR |
---------------------------------------
```

SUBSTR extracts the substring at offset OF from SPEC2 and writes it to SPEC1. Length L1 will always be positive. The extracted characters will not be out of bounds of the string specified by SPEC2: OF+L1 <= L2.

Data Input to SUBSTR



Data Altered by SUBSTR

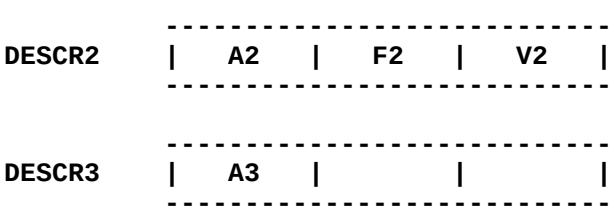
==========

179. SUBTRT (subtract addresses)

```
-----------------------------------------------------
|          SUBTRT    DESCR1,DESCR2,DESCR3,FLOC,SLOC |
-----------------------------------------------------
```

SUBTRT is used to subtract one address field from another. A2 and A3 are considered as signed integers. If A2-A3 is out of the range available for integers, transfer is to FLOC. Otherwise transfer is to SLOC.

Data Input to SUBTRT



Data Altered by SUBTRT

	A2-A3	F2	V2
DESCR1			

Programming Notes:

1. A2 and A3 may be relocatable addresses.

2. The test for success and failure is used in only one call of this macro. Hence the code to make the check is not needed in most cases.

3. DESCR1 and DESCR2 are often the same.

==========

180. SUM (sum addresses)

SUM	DESCR1,DESCR2,DESCR3,FLOC,SLOC

SUM is used to add two address fields. A and I are considered as signed integers. If A+I is out of the range available for integers, transfer is to FLOC. Otherwise transfer is to SLOC.

Data Input to SUM

	A	F	V
DESCR2			

	I		
DESCR3			

Data Altered by SUM

	A+I	F	V
DESCR1			

Programming Notes:

1. A may be a relocatable address.

2. The test for success and failure is used in only one call of this macro. Hence the code to make the check is not needed in most cases.

3. DESCR1 and DESCR2 are often the same. Also, DESCR1 and DESCR3 can be the same.

==========

181. SYSTEM (execute an operating system command)

SYSTEM	SPEC

SYSTEM passes the command string (C1...CL) to the operating system for execution. When execution is complete, the macro finishes.

Data Input to SYSTEM

SPEC	A			O	L
A+O	C1	...	CL		

==========

182. TAN (tangent)

TAN	DESCR1,DESCR2

TAN is used to take the tangent of an angle in radians.

Data Input to TAN

DESCR2	A		R

Data Altered by TAN

DESCR1	TAN(A)	0	R

==========

183. TESTAI (test bits in A field)

TESTAI	DESCR,BITS,ZLOC,NZLOC

TESTAI is used to test the bits in the A field of DESCR. BITS is an binary integer with the bits on to be tested. If A has any of the bits in BITS on, then transfer is to NZLOG. Otherwise transfer is to ZLOC.

Data Input to TESTAI

DESCR	A		

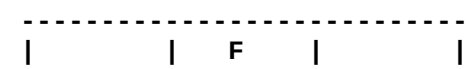
==========

184. TESTF (test flag)

TESTF	DESCR,FLAG,FLOC,SLOC

TESTF is used to test a flag field for the presence of a flag. If F contains FLAG, transfer is to SLOC. Otherwise transfer is to FLOC.

Data Input to TESTF



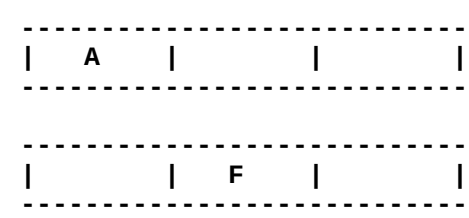
==========

185. TESTFI (test flag indirect)

```
-----------------------------------------
|          TESTFI    DESCR,FLAG,FLOC,SLOC |
-----------------------------------------
```

TESTFI is used to test an indirectly specified flag field for the presence of a flag. If F contains FLAG, transfer is to SLOC. Otherwise transfer is to FLOC.

Data Input to TESTFI



==========

186. TITLE (title assembly listing)

```
--------------------------------
|          TITLE     'C1...CN' |
--------------------------------
```

TITLE is used at assembly time to title the assembly listing of the SNOBOL5 system. TITLE should cause a page eject and title subsequent pages with C1...CN.
Programming Notes:

1. TITLE need not be implemented as such. It may simply perform no operation.

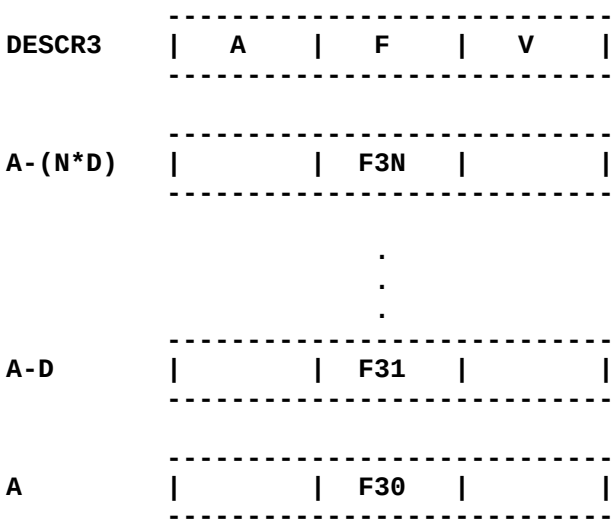
==========

187. TOP (get to top of block)

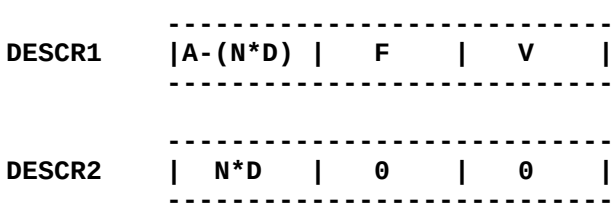
```
-----------------------------------------
|          TOP       DESCR1,DESCR2,DESCR3 |
-----------------------------------------
```

TOP is used to get to the top of a block of descriptors. Descriptors at A, A-D,...,A-(N*D) are examined successively for the first descriptor whose flag field contains the flag TTL. Data is altered as indicated, where F3N is the first field to contain TTL.

Data Input to TOP



Data Altered by TOP



Programming Notes:

1. N may be 0. That is, F30 may contain TTL.

==========

188. TRAPCK (check for control-break interrupt)

```
----------------------
|          TRAPCK     |
----------------------
```

TRAPCK is placed in a few spots in the SIL source so that execution can be neatly terminated via asynchronous means such as hitting the control-break key. If execution is to be terminated, then this macro should transfer to the label SYSCUT. Otherwise it should do nothing. This is placed in key points within the system so that infinite loops in pattern matching and statement execution can be stopped.

Programming Notes:

1. This is an new macro for Oregon SNOBOL5.

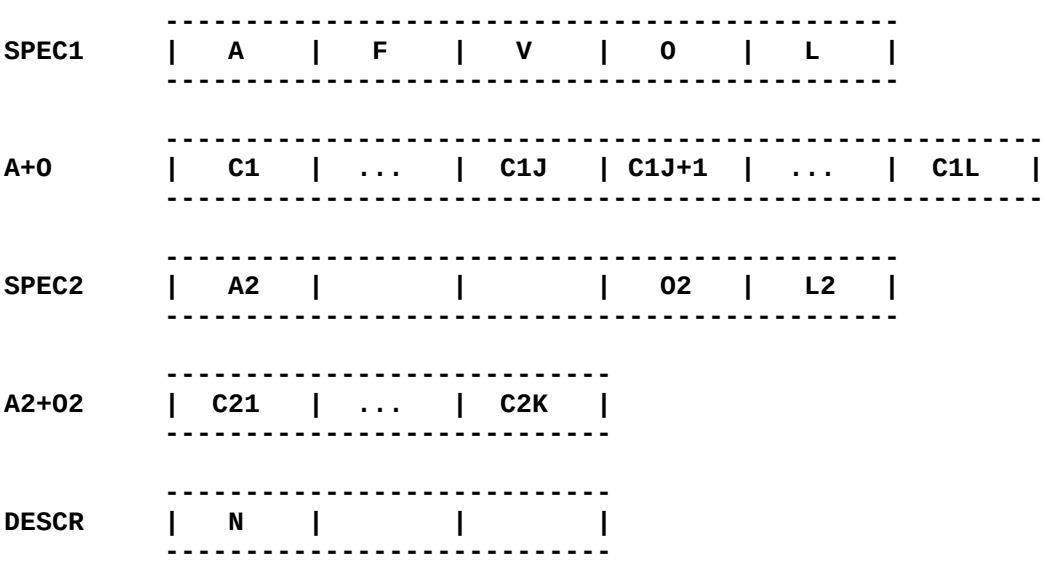
==========

189. TRIMSP (trim blanks from specifier)

```
---------------------------------------
|          TRIMSP    SPEC1,SPEC2,DESCR |
---------------------------------------
```

TRIMSP is used to obtain a specifier to the part of a specified string up to a trailing string of blanks. If DESCR is specified, the string is not trimmed shorter than length N. If SPEC2 is not a null string, then the characters C21 to C2K are what are trimmed from the string. Otherwise blanks and tabs are trimmed.

Data Input to TRIMSP



Data Altered by TRIMSP

Programming Notes:

1. If CL is not white space or one of the SPECT2 characters then J = L.

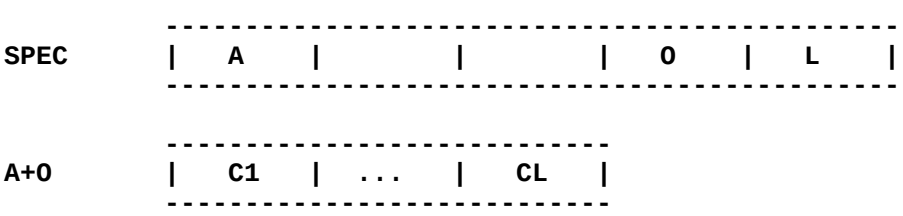
2. If L = 0, TRIMSP is equivalent to SETSP.

==========

190. UNLOAD (unload external function)

UNLOAD is used to unload an external function. C1...CL represents the name of the function that is to be unloaded.

Data Input to UNLOAD



Programming Notes:

1. UNLOAD is a system-dependent operation.

2. UNLOAD need not be implemented as such. If it is not, it should perform no operation, since the SNOBOL function UNLOAD, which uses the macro UNLOAD, has a valid use in undefining existing, but non-external, functions.

3. UNLOAD should do nothing if the function C1...CL is not a LOADed function.

4. UNLOAD is not implemented in Oregon SNOBOL5.

==========

191. VARID (compute variable identification numbers)

```
---------------------------------
|          VARID     DESCR,SPEC |
---------------------------------
```

VARID is used to compute two variable identification numbers from a specified string. K and M are computed by

```
K = F1(C1...CL)
M = F2(C1...CL)
```

where F1 and F2 are two (different) functions that compute pseudo-random numbers from the characters C1...CL. The numbers computed should be in the ranges

```
0 <= K <= (OBSIZ-1)*D
0 <= M <= SIZLIM
```

where OBSIZ is a program symbol defining the number of chains in variable storage and SIZLIM is a program symbol defining the largest integer that can be stored in the value field of a descriptor.

Data Input to VARID

```
          ------------------------------------------------
SPEC      |   A    |        |        |   O    |   L    |
          ------------------------------------------------

          ----------------------------
A+O       |   C1   |  ...   |   CL   |
          ----------------------------
```

Data Altered by VARID

```
          ----------------------------
DESCR     |   K    |        |   M    |
          ----------------------------
```

Programming Notes:

1. K is used to select one of a number of chains in variable storage. The K are address offsets that must fall on descriptor boundaries.

2. M is used to order variables (string structures) within a chain. See ORDVST.

3. The values of K and M should have as little correlation as possible with the characters C1...CL, since the "randomness" of the results determines the efficiency of variable access.

4. One simple algorithm consists of multiplying the first part of C1...CL by the last part, and separating the central portion of the result into K and M.

5. L is always greater than zero.

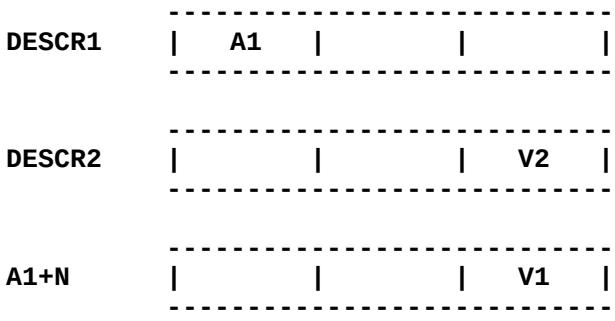
==========

192. VCMPIC (value field compare indirect with offset constant)

```
-------------------------------------------------------
|          VCMPIC    DESCR1,N,DESCR2,GTLOC,EQLOC,LTLOC |
-------------------------------------------------------
```

VCMPIC is used to compare a value field, indirectly specified with an offset constant, with another value field. V1 and V2 are considered as unsigned integers. If V1 > V2, transfer is to GTLOC. If V1 = V2, transfer is to EQLOC. If V1 < V2, transfer is to LTLOC.

Data Input to VCMPIC



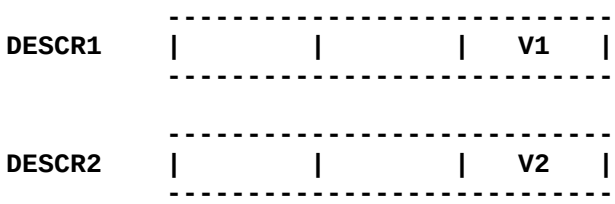
==========

193. VEQL (value fields equal test)

```
-------------------------------------------------
|          VEQL      DESCR1,DESCR2,NELOC,EQLOC |
-------------------------------------------------
```

VEQL is used to compare the value fields of two descriptors. V1 and V2 are considered as unsigned integers. If V1 = V2, transfer is to EQLOC. Otherwise transfer is to NELOC.

Data Input to VEQL



==========

194. VEQLC (value field equal to constant test)

```
-----------------------------------------
|          VEQLC     DESCR,N,NELOC,EQLOC |
-----------------------------------------
```

VEQLC is used to compare the value field of a descriptor to a constant. V is considered as an unsigned integer. If V = N, transfer is to EQLOC. Otherwise transfer is to NELOC.

Data Input to VEQLC

DESCR			V

Programming Notes:

1. N is never negative.

==========

195. XORFUN (perform a logical and function)

	XORFUN	SPEC1,SPEC2,SPEC3

XORFUN is used to logically eXclusive OR the bytes specified by strings SPEC2 and SPEC3 and place the result in SPEC1. If either SPEC2 or SPEC3 is shorter than the other, then the remaining characters of the shorter string are assumed to be zero bytes. F(Cn) = logical XOR of C2n and C3n.

Data Input to XORFUN

SPEC1	A1			O1	MAX(L2,L3)	
SPEC2	A2			O2	L2	
SPEC3	A3			O3	L3	
A2+O2	C21	...	C2L2			
A3+O3	C31	...	C3L2			

Data Altered by XORFUN

A1+O1	F(C1)	...	F(C(MAX(L2,L3)))

Programming Notes:

1. L2 and L3 may be zero.

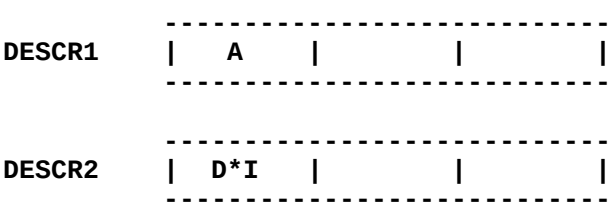
==========

196. ZERBLK (zero block)

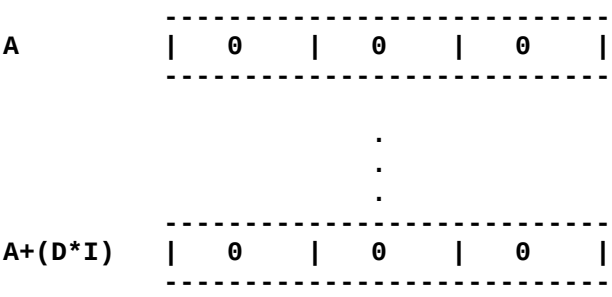
	ZERBLK	DESCR1,DESCR2

ZERBLK is used to zero a block of I+1 descriptors.

Data Input to ZERBLK



Data Altered by ZERBLK



Programming Notes:

1. I is always positive.

7. Implementation Notes

7.1 Optional Macros

There are several macros that are used in noncritical parts of the SNOBOL language. Some macros are used only to implement certain built-in functions. Others are required only for minor executive operations. The following list includes macros for which implementation is optional. For these macros, simple alternative implementations are suggested and the language features disabled are indicated. In selecting macros for inclusion in this list, a judgement was made concerning what features could be disabled and still leave SNOBOL a useful language.

Macro	Note	Alternative implementation	Features disabled
ADREAL	1	Branch to INTR10	Real arithmetic
BKSPCE		Branch to UNDF	The function BACKSPACE
CLERTB	2	Branch to UNDF	The functions ANY, NOTANY, SPAN, and BREAK
DATE		Set length of SPEC to 0	The function DATE
DVREAL	1	Set address of DESCR1 to 0	Real arithmetic
ENFILE		Branch to UNDF	The function ENDFILE
EXPINT		Branch to UNDF	Exponentiation of integers
EXREAL	1	Branch to INTR10	Real arithmetic
FILNAM		Perform no operation	Dynamic file name specification
GETBAL		Branch to UNDF	The built-in pattern BAL
INTRL	1	Perform no operation	Real arithmetic
LEXCMP	3	If GTLOC = LTLOC, branch to UNDF	The function LGT
LINK	4	Branch to INTR10	External functions
LOAD	4	Branch to UNDF	External functions
MNREAL	1	Branch to INTR10	Real arithmetic
MPREAL	1	Branch to INTR10	Real arithmetic
MSTIME		Set address of DESCR to 0	The function TIME, trace timing, post-run statistics
ORDVST		Perform no operation	Alphabetization of post-run dump
PLUGTB	2	Branch to INTR10	The functions ANY, NOTANY, SPAN, and BREAK
RCOMP	1	Branch to INTR10	Real arithmetic
REALST	1	Branch to UNDF	Real arithmetic
REWIND		Branch to INTR10	The function REWIND
RLINT	1	Branch to INTR10	Real arithmetic

RPLACE		Branch to INTR10	The function REPLACE
SBREAL	1	Branch to INTR10	Real arithmetic
SPREAL	1	Take the FAILURE exit	Real arithmetic
STPRNTB		Perform STPRNT instead	Double output for run statistics
TRAPCK		Perform no operation	Ability to interrupt execution
TRIMSP		Branch to INTR10	The function TRIM
UNLOAD	4	Perform no operation	External functions

Note 1: All operations relating to real arithmetic should be implemented or not implemented as a group.

Note 2: CLERTB and PLUGTB should be implemented or not implemented as a pair.

Note 3: LEXCMP must be properly implemented if LTLOC is the same as GTLOC. Note 4: LINK, LOAD, and UNLOAD should be implemented or not implemented as a group.

7.2 Machine-Dependent Data

In addition to the data given in the COPY files (q.v.) there are several format strings that generally have to be changed to suit a particular machine. The strings defined by FORMAT (which occur at the end of the source file) are in this category. The two strings CRDFSP and OUTPSP defined by STRING are also machine dependent. These are ignored in Oregon SNOBOL5.

7.3 Error Exits for Debugging

During the debugging phases, it is good programming practice to test for certain conditions that should not occur, but typically do if there is an error in the implementation. Stack underflow is typical. Transfer to the label INTR10 upon recognition of such an error causes the SNOBOL5 run to terminate with the message ERROR IN SNOBOL5 SYSTEM. Following this message, the statement number in which the error occurred is printed, as well as requested dumps and termination statistics that may be helpful in debugging.

7.4 Classification of Macro Operations

In the following sections, the macro operations are classified according to the way they are used.

Assembly Control Macros:

COPY DHERE END EQU EQUD LHERE PROC TITLE

Macros that Assemble Data:

ARRAY BUFFER DESCR FORMAT SPEC STRING

Branch Macros:

BRANCH BRANIC SELBRA

Comparison Macros:

ACOMP ACOMPC AEQL AEQLC AEQLIC
CHKVAL DEQL ICOMP ICOMPC IEQLC
LCOMP LEQLC LEXCMP RCOMP TESTAI
TESTF TESTFI VCMPIC VEQL VEQLC

Macros that Relate to Recursive Procedures and Stack Management:

GETSTD INCSP ISTACK MOVSTD PUTSTD
POP PROC PSTACK PUSH
RCALL RRTURN SPOP SPUSH

Macros that Move and Set Descriptors:

GETD GETDC MOVBLK MOVD MOVDIC MOVSTD
POP PUSH PUTD PUTDC ZERBLK

Macros that Modify Address Fields of Descriptors:

ADJUST BKSIZE DECRA DECRI GETAC GETLG
GETLTH GETSIZ INCRA INCRI MOVA PUTAC
RLINT SETAA SETAC SETAV SETAVO SPCINT

Macros that Modify Value Fields of Descriptors:

INCRV MOVV MOVV0 PUTVC SETAVO SETSIZ SETVA
SETVC

Macros that Modify Flag Fields of Descriptors:

RESETF RSETFI SETF SETFI

Macros that Perform Integer Arithmetic on Address Fields:

DECRI DIVIDE EXPINT INCRI MNSINT
MULT MULTC SUBTRT SUM

Macros that Perform Address Arithmetic on Address Fields:

DECRA INCRA MULTA

Macros that Deal with Real Numbers:

ADREAL ATAN ATAN2 COS DVREAL EXREAL INTRL
LOG LOG10 LOG2 MNREAL MPREAL R2HS R2HX
RCOMP REALST RLINT SBREAL SIN SPREAL SQRT
TAN

Macros that Move Specifiers:

GETSPC PUTSPC SETSP SPOP SPUSH

Macros that Operate on Specifiers:

ADDLG	APDSP	CENTER	FSHRTN	GETBAL	GETLG	GETLTH
INTSPC	LOCSP	LPAD	PUTLG	PUTSPC	PUTSTD	REALST
REMSP	REVERSE	RPAD	RPLACE	SETLC	SHORTN	SPCINT
SPREAL	STREAM	SUBSP	SUBSTR	TRIMSP		

Macros that Operate on Syntax Tables:

CLERTB	PLUGTB	STREAM

Macros that Construct Pattern & Tree Nodes and other structures:

ADDSIB	ADDSON	CPYPAT	INSERT	LINKOR	LOCAPT	LOCAPV
LVALUE	MAKNOD	TOP				

Macros for Input and Output:

BKSPCE	ENFILE	FILNAMI	FILNAMO	FINDUNIT	FORMAT	OUTPUT
REWIND	SEEK	STPRNT	STPRNTB	STRDNP	STREAD	

Macros that Depend on Operating System Facilities:

DATE	ENDEX	GETENV	INIT	LINK	LOAD
MSTIME	SYSTEM	TRAPCK	UNLOAD		

Macros that handle HEX and BIT strings:

ANDFUN	B2H	B2I	B2IS	B2R	B2S	BITI	
BITS	ENDIANB	ENDIANH	H2B	H2I	H2IS	H2R	H2S
HEXI	HEXS	HS2R	HX2R	LOBFUN	NAMDFUN	NORFUN	NOTFUN
ORFUN	R2HS	R2HX	XORFUN				

Miscellaneous Macros:

BLOCK	FUNC	ORDVST	RANDOM	RPLACE	SORT
SPCINT	TOP	VARID			

Optional macros for debugging: (See CRTMAIN.SNO for info about them)

DEBUG	DUMPAD	DUMPBLK	DUMPD	DUMPDA	DUMPREGS
DUMPS	DUMPTXT				

8. Oregon SNOBOL5 Source Files and Assembly Procedure

Prerequisites include a Windows and a Linux computer (Ubuntu is what I used). Install Microsoft Visual Studio Community 2022 and "add workloads". Link is:

https://visualstudio.microsoft.com/downloads/

Download and install "build tools":

https://visualstudio.microsoft.com/downloads/#build-tools-for-visual-studio-2022
https://aka.ms/vs/17/release/vs_BuildTools.exe

Click on "more" and install everything.

In a command prompt window, after executing "vcvars64.bat" which is usually found at:

C:\Program Files (x86)\Microsoft Visual Studio\2022\BuildTools\VC\Auxiliary\Build\vcvars64.bat

Check if you can execute ML64.EXE and LINK.EXE. If not, you have missed something above. On linux, make sure you have "objcopy". If not install "binutils". On Windows, you will need the working SNOBOL5.EXE available, otherwise some other version of SNOBOL could be used but might require some source file changes. This does work on Windows XP using the old Minnesota SNOBOL4. Place all of the source files below in a Windows directory and run the "CRTALL.CMD". In the CRTALL.CMD, there are scp and ssh commands to copy the executable to the Linux machine and run objcopy. You will have to modify those to use your network configurationa and id's. Otherwise you will have to copy the file using a USB stick or some other means and execute the objcopy command on the Linux machine.

8.1 The source files are:

CRTALL.CMD This is the batch file which creates the executable modules.

CRTMAIN.SNO This is a SNOBOL5 program which converts the SIL source macros into assembler code and data.

MAIN.A MAIN.B MAIN.C MAIN.D MAIN.E This is the SIL source for SNOBOL5. It is split into separate files for editing convenience and could be combined if appropriate adjustments are made to the programs involved:

RENUM.SNO Combines the MAIN.? files, adding sequence numbers to help with debugging and produces the file M.SIL.

PARTGEN.SNO Removes comments etc to make the input to CRTMAIN.SNO smaller in case one is processing this using the older Minnesota SNOBOL4. MAIN.SIL is the output.

OPTIMIZE.SNO This processed the code generated from CRTMAIN.SNO to enable storing the XPTR descriptor in registers, rather than in memory.

SNOBOL5.ASM This is the main assembler module for Oregon SNOBOL5. It includes most of the *.INC files and files generated by CRTMAIN.SNO. It defines various data items needed throughout and has the initialization code to get SNOBOL5 running.

ADDREQU.INC These are some addressing equ's and show how descriptors and specifiers are layed out in memory.

BLOCK.INC This is an assembler replacement for the SIL BLOCK routine to allocate blocks of storage.

DEBUG.INC This had code only useful when debugging is turned on in CRTMAIN.SNO. It is useful to find what code changes a memory location's value, for example.

DEBUG64.INC This is the debug package I used to debug SNOBOL5. It works on Windows and Linux. These code calls and macros can be inserted in the code as needed. The packages output functions are sometimes used for error mesages in SNOBOL5.

DUMPUNIT.INC Is used in debugging to show the content of the UNIT.INC structure for an I/O unit.

FILEATTR.INC This is some code using PATMAC.INC to parse the attribute flags for I/O.

FINDUNIT.INC This is code to find an unused I/O unit number.

FLTINIT.INC This is code to initialize everything needed for REAL (floating point) support.

FLTRTN.INC This code converts REAL numbers to STRINGs and vice versa. It includes a macro to check if a NAN (Not a Number) was created and change it to point to the SNOBOL source which created it.

GC.INC This is an assembler replacement for the SIL garbage collector.

GETENV.INC This is code to retrieve environment variable values.

GETSTACK.INC This code determines the size of the Linux stack.

INT2STR.INC This code converts an integer to a string.

IOCLOS.INC This code closes an I/O unit.

IOSEEK.INC This code implements the SEEK function.

LNXLOCTM.INC This is Linux code to get the local time.

LNXMHZ.INC This code finds an approximate speed of the CPU.

MYALLOC.INC This is an operating system independent memory allocator.

NQ8.SNO This the N-queens problem solving program as a simple sanity test case.

OPENREAD.INC This code opens an I/O unit for reading.

OPENWRIT.INC This code opens an I/O unit for writing.

ORDVST.INC This code sorts SNOBOL variables in alphabetic order for &DUMP = 1 or DUMP(1).

PATMAC.INC These are the assembler pattern matching macros.

RAND.INC This is a random number generator.

SILCODE.INC This contains various SIL macro functions that are more complicated and not as suitable as a simple macro expansion.

SILMACS.INC This contains a macro used by ACOMP.

SORT.INC This implements the SORT() function.

STPRNT.INC This implements the STPRNT SIL function.

STREAD.INC This implements the STREAD SIL function.

STREAM.INC This implements the STREAM SIL function.

UNIT.INC This declares the I/O unit structure.

UTILMAC.INC This contains macros for handling the stack when calling Windows functions. It also has macros for storing and loading the XPTR descriptor from/to registers.

VERSION.INC This gives the version ID for SNOBOL5. It should be updated when changes area made. During the alpha stage, this will not be updated. ####

addrmap.inc This is generated when debugging is turned on.

equs.inc This is generated by CRTMAIN.SNO.

intcod.inc This is generated by CRTMAIN.SNO and then optimized by OPTIMIZE.SNO.

intdta.inc This is generated by CRTMAIN.SNO.

map.s If gendebug is turned on in CRTMAIN.SNO, the output of a simple snobol5 run produces a file which has the address map. A windows version of the grep command is useful in extracting that and sorting it in file map.s. This is useful only during debugging.

All of the above source is available as a zip file in the link below. You should create a directory and unzip the file in that directory. Make SNOBOL5.EXE available in your path. Then run VCVAR64.BAT and CRTALL.CMD and you will build both Windows and Linux executables per the above instructions.

http://snobol5.org/s5source.zip

8.2 Acknowledgement

The SIL version of SNOBOL was implemented jointly by Ralph Griswold, Jim Poage, Ivan Polonsky. Dave Farber and possiblly others, initially at Bell Laboratories and later University of Arizona. Ralph Griswold is the author of "Implementing SNOBOL4 in SIL; Version 3.11", Technical Report S4D58, Department of Computer Science, University of Arizona. The bulk of the material in this manual was taken from that report. Phil Budne's, Mark Emmer's, and Robert Dewar's versions of SNOBOL were inspirations in various ways. Adaptations of the SIL implementation for Minnesota SNOBOL4 and Oregon SNOBOL5 were made by Viktors Berstis.

8.3 Additional Implementation Material

There is a substantial amount of additional material available to the would-be installer of the SIL implementation of SNOBOL5. Much of the basic documentation is given in a book that might be available through book suppliers. The rest of the material is available here at these links.

8.4 Version 3.11 SIL source code

Unfortunately, I no longer have the machine readable version of this unless someone gives me a working 9 track tape drive that can be used with a current computer system. Or sends me the file. However, here is a printed version of this old SIL code, including a cross reference. It might be useful when trying to understand the updated code. Unchanged code lines have the same sequence numbers.

http://snobol5.org/SILv3.11_19760609_S4D26b.pdf
Some corrections here:
http://snobol5.org/CorrectionsSIL3.11_19820426_s4n24.pdf

8.5 Description of Source code and SIL

The original and new version of this document:

http://snobol5.org/ImplementingS4inSilV3.11_198102_s4d58.pdf
This document for SNOBOL5:
http://snobol5.org/s5doc.pdf

8.6 "The Macro Implementation of SNOBOL4" by Ralph E. Griswold

Published by W. H. Freeman & Co. in 1972, this book is now hard to find. It describes SNOBOL4 data structures, algorithms, the SIL macros, and gives examples from the IBM 360 and CDC 6000 implementations. The terminology used in this book is slightly different from that used in the actual SIL source. For example, descriptors are illustrated in reverse from current implementations. It can be very useful in understanding the internal workings of SIL based SNOBOL4. There is a document of corrections to this book here:

http://snobol5.org/Corrections_TheMacroImplementaionOfS4_19870209.pdf

These documents have some of the basics that were covered in this book:

http://snobol5.org/S4_Internal_Structures_19741213_S4D26.pdf
http://snobol5.org/S4_StructureAndImplementation_SHARE37_19710812.pdf

You may be able to find the book here:

https://archive.org/details/macroimplementat0000gris/mode/2up?q=snobol

8.7 The SNOBOL4 Programming Language, second edition

Authors, R.E Griswold, J.F. Poage and I.P. Plonsky, published by Prentice Hall. This book covers the SNOBOL4 programming language in full. The book is out of print and also hard to find, but we do have the pdf version. I still have some unused "new" copies stored away, in case someone is desperate to buy one.

http://snobol5.org/greenbook.pdf

8.8 Other resources

Phil Budne keeps a web site with additional documents and resources about SNOBOL here:

http://www.regressive.org/snobol4/

8.9 Future plans and goals.

My goal for this project was to update SNOBOL with additional function I find useful while making it run on both Windows 7 and up and Linux as compatably as possible. I wanted to take full advantage of 64 bit addressing. The project was coded in assembler to gain extra performance. One of the goals is to use a minimum of external libraries (but I understand that this might not be possible for some future functions). This is still in alpha stage, but I thought I would make the source available now, just in case some unfortunate event would not let me finish this project. I am still making small changes and fixing small bugs, as I discover them. Any input from other users would be useful and appreciated.

My plans are to look at what it would take to implement dynamic loading and linking to external code. This may be a big project. Another goal is to be able to support some sort of windowing. Doing this compatably in both Windows and Linux looks like a challenge. There is something called Wayland that might be useful for this. If this is done, some additional graphics functions would be useful to implement. Other addiions include receiving and sending on internet ports. It looks like it might not be too hard to make an executable file from a SNOBOL5 program, so you would not have to download two things for someone to run it. It could be made to somewhat obfuscate the SNOBOL source code, but given that this is open source, it also would not be too hard to reverse engineer. In a further future, a version that runs on Android or Raspberry PI might be nice, again a large project and Budne's CSNOBOL4 would do for most users. Finally, I will make some tutorial videos for those learning to use SNOBOL5. Any further suggestions are welcome. Thanks!

- Viktors Berstis