7970

U.C.S.B. ONLINE SYSTEM MANUAL

This update documents the modifications to the UCSB Online System as of September 1, 1971. A correct list of pages is on the reverse of this page. Please report any errors to the online consultant, so that corrections may be included in subsequent updates to this manual.

The following list of page numbers indicates the correct order of the pages in the UCSB ONLINE SYSTEM MANUAL. Pages revised in the September 1, 1971 update are indicated by S.

i-vi	S
1-14	
15-16	S
17-18	
19-22	S
23-24	
25	S
25.1	S
25.2	S
26	S
27-36	
37-41	S
41.1	S
41.2	S
42	S
43-50	
51-60	S
60.1	S
61-62	S
63-74	
75-76	S
77-78	
79-80	S
81-82	
83-88	S
89-106	
107-111	S
111.1	S
111.2	S
112-116	S
116.1	S
117-118	S
118.1	S
119-120	S
120.1	S
121-122	S
123-132	
133-140	S
141-150	
151-164	S
164.1	S
164.2	S
164.3	S
164.4	S
164.5	S
164.6	S
165-168	S
169-170	
171-174	S
174.1	S
175-180	S
180.1	S
180.2	S
181-184	S
185-194	
195-212	S
213-261	
262-274	S

Corrections and new text documenting additions to the online system are indicated by a verticle bar in the left margin next to the changed text. Changes which improve grammar, readability, or notation are not marked.

Revised Sept. 1, 1971

U.C.S.B. ONLINE SYSTEM MANUAL

TABLE OF CONTENTS

PAGE NO.

TABLE OF CONTENTS CONTINUED

PAGE NO.

TABLE OF CONTENTS CONTINUED

PAGE NO.

TABLE OF CONTENTS CONTINUED

TABLE OF CONTENTS CONTINUED

PAGE NO.

Appendix A

Appendix B

Appendix C

Appendix D

Appendix E

Appendix F

Appendix G

This manual is written under the assumption the reader is using a new (NSF) keyboard. For users of old keyboards the following differences should be noted.

1. There is no SHFT key (not to be confused with SHIFT) on the lower keyboard, this means that the storage locations α-ω are not accessable.
2. There is no LVII key.
3. There is no PWR key.
4. There is no PROD key.
5. There is no SORT key. For MOLSF LII SORT, one may use MOLSF LV SQRT.
6. There is no CONV key. For MOLSF LII CONV, one may use MOLSF LV NEG.
7. A complete set of punctuation keys is not accessable. In particular "¬" and "=" are not accessable. This means that REPT (...) I=1,7 ... is not accessable. Any search on a not equal condition is not accessable.
8. For the LO key use LI INDEX or LI SHIFT.
9. For the CASE key use INDEX or SHIFT.
10. For the SEL key use TYPE DIFF.

2.1 SYST - SYSTEM LEVEL

2.1.1 ACCESS TO THE ONLINE SYSTEM - LOGIN

The SYST key notifies the online system that a user wishes to login. A new user may obtain a user number and an identification code (ID code) from the UCSB Computer Center Office. The ID code prevents unauthorized users from using your funds. An optional user name provides added qualification. The user number and user name, if selected, identify which program library is to be used, which facilities this user may access, whether a job name is required, and the funds remaining in this account. The procedure for obtaining a user number is outlined in Appendix A.

PROCEDURE FOR SYSTEM INITIATION

To login to the online system and explicitly load the mathematical language, a user would first make sure the equipment is turned on; then progress through the following sequence:

KEYBOARD ENTRY	OLS QUERY/RESPONSE
SYST	ENTER USER NUMBER (user number)
(user number) RETURN	ID NUMBER =
(ID number) RETURN	USER NAME = (user name)
(user name) RETURN	JOB NAME = (job name)
(job name) RETURN	AUTOSAVE CODE = (integer)
	LOAD MOLSF
MOLSF RETURN	FILE LOADED

If any of the above information is not required on your user number, the online system will not ask for it. If you make an invalid entry, the online system will repeat its query.

The autosave code given after user identification is completed allows one to restart after a system failure. The autosave number identifies a user workspace that is preserved after most system failures. The number should be remembered so that if the online system fails, the user may restart. In this example the user selected the language MOLSF. MOLSF is the name of a language and is reserved (one may not store a file with that name). When a language name is loaded at sign on the user is placed on that language with no data stored and no programs defined.

2.1.2 SIGNING OFF - LOGOUT

If, during your session at the console, you have generated any programs or data that you wish to save, you must store them in your user library before you sign off. Library operations are discussed in sections 2.1.5 to 2.1.8.

To sign off and terminate the billing process, the user must press SYST DOWN; otherwise the next user can accrue his costs to your account. Pressing the SYST key causes the message "WORK AREAS UPDATED" to be displayed to indicate to the user that he is on the SYST level. Pressing the DOWN key causes, after a pause, the message "WORK AREAS PURGED" to be displayed. This indicates that the billing process has been terminated and system facilities are no longer available. User workspace, identified by the autosave code, is also freed. This means that restart through the warmstart facilities is not possible.

Your scope should now be turned off.

the seven levels LI-LVII and may be REAL or COMPLEX data. A maximum of sixteen subfiles may be in a file: a USER subfile, a L0 data subfile, LI-LVII REAL and COMPLEX subfiles. If the language does not support data system storage of a certain level or a level is not defined, the system will not allow the user to store that type of subfile. The subfiles which may be stored in the permanent library are shown below.

PRESENTLY SUPPORTED SUBFILES

Keys pressed	Subfile loaded, stored, or deleted
	Basic System subfiles
USER	User programs and special characters. Both are contained in one subfile.
L0	L0 storage locations.
	COL subfiles
LI REAL	LI REAL strings.
LI CMPLX	LI COMPLEX strings.
LIII REAL	LIII REAL FILE.
LIII CMPLX	LIII COMPLEX file.
	MOLSF subfiles
LI REAL	LI REAL storage locations.
LI CMPLX	LI COMPLEX storage locations.
LII REAL	LII REAL storage locations.
LII CMPLX	LII COMPLEX storage locations.
LIII REAL	LIII REAL storage locations.
LIII CMPLX	LIII COMPLEX storage locations.

When loading from or storing into the permanent library, the user may specify that only selected parts of a subfile be transferred. He does this by supplying a predicate list immediately after the subfile specification. The following are valid predicate lists:

Keys pressed	Segment of Subfile Loaded or Stored
User Subfile Predicate Lists	
USER lvl RETURN	User level specified.
USER lvl$_1$-lvl$_2$ RETURN	User levels lvl$_1$ through lvl$_2$ inclusive.
USER lvl$_1$ lvl$_2$...RETURN	User levels specified.
USER CASE number RETURN	Case level specified.
USER CASE number$_1$-number$_2$ RETURN	Case levels number$_1$ through number$_2$ inclusive.
USER CASE number$_1$ number$_2$ RETURN	Case levels specified.
USER lvl-number RETURN	User levels starting at level specified to LVII inclusive and case 3 through the case specified.
Data Subfile Predicate Lists	
LO A RETURN	Variable A of level 0.
LO D-F RETURN	Variables D through F of level 0.
lvl (REAL or CMPLX) A RETURN	Variable A of data level specified.
lvl (REAL or CMPLX) A-H RETURN	Variables A through H of the data level specified.

A not sign ("¬") after the subfile when defining a segment of a subfile tells the online system to load/store everything except the segment or segments specified.

NOTE: In the LOAD operation, specifying a predicate list causes the selected segment of the subfile to be merged with whatever other parts of that subfile type had previously been loaded or created. In the STORE operation, the specified segment replaces the entire subfile in the permanent library.

2.1.5 STORING A SUBFILE IN YOUR USER LIBRARY

If a user has written user programs, created special characters, created data, or created a COL file, and he wishes to use them during a subsequent session on the online system, he must store his programs and data in the permanent library before he signs off. The procedure for storing a subfile is:

KEYBOARD ENTRY	OLS QUERY/RESPONSE
SYST	WORK AREAS UPDATED
STORE (subfile)[predicate list] RETURN	STORE (subfile)[predicate list]
(name) RETURN	FILE NAME = (name)
[protection code] RETURN	PROTECT CODE = [protection code]

The predicate list may be a partial subfile as listed on page 20. The STORE procedure must be repeated for each subfile, even though several subfiles may be stored under one file name. A file name is from one to twelve alphameric characters. Note again, if you store a partial subfile in the permanent library, it becomes the entire subfile.

The first time a user uses a file name, he may supply a protection code consisting of at most twelve alphameric symbols. Thereafter, whenever a subfile is stored under the same name, the system will ask for the protection code before storing the working copy over the old file. If the user wishes no protection code when he first stores his file, he need only press RETURN when the system requests the protect code. In subsequent stores the system will not ask for a protection code before it stores the file.

EXAMPLE: A user wishes to store his working user system for the first time under the name VENICE with the protection code GONDOLA. The storing sequence is as follows:

KEYBOARD ENTRY	OLS QUERY/RESPONSE
SYST	WORK AREAS UPDATED
STORE USER RETURN	STORE USER
VENICE RETURN	FILE NAME=VENICE
GONDOLA RETURN	PROTECT CODE=GONDOLA
	DONE

2.1.6 LOADING A FILE

You may load any file in your library, while you are signed on, by the procedure:

KEYBOARD ENTRY	OLS QUERY/RESPONSE
SYST	WORK AREAS UPDATED
LOAD (name) RETURN	LOAD (name)
	FILE LOADED

The above sequence is also valid during login; however, the SYST and LOAD keys are not pressed. The file name supplied may be the name of a previously stored file or the name of a language. The load operation is basically a concatenation process; only the subfiles defined in that file replace previously defined subfiles. When loading a partial subfile (i.e. LOAD USER LI-LIII RETURN) it is merged with the current subfile. When loading a whole subfile (LOAD USER RETURN) it replaces the current subfile, but does not affect other subfiles.

2.1.8 DISPLAYING YOUR USER LIBRARY

A list of your files in the permanent library may be displayed by pressing the DISPLAY key. Every time RETURN is pressed one file name is displayed, as long as there are files left. Each file is listed in the form:

```
filename language subfile-type
```

The subfile shown is the first subfile stored in that file. To see succeeding subfiles within that file press comma. No message is displayed to indicate that all subfile or files have been displayed. The example below shows a typical user library display:

```
TEST                     MOLSF LII C
                               USER
INTEGRAL                 MOLSF USER
                               LO
                               LII R
PUNCH                    COL   LIII C
PUNCH2                   COL   USER
                               LIII R
```

C and R are used to indicate complex and real subfiles, respectively. If there is more than one user name on your user number, the key sequence "DISPLAY USER name RETURN" followed by successive RETURN's displays the library for the user name that you specified. Your own library is displayed without specifying a user name.

2.1.9 OPERATOR DEFINITIONS FOR THE SYSTEM (SYST) LEVEL

DOWN

purges all temporary workspace, terminates the billing process, and signs the user off the online system.

SUM

displays online billing charges from login to the pressing of the SUM key.

REFL

interchanges the user levels specified. This operator applies only to the contents of the user levels in temporary storage.

SUB

replaces the second user level specified with a copy of the first user level specified. If any user programs exist on the second user level, they are deleted and may not be recovered. This operator applies only to the contents of the levels in temporary storage.

NEG

deletes the specified user level. This operator applies only to the contents of the user levels in temporary storage.

EVAL

displays the names of the operator keys for which user programs have been defined on the specified level. All categories of user programs are displayed: user programs, user CTX programs, and user DISPLAY programs. To display a list of all user programs not defined, press the not key "¬" before specifying a level. This operator applies only to the contents of the user levels in temporary storage.

MAX

displays name and text of all user programs which have been defined on the specified level. An erase is inserted between each display. This operator is designed for use with a printer or teletype (see sections 2.4.4 and 2.4.5). This operator applies only to the contents of the levels in temporary storage.

DEL

deletes entire file or specified subfile from the permanent library.

LOAD

loads an entire file, a specified subfile, or a specified portion of a subfile from the permanent library into temporary storage. The user is left on the language with which the file or subfile is associated. The file or subfile is merged with other subfiles already existing in temporary storage. When a portion of a subfile is loaded, it replaces only those variables or levels specified in the predicate list.

STORE

the subfile or portion of a subfile specified replaces the entire subfile under the same file name in the permanent library or creates a new subfile in the permanent library.

DISPLAY RETURN

displays the first file name in the permanent library. Subsequently

(1) RETURN displays the next file name.
(2) , displays additional subfile types, if any.
(3) USER asks for a user name; thus the user is able to display the file names in the permanent library of any other user on his user number.

2.2 THE TYPE LEVEL

On the TYPE level the lower keyboard keys function like those on a typewriter. The keys RETURN, BACK, and SPACE provide "carriage" control just as on a typewriter. The operator keys UP and DOWN move the typing position up one line and down one line respectively. The RS operator positions the display to the upper lefthand corner of the output device. On the TYPE level, the CASE key operates like a typewriter shift key, changing to a different set of characters. In CASE 1, which is normally used, the lower keyboard keys type the symbols appearing on their faces. By pressign either CASE 2 or holding down the SHFT key the alphabetic keys type the Greek alphabet, the numeric keys type superscripts, and the punctuation keys type other symbols, such as "=", as shown in Table 2.2.1. The remaining shift levels, CASE 3 through 9, are used for message and symbol generation.

Once a CASE level has been specified the online system remains on that level until a new level is defined. Pressing TYPE, DISPLAY, CASE 1 or CASE puts the user on CASE 1 for resumption of normal typing.

The TYPE level enables the user to include messages in a user program. Suppose, for example, that the user wishes to evaluate A+B and to display the message "A+B=" followed by the sum. He would first press TYPE to indicate that this part of the program is to be typed out as a message, rather than interpreted as an operator or operand. Every key pressed on the lower keyboard, between TYPE and the next level key, is

BACK

removes the last point stored in the character vector list and repositions the dot on the scope to the preceeding location. In effect, this erases the last direction keypush. To see the change, however, the user must erase the scope and press DISPLAY RETURN.

2.3 LEVEL 0 - L0 - INDEX LEVEL

L0 is a level for integer arithmetic. The major purpose of this level is to allow the user to vary operand values where integers would normally be entered. There are two level 0 working registers, the remainder and the quotient registers. Most operators operate only on the quotient register; however, ⊙, ⊘, INV, and REFL modify the remainder register as well as the quotient register.

2.3.1 LEVEL 0 OPERAND FORMS

(N = an unsigned integer; S = a level 0 storage location referenced by an alphabetic key; A = a level 0 storage location or an unsigned integer).

S — uses the contents of S.

-S — negates, then uses the contents of S.

S+ — uses the contents of S, and then increments the contents of S by one and stores the result in S.

S+A — uses the contents of S and then increments the contents of S by A and stores the result in S.

S- — uses the contents of S and then decrements the contents of S by one and stores the result in S.

S-A — uses the contents of S and then decrements the contents of S by A and stores the result in S.

N+A — uses the value of N times 10 to the Ath power; i.e., 3 + 2 equals 3×10^2 or 300; 3 + A equals 3×10^A.

N — uses the positive integer N.

-N — uses the negative integer N.

2.3.2 OPERATOR DEFINITIONS FOR LEVEL 0

(L = any level 0 operand; S = any level 0 storage location)

⊕ L, ⊖ L — performs the indicated operation on the contents of the quotient register.

⊙ L — multiplies the contents of the quotient register by the operand. The low-order thirty-two bits are placed in the quotient register, and the high-order thirty-two bits are placed in the remainder register.

⊘ L — divides the contents of the quotient register by L. The quotient is placed in the quotient register. The remainder is placed in the remainder register. Division by zero results in the error message "FIXED POINT DIVIDE CHECK".

REFL — interchanges the remainder and quotient registers.

SUM — loads the billing charge (in pennies) since login into the quotient register.

LOAD L — loads the operand into the quotient register.

STORE S — places the contents of the quotient register in storage location S.

DISPLAY S — displays the contents of storage location S.

DISPLAY RETURN — displays the contents of the quotient register.

SUB,EVAL — language dependent (see sections 3.10 for COL and 4.10 for MOLSF)

The following operators accept trailing predicates. That is, if the next key(s) is any level 0 operand, then that operand will be used for the computation. If the next key(s) is not a level 0 operand, then the operand is the contents of the quotient register.

SQ squares the operand.

NEG negates the operand.

INV inverts the operand leaving the remainder in the remainder register.

MOD computes the absolute value of the operand.

DEL places a one in the quotient register if the operand was zero; otherwise the register is set to zero.

2.4 SPECIAL OPERATORS

2.4.1 RESET

RESET is a special operator which is available at all times. RESET purges all keys which have not been processed, copies user workspace in main storage to auxiliary storage to allow warmstart, displays the message "RESET COMPLETED" to signal successful completion of the reset operation, and, finally, places the user on the TYPE level. RESET is especially useful when a program is in an unintentional loop.

2.4.2 ERASE

The ERASE special operator erases the display screen on graphical output devices. ERASE does not affect the current operation and it works on all levels, in all modes except LIST.

2.4.3 REPEAT - REPT

The REPT key allows one to repeat nearly any sequence of keys. A single key, which is not a special operator, may be repeated by the sequence "REPT key LO operand". A series of keys, including special operators, may be repeated with the sequence "REPT (keys) LO operand". In both forms the level 0 operand specifies the number of repetitions, and is evaluated before the key or keys are executed.

EXAMPLE:

TYPE REPT (ABC) 5 RETURN

This series of keys will type "ABC" five times.

The level 0 operand may be replaced by another operand of the form "A=I,J,K RETURN", where A is a level 0 storage location and I, J, and K are level 0 operands. Before any keys are processed storage location A is set to I and the terminating conditions are checked. The terminating condition used depends on the value of K. Given K is greater than or equal to zero, the key sequence is executed if A is less than or equal to J. Given K is less than zero, the key sequence is executed if A is greater than or equal to J. After the key sequence is executed, A is incremented by K (or one if K is omitted), and the termination conditions are checked prior to repeating the key sequence.

EXAMPLE:

REPT (LO DISPLAY A) A=1,7,2 RETURN

The numbers 1, 3, 5, and 7 will be displayed.

2.4.4 SELECT - SEL

When a user first logs on the online system, he has one primary output device, normally a display scope. SEL allows a user to send displays to an alternate output device or to multiple output devices. Select must be followed by an integer constant. If the hardware is available, SEL 1 RETURN   selects the user's primary output device.

SEL 2 RETURN   selects a plotter.

SEL 3 RETURN   selects a teletype.

SEL 4 RETURN   selects the hardcopy output data set. Output in this data set may be punched on cards or printed on the line printer. (see section 2.4.5)

SEL 22 RETURN   selects output device number 22. (Assumes prior arrangements have been made which enables the user number to select other online stations.)

SEL 1,3 RETURN   selects multiple devices 1 and 3, the user's primary output device and a teletype.

The use of SEL is restricted. Devices 1 and 4 are always available. If a plotter or teletype is available, device 2 or 3 may be selected. SEL for any other device number or multiple devices requires prior arrangements with online system personnel.

If a user selects an invalid device number, his output device selection does not change. If the user selects multiple output devices and any of the device numbers are invalid, the entire list will be ignored. An invalid device number does not return a user to his primary output device; it leaves him on any device(s) he has already successfully selected.

2.4.5 HARDCOPY

After a user has sent output to the hardcopy output data set, he must submit a batch job (either through the card reader or through remote job entry; see section 3.5.2) that prints or punches the data in that data set. The user must run this job before midnight on the day he selected device 4 or his data will be lost. The batch job should consist of the following cards:

```
     //HARDCOPY JOB (ACCT,USERNAME),'output box'
     // EXEC HARDCOPY,OPT=option,LSIZE=number,USER=number,
     // NAME=name,LOGIN='hh:mm'
     //
```

The JOB card is standard and is described in the user's guide.

The EXEC card parameters are as follows:

1. "option" may be either PRINT, PUNCH or BOX.

   OPT=PRINT causes your output to be printed single space using the line size specified with the LSIZE= parameter.

   OPT=PUNCH causes your output to be punched on cards and printed with LSIZE=80.

   OPT=BOX causes your output to be printed in "boxes", 25 characters high, 12 boxes to a printed page. (Output printed under this option will include a "0" at the lower left boundary of any box in which an overlay has occurred).

   This parameter is optional and will default to PRINT if omitted.

2. LSIZE= the number of characters printed on each line of output; this number may range from 1 to 132. The LSIZE= parameter is optional and will default to 120 for the printer and 80 for the card punch if omitted.

3. USER= the online system user number on which you were running when you generated the output. This parameter is required.

4. NAME= the online system user name (exactly as entered at sign-on) on which you were running when you generated the output. This parameter may be omitted only if your user number does not have associated user names; it must be omitted in this case. If your name contains commas, spaces, or other special characters, it must be enclosed in apostrophes.

5. LOGIN= time of day in hours and minutes (24-hour clock) enclosed in apostrophes. This parameter allows you to specify that only output generated at a session starting later than the time coded is to be processed. Since the output data set is scratched only at midnight, you will get all of the output generated since midnight everytime you run HARDCOPY unless you code the LOGIN= parameter. Note that the only time being considered is the time that you signed on for a particular session. This parameter is optional and will default to '00:00' if omitted.

2.5 USER PROGRAMS - LIST MODE

Typically, the online system user interacts manually with the primary operators defined by the online system. However, once a user has found a key sequence that solves all or part of his particular problem, he would like to make this key sequence a subroutine which becomes part of the online system. Such subroutines are called USER programs. USER programs are created by using LIST mode, stored or modified on the EDIT level, and accessed (executed or recalled for modification) by the USER key. A collection of USER programs is called a USER system. USER systems may be stored permanently as described in section 2.1.5 on storing subfiles. The special LIST mode operators TEST, PRED, and ENTER control program flow and enter data or key sequences into USER programs.

2.5.1 STRUCTURE OF THE USER SYSTEM

The USER system has eight levels which are accessed by the USER key. The levels are designated as USER LO, USER LI, ..., USER LVII. The thirty-one operator keyboard keys are available on each USER level as storage locations for USER programs. Thus the key sequences USER LI SIN, USER LVII SORT, etc., each identify the storage location of one USER program. Additional storage is provided by using the CTX key preceding the operator key. For example, the key sequences USER LI CTX SIN, USER LVII CTX SORT, etc., identify the storage locations of unique USER programs. A maximum of (8 levels) x (31 operator keys) x (2) = 496 USER programs that can be stored on any one USER system. This maximum pointer remains unchanged until the user goes to EDIT level and manually moves it. NOTE: the ENTER key is the only way to return to LIST mode and has this effect only on the EDIT level. Pressing the LIST key at this point would destroy the current program.

The change to LIST mode is indicated by blotting out the post list marker. Once this has been done the user may type the new keys to be inserted. The keys appear on the display scope at the end of the program, but are inserted before the edit pointer. After the new keys are entered, the user presses LIST to return to the EDIT level. He may store his program or reposition the edit pointer and modify another part of his program.

The user may verify that the keys were properly inserted by pressing:

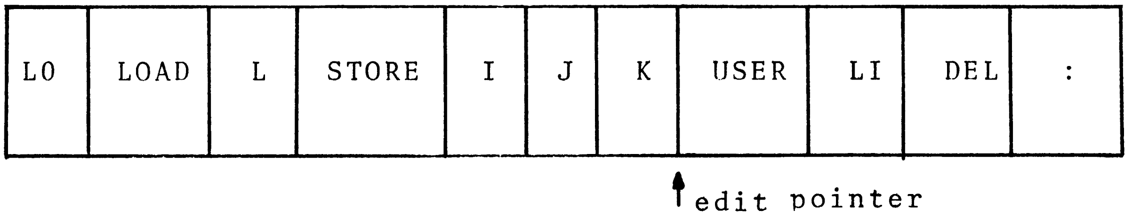
DISPLAY RETURN or

ID DISPLAY RETURN

EXAMPLE: Using the USER program of Section 2.6.4, enter the key "K" between the "J" and the "USER" keys. The editing procedure is:

MOD USER	positions the edit pointer between J and USER.
ENTER	enters LIST mode, blots out the post list marker.
K	inserts the key into the program.
LIST	changes to the EDIT level.

The user is now ready to store his program or reposition the edit pointer and modify some other part of his program. The internal list of the program is:



LO	LOAD	L	STORE	I	J	K	USER	LI	DEL	:

Note that the edit pointer's position has not changed.

EXAMPLE: suppose the program

LII REAL ID SQ ⊙ -0.5 EXP DISPLAY RETURN

has been incorrectly keyed in as

LII SQ ⊙ 0.5 EXP DISPLAY RETURN

The editing procedure, as soon as the LIST key has been pushed, is as follows:

MOD SQ — locates the editing point and displays the pointer between LII and SQ by underscoring SQ.

ENTER — changes to LIST mode, blots out ":".

REAL ID — inserts these keypushes before SQ, displays them at end of program.

LIST — changes back to EDIT mode, displays ":".

DISPLAY RETURN (optional) — displays the keypushes in proper sequence and shows that the insertion has been made.

⊕ 2 RETURN — moves the edit pointer two places toward the end of the program.

EVAL (optional) — displays the pointer between "⊙" and "0" by underlining "0".

ENTER - LIST — inserts "-" at the new editing point, displays it at the end and displays ":".

STORE USER LI ⊕	stores corrected program, displays "LI ⊕ UPDATED".
USER LI DISPLAY ⊕	displays correct program, without editing marks.

2.6.6 DELETION OF KEYS FROM A USER PROGRAM

The SPACE key and the BACK key are used on the EDIT level, to delete keypushes to the right or left, respectively, of the edit pointer. Either key, followed by an integer n, followed by RETURN, deletes n successive keypushes. If no integer is given, one keypush is deleted for each depression of SPACE or BACK. Each deleted key is blotted out on the output device.

In the example above, suppose the user had keyed in

LII REAL CMPLX ID SQ ⊘ -5.0 EXP DISPLAY RETURN

He could correct it as follows:

MOD ID	locates the editing point and displays the pointer between CMPLX and ID by underscoring ID.
BACK	deletes CMPLX.
⊕ 2 RETURN	locates the edit pointer between SQ and ⊘ .
EVAL (optional)	displays the location of the edit pointer by underscoring ⊘ .
SPACE 5 RETURN	deletes 5 keypushes ⊘ , -, 5, ., 0, and blots them out on the output device.
ENTER ⊙ -0.5 LIST	inserts correct keypushes, displays ":".
DISPLAY RETURN (optional)	displays program in proper sequence.

STORE USER I ⊕ stores program, displays "LI ⊕ UPDATED".

USER I DISPLAY ⊕ (optional) displays program.

If the user wishes to delete all keypushes on one side of the editing point, he may push DEL RS for the right side, DEL LS for the left side. These operations do not produce any visual (scratching out) effect.

2.6.7 BLOCK KEY SEQUENCE EDITING

As the preceding text explains, the edit pointer locates that position in the USER program where keys are to be inserted or deleted. As well as locating this editing point, the edit pointer divides the internal key list into two parts: that portion to the left of the pointer, i.e. from the beginning of the program to but not including the edit pointer; and that portion to the right of the pointer, i.e., from the edit pointer to the end of the program. By appropriately positioning the edit pointer the user can manipulate blocks of keys from the current program or a previously stored program. As a mnemonic aid, that portion of the program preceding the edit pointer is called the left side (LS), and that portion of the program following the edit pointer is called the right side, (RS). By appropriately manipulating the edit pointer, and loading and storing the LS or RS of the program, a long program can be rearranged. This is best illustrated by the examples which follow.

EXAMPLE: Block transfer

Correct Program:

LII CMPLX LOAD A SIN STORE C LOAD B LOG ⊕ C

Incorrect Version:

LII CMPLX LOAD B LOG LOAD A SIN STORE C ⊕ C

The editor's objective in this problem is to transfer LOAD A SIN STORE C to its correct position between CMPLX and LOAD B.

Assume the incorrect program has been stored under USER LII MAX. The user presses USER LII DISPLAY MAX. The incorrect program appears on the output device, and the online system console enters EDIT level. The block transfer is achieved by the following set of instructions:

MOD ⊕	pointer placed to left of ⊕ .
STORE RS	⊕ C stored temporarily.
MOD LOAD A	pointer placed to the right of LOG.
LOAD RS	⊕ C inserted after LOG.
STORE RS	LOAD A SIN STORE C is stored temporarily.
MOD LOAD	pointer placed to right of CMPLX.
LOAD RS	LOAD A SIN STORE C inserted after CMPLX.
DISPLAY RETURN	displays program in proper sequence.

EXAMPLE: Block Deletion

Correct Program:

LII REAL LOAD F MAX

Incorrect Version:

LII REAL ID ⊙ A ⊖ B SQ STORE C LOAD F MAX

The editor's aim is to remove ID ⊙ A ⊖ B SQ STORE C.

Method 1:	MOD ID	pointer placed to left of ID.
	SPACE 8 RETURN	deletes 8 keys to right of pointer.
Method 2:	MOD LOAD	pointer to left of LOAD.
	STORE RS	LOAD F MAX stored temporarily.
	MOD ID	pointer to left of ID.
	DEL RS	deletes everything to right of pointer.
	LOAD RS	appends LOAD F MAX to LII REAL

Observe that Method 1 requires knowledge of the exact number of keys to be erased, but Method 2 does not.

2.6.8 OPERATOR DEFINITIONS FOR THE EDIT LEVEL

MOD

allows the user to specify the edit pointer location. The user identifies the key he wants to appear at the right of the pointer by typing it after pressing MOD. If that key appears but once in the program, typing it is sufficient identification, and the key will be underscored. If that key appears more than once, then succeeding keys must be pressed until identification of the pointer location is uniquely determined. One key will be underscored when the position is uniquely fixed. If a non-existent sequence is pressed after MOD, the diagnostic "NON-EXISTENT STRING" is displayed. The search may be aborted by pressing LIST before a unique key sequence has been designated.

BACK

deletes the key preceding the edit pointer.

SPACE — deletes the key following the edit pointer.

BACK n RETURN
SPACE n RETURN — repeats the respective operation n times in succession, n an integer.

ENTER — puts the OLS console in LIST mode. Any button (except LIST or RESET) hit after ENTER is inserted into the program at the left of the edit pointer. If LIST is pressed after ENTER, the console changes to the EDIT level.

⊕ — shifts the pointer one key to the right.

⊖ — shifts the pointer one key to the left.

⊕ n RETURN
⊖ n RETURN — repeats the respective operation n times in succession, n an integer.

EVAL — displays the location of the edit pointer on the output device.

ERASE — erases the output device.

ID — erases the output device, moves carriage to upper left-hand corner of the output device.

REFL or
UP or
ENL — moves the edit pointer to the head of the program, and underscores the first key.

CON or
DOWN — moves the edit pointer to the end of the program and underscores the post list marker.

DEL RS — deletes everything to the right of the edit pointer.

DEL LS — deletes everything to the left of the edit pointer.

DEL RETURN — deletes left side and right side.

DEL USER (level) (operator)
deletes specified user program.

STORE RS
stores everything to the right of the edit program in a temporary location called the right side save area and removes it from the list buffer.

STORE LS
stores everything to the left of the edit pointer in a temporary location called the left side save area and removes it from the list buffer.

STORE RETURN
stores left side and right side and removes the entire program from the list buffer.

STORE USER (level) (operator)
stores contents of list buffer in specified storage location.

SUB
restores a user program in the same location. An attempt to store a new program by pressing SUB results in the diagnostic "PROGRAM HAS NO SOURCE".

LOAD RS
inserts the keys stored in the right side save area into the program at the left of the edit pointer.

LOAD LS
inserts the keys stored in the left side save area into the program at the left of the edit pointer.

LOAD USER (level) (operator)
inserts the specified user program into the program at the left of the edit pointer.

DISPLAY RS
displays everything to the right of the edit pointer.

DISPLAY LS
displays everything to the left of the edit pointer.

DISPLAY RETURN
displays entire program in proper sequence.

DISPLAY USER (level) (operator)
Same as LOAD except program is also displayed.

2.7 SPECIAL LIST MODE OPERATORS

2.7.1 THE ENTER KEY

The ENTER key allows the user to halt a program to enter data or execute other manual operations. When an ENTER instruction is encountered in a user program, the program is stopped and the OLS console is returned to the Manual mode. The user can then perform any basic operations he wishes. When he is through with his manual operations, he presses the ENTER key, which signals the online system to resume executing the USER program where it left off. ENTER can be used to enter data into a program or to check a recursive program each time before it cycles. In the latter case the instruction serves effectively as a program stop or halt command.

In the following example for computing X^n for positive X, the ENTER instruction allows the user to insert the value of n.

```
LIST
LII REAL LOAD X LOG ⊙
ENTER EXP DISPLAY RETURN
LIST
STORE USER LII SIN
```

When the program is run, it will put the OLS console into Manual mode at the point in the program where ENTER is located. Now the user may type a number if n is to be a constant, or an alphabetic key if the exponent is a stored vector. In any case, as soon as he presses ENTER, execution of the program will be resumed, and $e^{n \ln X} = X^n$ will be computed and displayed.

It is usually advisable, especially in a longer problem, to include in the program some visual indication that the ENTER point is about to be reached. (NOTE: any keys that are pushed while the program is executing will be queued until the program halt is executed, then they will be executed). This timing indication can often be combined with a display of a parameter value, which is desirable for checking one's typing and for identifying a graph. The example above could thus be programmed:

```
LIST
TYPE RETURN
WHAT SPACE N?
LII REAL LOAD ENTER DISPLAY 1 RETURN
⊙(LOG X) EXP DISPLAY RETURN
LIST
STORE USER LII COS
```

2.7.2 THE TEST KEY

The TEST operator gives the user branching capability within a program. Except for level 0, the number being tested, henceforth denoted by N_T, is dependent upon the current language and level as follows:

Level	Number Tested (N_T)
LO	contents of quotient register
MOLSF	
LI REAL	contents of β_I working register
LI CMPLX	contents of α_I working register.
LII REAL	contents of first component of β_{II} working register.
LII CMPLX	contents of first component of α_{II} working register.
LIII REAL	contents of component (1,1) of β_{III} working register.
LIII CMPLX	contents of component (1,1) of α_{III} working register.
COL	
LI	result of last LI EVAL operation.
LII	value of the record count minus value of the active file marker.
LIII	value of the active file marker.

As a memory aid, note that on MOLSF the working register tested is always the real part of a number.

N_T is tested for the three conditions positive, negative, and zero, either separately or in combination. In using TEST, the user may specify that if a certain condition is satisfied one of the following events will occur:

1) Execute the prescribed list of keypushes.

2) Clear the execution list of all pending keypushes and execute the following sequence.

3) Suppress execution of a series of keypushes until a specified subsequence occurs.

4) Skip the number of keypushes specified by the following integer or level 0 variable.

The several branching possibilities described above are discussed in the ensuing sections. For purposes of clarification alphabetic letters will be employed to indicate a sequence of button pushes. Thus A might imply the sequence USER LI SQ, B the sequence TYPE ERROR RETURN, etc.

BASIC TEST FORMAT

The use of the TEST operator in its basic form allows a program to branch to one of several other programs or sequences depending on whether the TEST parameter N_T is positive, negative, or zero. The basic format of TEST to accomplish this branching capability is

TEST + (A) - (B) 0 (C) D

Note that all conditional sequences are enclosed in parentheses and are preceded by the condition (lower keyboard +, -, or 0) against which N_T is to be tested. If a sequence is not enclosed in parentheses, it will be executed unconditionally. For the example shown above the following branches occur:

1) If $N_T > 0$, execute A then D.

2) If $N_T < 0$, execute B then D.

3) If $N_T = 0$, execute C then D.

The branching facilities are probably best understood by considering the flow chart or state diagram of Figure 2.7.1

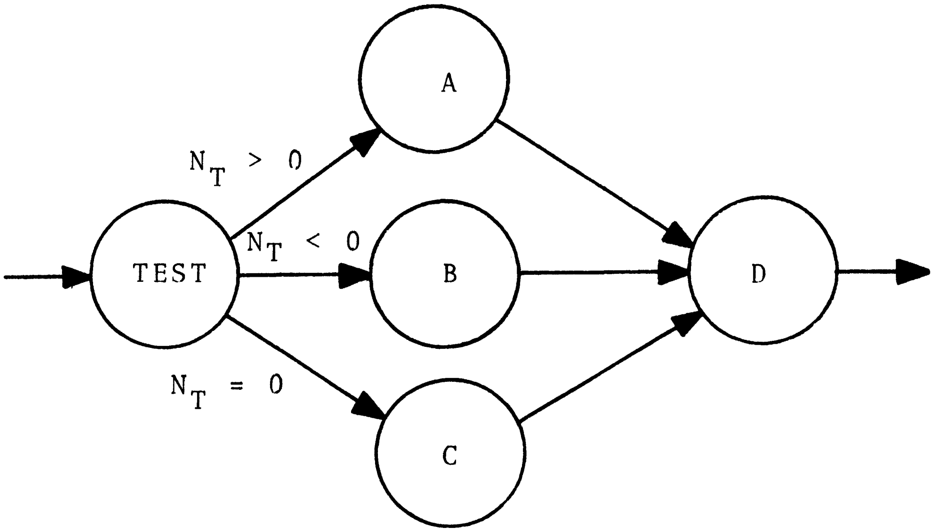


Figure 2.7.1 Flow chart branching for TEST program: TEST + (A) - (B) 0 (C) D

As indicated in the figure the basic format provides a three-way branch for the program depending on the test condition. The key-the online system executes the name program for the following key if that key is an operator. If a name program exists it will be displayed on the output device; if not, the operator will be displayed.

EXAMPLE: Assume the following name programs have been stored: USER LI DISPLAY LS is "START USER LI" and USER LI DISPLAY RS is "SEARCH". Then pressing the following keys LIST USER LI LS RS REFL will cause the following display:

```
START LIST
   USER LI START SEARCH
   REFL
```

CARD ORIENTED LANGUAGE (COL)

COL is a string processing language for the creation and manipulation of character strings, records and files. COL's capabilities enable one to:

1. Write a computer program in any language supported by the Computer Center (i.e., FORTRAN, PL/1, COBOL, ALGOL, SNOBOL, RPG, ASSEMBLER, etc.).
2. Create a data file.
3. Modify a program or data file.
4. Submit a program and data to the operating system as a batch job and access its output.
5. Access a data set from the operating system and create a COL file from it.
6. Scan files for particular characters or character strings.
7. Translate strings.
8. Create, concatenate, convert, search, compare, and save strings and substrings.
9. Convert numerical character strings to integers.

NOTE: Someone reading this chapter for the first time is advised to skip the sections on level I as they require knowledge of the operators on levels II and III.

When a user first signs on COL the record length is set to 80 characters. The user changes the current record length on level I, II, or III by pressing:

CTX N RETURN

where N is an integer between 1 and 254, or a level 0 operand. The current record length is displayed on level I, II, or III by pressing:

DISPLAY CTX

The length of the character string buffer varies; however, it may not exceed the maximum declared record length.

ACCESSING COL

COL is loaded in the same manner as any other language (see Section 2.1.6). If the user is working on any other language, loading a previously stored COL file switches him to COL.

3.2 LEVEL I - A STRING MANIPULATION LEVEL

Level I is a character string manipulation level. Its operators manipulate a variable length string buffer. Level I and level II share the same work area, the active buffer; however, to avoid confusion when we are discussing level I, the active buffer will be called the string buffer and its contents the active string. As discussed above, the maximum length of the active string is equal to the maximum record length currently declared on level II. The shortest string is the null string. On entry to level I, the length of the active string is not changed. In particular, if the user has added to or changed the active buffer while on level II and wishes this to become the active string, he must explicitly recompute the length of the active string. This is done by the DEL operator. DEL also deletes trailing blanks.

Temporary data storage on level I is provided by alphabetic storage locations: REAL A-Z, α-ω, and CMPLX A-Z, α-ω. Each one of the storage locations is initially set to the null string. When a string is stored both its length and contents are retained.

3.2.1 LEVEL I OPERAND FORMS

Level I operand forms can be grouped into three categories depending upon the source or destination of the string. A literal operand is one created by the lower keyboard keys. It is defined by an apostrophe, followed by a character string followed by RETURN or any operator key.

The seventh, eighth, and ninth characters from the string stored in storage location "A" are entered into the string buffer. The contents of "A" are not altered.

LOAD LII 77,N,5 RETURN

The Nth thru N plus fourth characters from the seventy-seventh record in the active file are loaded into the string buffer. The contents of the active file are not altered.

LOAD RETURN

The string buffer is set to the null string.

DISPLAY RETURN

displays the active string. It does not change the active string.

LOAD also has a fourth operand form. A period followed by 0-9, A,B,C,D,E,F allows one to load a hexadecimal number into the string buffer.

The STORE key is the converse of LOAD. It transfers the contents of the string buffer to the indicated storage location. The string buffer is not altered. The previous contents of the specified storage location are lost. STORE can be followed by an alphabetic or an interlevel operand, but may not specify a substring location.

EXAMPLE:

STORE LII M RETURN

The contents of the string buffer replace the Mth record of the active file.

3.2.3 SUBSTRING MANIPULATION

The ⊕ and ⊖ operators enable one to concatenate strings. ⊕ followed by any level I operand concatenates the operand at the end of the string buffer. ⊖ followed by any level I operand inserts the entire operand at the start of the string buffer.

The SUB and ⊘ operators enable one to keep or delete any substring of the active string. SUB preserves the specified substring, i.e. the specified substring is all that remains in the string buffer. The ⊘ operator deletes the specified substring from the string buffer. For a complete list of SUB and ⊘ operand forms, see the summary at the end of this chapter.

3.2.4 SEARCHES AND COMPARISONS

RS followed by any level I operand starts with the first character of the string buffer and searches right for the specified operand. LS followed by any level I operand starts at the end of the string buffer and searches left for the specified operand. Both LS and RS assume a second operand that tells which occurrence of the string one is searching for. If no occurrence operand is specified, the online system assumes the user is searching for the first occurrence of the string. If no match is found, then the level I search pointer is set to zero. If a match is found, then the level I search pointer is set to the value of the position where the character string was found.

MOD followed by successive characters searches the string buffer from left to right until (a) enough characters are given to define a unique matching character string or (b) the succession of characters is terminated and an occurrence number is specified. The level I search pointer is set to the position of the first character of the matching unique string or the specified occurence, and displayed. The diagnostic "NO MATCH" or "NO SUCH OCCURRENCE" is displayed if either of these conditions exist; in either case the level I search pointer is set to zero.

Comparisons are made with the EVAL operator. EVAL may be followed by any level I operand. The string buffer is compared with the operand. The results of the comparison are returned as an integer which is accessed by: LO EVAL + If the active string and operand have identical contents and length, then the integer returned is zero. If the active string and the operand are not equal, then the integer returned depends on the IBM 360 collating sequence, which is: ¢•<(+|&!$*);¬-/,%_>?:#@'="αβχδε ΠγθιΣκλμηοπφρστυνωξψζ AB...Z1...9 (It can be viewed online by pressing: ID DISPLAY RETURN). The EVAL operator compares the string buffer and operand character by character from left to right. The comparison proceeds until non-matching characters are encountered or one of the strings is exhausted. If unmatching characters terminate the comparison and if the character in the active string occurs in the collating sequence before the character in the operand, then the integer is set to minus one. It is set to plus one if the opposite occurs. If unequal lengths terminate the comparison and if the active string is shorter than the operand, then the integer is set to minus one. If the operand is shorter, then the integer is set to plus one.

3.2.5 TRANSLATING STRINGS

Level I has two operators for translating strings. The first translates all occurrences of one operand to another operand. The second operator translates individual characters. The latter works much like a translate table.

The SIN operator followed by two operands will search for all occurrences of the first operand. Whenever it finds the first operand it will replace it with the second operand. The operands may be of different length. The format of the SIN operator is:

SIN operand RETURN operand RETURN

EXAMPLE: Assume the following sentence is in the string buffer: "TODAY THE DAY RATE IS $5.00." The following sequence will change "DAY" to "NIGHT":

SIN 'DAY RETURN 'NIGHT RETURN

The sentence would now read: "TONIGHT THE NIGHT RATE IS $5.00." The following sequence could be used to delete the word "NIGHT":

SIN 'SPACE NIGHT RETURN RETURN

The string buffer would now contain "TONIGHT THE RATE IS $5.00." The null string is a valid second operand.

The COS operator followed by two operands will translate each character of the first operand to the corresponding character of the second operand.

EXAMPLE: Suppose the following sentence is in the string buffer: "IN 1946, CHARLES I WAS BEHEADED: IN 1649, GOERING

SHOULD HAVE BEEN." The following key sequence will correct the active string.

COS '96 RETURN '69 RETURN

The string buffer now properly reads: "IN 1649, CHARLES I WAS BEHEADED, IN 1946 GOERING SHOULD HAVE BEEN."

3.3 LEVEL II - A RECORD MANIPULATION LEVEL

The operators on level II enable the user to create and modify records, and to store them in the active file, thus creating a COL file. Level II simulates a keypunch, but has capabilities a keypunch cannot provide.

3.3.1 RECORD CREATION

As indicated above, records are created in a software work area called the active buffer. The length of the active buffer is declared with the key sequence:

CTX N RETURN

where N is an integer between 1 and 254, or a level 0 operand. The default value is 80. The current buffer length is displayed with the key sequence:

DISPLAY CTX

A record is created by entering lower keyboard keypushes. Each key is displayed as it is entered and stored in the active buffer. Depressing the CASE key signals the system to interpret the next keypush (and only the next key) as upper case, i.e., alphabetic keys are interpreted as Greek letters and numeric keys as special characters.

3.3.2 RECORD MODIFICATION AND MANIPULATION OF POINTERS

Associated with the active buffer is the active buffer pointer. The value of the active buffer pointer determines the position job. The member will be printed and the printed output put in the output box specified on the user's JOB card.

EXAMPLE: Print the output from the preceding JOB called "JOBNAME". The user would first create a three card JOB.

```
//PRINT JOB   (ACCT,USERNAME),'GEOLOGY'
//STEP1 EXEC  PRJEOUT,NAME=JOBNAME
//
```

Next he would execute it by pressing:

LIV SUB RETURN

He would then pick up his output from the geology output bins in the Computer Center.

3.5.4 DISPLAYING THE STATUS OF SYSTEM DEVICES

The user can see the status of operating system devices by pressing DISPLAY followed by the proper operand followed by RETURN. A complete list of operands is given in the summary tables which conclude this chapter.

3.6 OPERATOR DEFINITIONS FOR LEVEL I (COL)

3.6.1 LEVEL I OPERAND FORMS

(S = a storage location; L, M, and N = any level 0 operands)

FORM	MEANING
'string	The string "string". NOTE: No closing apostrophe.
S	The string stored in storage location S.
S,M	Substring of S starting at character M and continuing to the end of the string.
S,M,L	Substring of S starting at character M and continuing for L characters.
,	The inactive string.
,M	Substring of the inactive string starting at character M and continuing to the end of the string.
,M,L	Substring of the inactive string starting at character M and continuing for L characters.
LII N	Level II record N.
LII,	The level II record indicated by the active file marker.
LII N,M	Substring of level II record N starting at character M and continuing to the end of the record. If N is omitted, use the record indicated by the active file marker.
LII N,M,L	Substring of level II record N starting at character M and continuing for L characters. If N is omitted, use the record indicated by the active file marker.

3.6.2 LEVEL I OPERATORS

(Ø = any level I operand; S = a storage location; N and M = any level 0 operands; C = a single character.)

⊕ Ø

concatenates the specified operand to the end of the active string. The length of the active string is incremented by the length of the operand.

⊖ Ø

inserts the specified operand in front of the active string. The length of the active string is incremented by the length of the operand.

SUB N,M or
⊙ N,M

The character string starting at character N of the active string and continuing for M characters replaces the previous active string. The length of the buffer is set to M.

SUB N or
⊙ N

The character string starting at character N and continuing to the end of the string buffer replaces the previous string buffer. The length of the active string is decremented by N-1.

SUB ,M or
⊙ ,

The character string starting at the level I search pointer and continuing for M characters replaces the previous active string. The length of the active string is set to M.

SUB , or
⊙ ,

The character string starting at the level I search pointer and continuing to the end of the active string replaces the previous active string. The length of the active string is decremented by the search pointer minus one.

⊘

deletes a specified substring from the active string and leaves the remaining characters in the string buffer. It has the same operand forms as SUB.

ARG Ø RETURN — inserts the specified level I operand into the active string. The level I search pointer specifies the location of the first character to be inserted. The level I search pointer is not changed. If the character string inserted causes the number of characters to exceed the record length, a character at the end of the active string will be lost for each character inserted. No indication of this condition is given.

DEL — strips all trailing blanks from the active string and recalculates the length of the active string.

INV — switches the active string and the save string.

LS Ø RETURN N RETURN
RS Ø RETURN N RETURN — searches the active string for the Nth occurrence of the specified operand. RS starts with the first character and searches right. LS starts at the end of the active string and searches left. If the specified operand is not found, then the level I search pointer is set to zero. If the Nth occurrence of the specified operand is found, then the level I search pointer is set to the value of the position where the string was found. The second operand may be omitted. If it is, the online system assumes the user is searching for the first occurrence of the specified operand, and sets the level I search pointer accordingly. The first operand may be preceded by the not " ¬ " key, in which case the online system searches for the Nth occurrence not equal to the specified operand and sets the level I search pointer accordingly.

NOTE: level I RS is preferred over level I MOD, as RS does not attempt to match the operand with the string buffer until the operand has been completely specified.

EVAL Ø

compares the active string with the operand and sets a code based on the results of the comparison. The comparison proceeds from left to right. When the length of the active string is not equal to the length of the operand, the shorter string determines how much of the longer string will be used for the comparison. If the first N characters (when N is the lesser of the two lengths) are not equal the code is set to plus or minus one depending on the first non-matching characters. A minus one indicates that the first non-matching character in the active string occurred earlier in the collating sequence. A plus one indicates the reverse. When the first N characters are equal the longer string is considered to be farther in the collating sequence. When both strings are equal in length and content a zero is returned. The results of this comparison may be tested in a user program by TEST or viewed by LO EVAL + DISPLAY RETURN.

MOD CCCC...

searches the string buffer for "CCCC...". If it is found uniquely, the level I search pointer is set to that value and displayed. If it is not found, the level I search pointer is set to zero and the diagnostic "NO MATCH" is displayed. If the string is not unique the sequence MOD "CCCC" may be followed by the keys RETURN N RETURN where N, a level 0 operand, specifies to which occurrence of "CCCC" the level I search pointer is to be set. If there is no such occurrence the level I search pointer is set to zero and the message "NO SUCH OCCURRENCE" is displayed.

NOTE: level I MOD is identical in operation to level II MOD except level I MOD changes the level I search pointer.

SIN Ø$_1$ RETURN
Ø$_2$ RETURN

replaces all occurrences in the active string of the first operand with the second operand. The operation may change the length of the active string. The null string is valid second operand. In that case all occurrences of the first operand would be delted. The first operand may be preceded by the not "¬" key in which case all occurrences not equal to the first operand are replaced by the second operand.

EXAMPLES:

Active string = "ABCCDEABC"
SIN 'AB RETURN 'XY RETURN

produces: "XYCCDEXYC"

Active string = "ABCCDEABC"
SIN 'AB RETURN 'XYZ RETURN

produces: "XYZCCDEXYZC"

COS $Ø_1$ RETURN
$Ø_2$ RETURN

The characters in the active string specified by the first operand are translated to the corresponding characters in the second operand. All characters that appear in the string buffer and the first operand will be translated to the corresponding characters in the second operand. All others will not be changed. Duplicate characters in the first operand are ignored.

EXAMPLE: Active string = "THIS IS A MESSAGE" COS 'IHA RETURN 'XYZ RETURN

produces: "TYXS XS Z MESSZGE".

Each I was changed to X, the H to Y and each A to Z.

EXP Ø RETURN

expands hexadecimal digits in packed format to zoned format. Each character in the specified string is expanded to two characters. (See IBM SYSTEM/360 PRINCIPLES OF OPERATION for an explanation of the above formats).

SUM N RETURN

sets the level I search pointer to N.

ID

causes all presently supported characters to be loaded into the string buffer in the IBM collating sequence.

LOG

loads the present date and time into the string buffer.

LOAD Ø

loads the specified operand into the string buffer.

LOAD RETURN

loads the null string into the string buffer.

LOAD LO N RETURN

converts the specified level 0 operand to a character string and puts the character string in the string buffer. N may be omitted. If it is, the contents of the level 0 quotient register are converted.

DISPLAY Ø

loads the specified operand into the string buffer and displays it.

DISPLAY SPACE Ø

loads the specified operand into the string buffer and displays it without a carriage return preceding the display.

MAX Ø RETURN

displays the hexadecimal representation of the specified operand.

STORE S

replaces the previous contents of storage location S with the contents of the string buffer. The string buffer is not altered.

STORE LII N

replaces the contents of level II record N with the contents of the string buffer. LII record N must have been previously defined. The string buffer is not altered.

3.7 OPERATOR DEFINITIONS FOR LEVEL II (COL)

(M,N, and I = any level 0 operands; C = a single character.)

MANIPULATING POINTERS TO A RECORD

⊕ N RETURN

increments the active buffer pointer by N. A negative operand implies the ⊖ operator.

⊖ N RETURN

decrements the active buffer pointer by N. A negative operand implies the ⊕ operator.

⊙ N RETURN

sets the active buffer pointer to N.

SIN N RETURN

increments the save buffer pointer by N. A negative operand implies the COS operator.

COS N RETURN

decrements the save buffer pointer by N. A negative operand implies the SIN operator.

LOG N RETURN

sets the save buffer pointer to N.

BACK

replaces the preceding character in the active buffer with a blank and decrements the active buffer pointer and the save buffer pointer by 1.

NOTE: On the screen BACK blots out the deleted character; thus to see newly entered characters press RETURN which returns the carriage to the next line.

RS

fills the active buffer with blanks (does not change either pointer).

LS

sets the active buffer pointer and the save buffer pointer to 1, and returns the carriage to left of display device (does not change contents of active buffer).

MOD CCC...

(where CCC... is a character string) searches for a unique character string. If no match is found the diagnostic "NO MATCH" is displayed.

If a match is found the column number of the first character is displayed and the active buffer pointer and save buffer pointer are set to that value.

MOD CCC RETURN N RETURN

for multiple occurrences of a character string, N specifies that one is searching for the Nth occurrence of the specified string. If there is no Nth occurrence the diagnostic "NO SUCH OCCURRENCE" is displayed.

MOVING CONTENTS BETWEEN ACTIVE AND SAVE BUFFERS

INV — switches the active and save buffers.

REFL N RETURN — copies N characters from the save buffer to the active buffer. The save buffer pointer locates the first character of the character string to be moved. The active buffer pointer locates the destination of the first character. The active buffer pointer and the save buffer pointer are incremented by N.

REFL RETURN — the entire save buffer is copied to the active buffer. The active buffer pointer and save buffer pointer are not changed.

DISPLAYING AND LOADING RECORDS

DISPLAY RETURN — displays the contents of the active buffer, the value of the active buffer pointer, the value of the save buffer pointer, and the number of records in the active file.

DISPLAY N RETURN — displays the Nth record in the active file, sets the value of the active file marker to N, and loads the record into the save buffer. After DISPLAY N RETURN:

(1) Each additional RETURN displays the next record in the active file, increments the active file marker, and loads the record into the save buffer.

(2) BACK displays the preceding record in the active file, decrements the active file marker, and loads the record into the save buffer.

(3) ? displays the value of the active file marker, i.e., the number of the last record displayed.

DISPLAY . RETURN

displays the last record in the active file, sets the value of the active file marker to the record number, and loads the record into the save buffer.

DISPLAY ?

displays the record indicated by the active file marker and loads the record into the save buffer.

LOAD N RETURN

loads the Nth record from the active file into the save buffer. If N is omitted, N is assumed to be the value of the active file marker.

INSERTING AND DELETING CHARACTER STRINGS

ARG CCC RETURN

inserts the character string CCC into the active buffer. The active buffer pointer defines the location of the first character to be inserted. The active buffer pointer and the save buffer pointer are not changed. If the character string inserted causes the number of characters in the active buffer to exceed the record length, a character at the end of the active buffer will be lost for each character inserted. No indication of this condition is given.

DEL N RETURN

deletes N characters from the active buffer. The active buffer pointer defines the first character to be deleted. The remaining characters in the record are shifted left to fill the spaces occupied by the deleted characters. The active buffer pointer and the save buffer pointer are not changed.

DEL RETURN

deletes all characters to the right of the active buffer pointer. The active buffer pointer and the save buffer pointer are not changed.

STORING AND DELETING RECORDS

STORE

stores the record in the active buffer at the end of the active file, copies the record in the active buffer to the save buffer, clears the active buffer, returns the carriage, and sets the active buffer pointer and the save buffer pointer to 1.

SUB N RETURN

replaces the Nth record in the active file with the record in the active buffer, copies the record in the active buffer to the save buffer, clears the active buffer, returns the carriage, and sets the active buffer pointer and the save buffer pointer to 1. If N is omitted, N is assumed to be the value of the active file marker.

UP N RETURN

inserts the record in the active buffer before the Nth record in the active file, copies the record in the active buffer to the save buffer, clears the active buffer, returns the carriage, and sets the active buffer pointer and the save buffer pointer to 1. If N is omitted, N is assumed to be the value of the active file marker. The subsequent records are appropriately renumbered.

DOWN N RETURN

deletes the Nth record from the active file. If N is omitted, N is assumed to be the value of the active file marker. The subsequent records are appropriately renumbered.

ATAN +

enables the auto-store option. The auto-store option automatically stores the contents of the active buffer in the active file when the value of the active buffer pointer exceeds the current record length. The value of the active buffer pointer exceeds the record length when a character is entered into the last column of the active buffer, when the TAB or NEG key is pressed and there are no more tabs on the drum card, when the automatic skip-duplicate-left-zero option causes a skip past the last field, or when the ⊕, ⊖, or ⊙ key causes the value of the active buffer pointer to exceed the record length. The record is stored at the end of the active file and the active buffer is copied to the save buffer. The active buffer is then set to blanks and the active buffer pointer and save buffer pointer are set to 1. Thus the next character pressed goes into the first column of the active buffer.

ATAN - disables the auto-store option.

RECORD LENGTH

DISPLAY CTX displays current record length.

CTX N RETURN sets the record length to N.

EVAL - suppresses display of buffer pointers and card count on LII DISPLAY RETURN.

EVAL + restores display of buffer pointers and card count on LII DISPLAY RETURN.

EVAL ? displays the value of the active file marker (i.e. the record number of the last record displayed).

EVAL (displays the value of the active buffer pointer.

EVAL) displays the value of the save buffer pointer.

EVAL . displays the number of records stored in the active file.

EVAL , displays the tab (column control) card.

COLUMN CONTROL OPTIONS

⊘ + enables automatic skip-duplicate-left-zero option.

⊘ - disables automatic skip-duplicate-left-zero option.

SQ M,N,A RETURN

defines a field N characters long, starting at character M; A defines the operation code for this field:

S skips the entire field
D duplicates the entire field
N right adjusts a numeric field (left zero)

SQRT

clears all tabs and field definitions.

SET M,N,...RETURN or
SUM M,N,...RETURN

sets a tab at column M, column N,...

SET RETURN or
SUM RETURN

sets a tab at the value of the active buffer pointer.

CLR or
DIFF

clears the tab at the value of the active buffer pointer.

TAB or
NEG

skips to the next tab setting.

EXP

copies the contents of the active buffer to the drum card, illegal drum card characters are replaced by blanks. The contents of the active buffer are not changed.

3.8 OPERATOR DEFINITIONS FOR LEVEL III (COL)

(M,N = any level 0 operands; Ø = any level I operand)

⊕

concatenates the inactive file onto the active file. (The inactive file is purged).

⊙ N RETURN

sets the active file marker to N.

INV

switches the active file and the inactive file.

UP N RETURN

inserts all of the inactive file before the Nth record in the active file. If N is omitted, it is assumed to be the value of the active file marker. The inactive file is unchanged.

DOWN

purges the active file and sets the active file marker to zero.

EVAL N RETURN Ø RETURN

searches the active file starting at the first record beyond the active file marker for the designated character string. N is the column where one expects to find the first character of the character string. Ø is any valid level I operand. If the specified operand is not found the active file marker is set to zero and displayed on the output device. If the specified operand is found the active file marker is set to the record number of the record containing the operand and the value of the active file marker is displayed. The level I operand may be preceded by the not "¬" key, in which case the search is made for the first record in which the specified operand does not appear.

MOD Ø RETURN

searches the active file starting at the first column in the first record beyond the active file marker for the specified string. Ø is any valid level I operand. If the string is found, the file marker is set to the record number of the record containing the string, the level I search pointer is set to the column number of the substring, and the file marker is displayed If the string is not found both the file marker and the search pointer are set to zero. The level I operand may be preceded by the not "¬" key, in which case the search is made for the first record in which the specified operand does not appear.

NOTE: The character string sought must be on one record.

SORT N,M RETURN or
SQRT N,M RETURN

sorts a copy of the inactive file, as specified and concatenates the resultant sorted file onto the active file. The inactive file is unchanged. N specifies the first column of the sort field and M is the length of the sort field. M may be omitted in which case the rest of the record is used as the sort field.

CONJ N,M RETURN

merges the inactive file into the active file. The inactive file is not changed. N specifies the first column of the merge field. M specifies the length of the merge field. M may be omitted; if it is, the rest of the record is used as the merge field. CONJ assumes the files are sorted; no checks are made.

ARG M,N,I RETURN

inserts a copy of M records from the inactive file before record N in the active file. I is the record number of the first record in the inactive file to be copied. N and I may be omitted, in which case their values default to the respective file markers. M is required. The inactive file is not changed by this operation.

DEL M,N RETURN

deletes M records from the active file, starting with record number N. N may be omitted, in which case it defaults to the active file marker. The remaining records are appropriately renumbered.

MAX N,M RETURN

puts a sequence number in the specified field of each record in the active file. N specifies the first column of the sequence number field. M specifies the length of the sequence number field. M may be omitted; if it is, the rest of the record is used as the sequence number field. The sequence numbers are padded on the left with zeroes.

DISPLAY M,N RETURN

displays M records, starting with record number N.

DISPLAY M RETURN

displays M records, starting with the first record beyond the active file marker.

DISPLAY , N RETURN

displays the remainder of the active file starting with record N.

DISPLAY , RETURN

displays the remainder of the active file starting with the first record beyond the active file marker.

DISPLAY RETURN

displays all of the active file.

3.9 OPERATOR DEFINITIONS FOR LEVEL IV (COL)

LOAD

concatenates the specified data set from the operating system with the contents of the active file. The entire record of the OS data set is kept (up to a maximum of 254 characters).

NOTE: The COL record length is unchanged; thus DISPLAY will not display the entire record if the OS data set record length is greater than the COL record length.

SUB

submits the active file to the operating system for batch processing.

DISPLAY Ø RETURN

displays the jobs active in the system and the status of the indicated devices. Ø is one or more of the following operands:

J = jobs currently active in the operating system.

D = direct access data devices.

T = tapes.

U = all unit record equipment.

G = graphics devices.

C = communication devices (none).

A = all of the above.

3.10 OPERATOR DEFINITIONS FOR LEVEL 0 EVAL (COL)

EVAL CTX loads the current record length into the level 0 quotient register.

EVAL . loads the number of records in the active file into the level 0 quotient register.

EVAL ? loads the current value of the active file marker into the level 0 quotient register.

EVAL (loads the current value of the level II active buffer pointer into the level 0 quotient register.

EVAL) loads the current value of the level II save buffer pointer into the level 0 quotient register.

EVAL RETURN loads the current length of the level I active string into the level 0 quotient register.

EVAL S loads the length of the character string stored under level I storage location S into the level 0 quotient register.

EVAL + loads the result of the last level I EVAL operation into the level 0 quotient register.

EVAL , loads the current value of the level I search pointer into the level 0 quotient register.

EVAL LI treats the level I active string as a decimal number, converts it to an integer, and places the result in the level 0 quotient register. If there are one or more minus signs in the active string, the resultant number will be negative. If a non-decimal character is encountered, the diagnostic "OPERATION ABORTED" is displayed. If the active string represents an integer greater than the computer can handle, the diagnostic "FIXED POINT OVERFLOW" is displayed.

EVAL # treats the level I active string as a hexadecimal number, converts it to an integer, and places the result in the level 0 quotient register. If there are one or more minus signs in the active string, the resultant number will be negative. If a non-hexadecimal character is encountered, the diagnostic "OPERATION ABORTED" is displayed. If the active string represents an integer greater than the computer can handle, the diagnostic "FIXED POINT OVERFLOW" is displayed.

EVAL - loads the decimal equivalent of the IBM EBCDIC bit code for the first character in the level I string buffer into the level 0 quotient register.

EVAL ! treats the first four characters of the level I string buffer as a bit string, which represents an integer. The integer is moved to the level 0 quotient register.

MATHEMATICALLY ORIENTED LANGUAGE
SINGLE PRECISION FLOATING POINT (MOLSF)

MOLSF has four levels of mathematical operators and data structures. Level I operators enable one to perform calculations on scalars (single numbers). Level II operators enable one to perform calculations on vectors (ordered lists of scalars). Level III operators enable one to perform calculations on two-dimensional arrays. Level V is reserved for operators for which there is no space on existing levels. For example, a user can pass MOLSF data to a FORTRAN or PL1 batch program and have the results of these batch programs returned to MOLSF data structures.

The selection of MOLSF operators has been made to provide a balance between ease of mathematical formula construction and simplicity of operator definitions. The sections preceding the definition of the MOLSF operators provide the background necessary to efficiently use MOLSF. They discuss MOLSF's internal number representation, MOLSF's data structures, MOLSF's computational format and the working registers, MOLSF's operand forms, the explicit loading of data into the working registers, the storing of data for later use, and finally a detailed description of MOLSF display facilities. (Note: Except for the first sub-section in the display section, display may be left for a later reading.)

4.1 NUMBER REPRESENTATION

MOLSF uses scientific notation (floating point) to represent scalars. Each number is defined by a mantissa and an exponent. For example, the number 4,900,000 may be written as 0.49×10^7, where 0.49 is the mantissa and 7 is the value of the exponent, or 0.0023 may be written as 0.23×10^{-2}. The actual representation system may be expressed as

$$y = M \times R^p$$

where y is the number to be represented, M is the mantissa, R is the radix or base, and p is the integer exponent. Numbers are stored and manipulated internally in floating point-binary form (R = 16), but are typed or displayed as decimal numbers (R = 10) in fixed or floating point form.

Numbers are entered in the form

$$\pm M \pm p$$

where M is the mantissa (which may include the decimal point in any position) and p is the power to which the base R = 10 is raised. If p = 0 it may be omitted. The first sign indicates whether the number itself is positive or negative and may be omitted if it is +; the second sign shows whether the exponent is positive or negative, and must be included if p is included. Thus -0.49×10^7 would be typed in as -.49+7 and 0.23×10^{-2} as .23-2.

To summarize, the rules pertaining to the typing of numbers

A + B + C + D

MOLSF allows the user to simplify this expression by juxtaposing operands for all binary operators (⊕ , ⊖ , ⊙ , ⊘).

EXAMPLE: To add four scalars stored under A, B, C, and D the following sequences are equivalent.

LOAD A ⊕ B ⊕ C ⊕ D

LOAD A ⊕ BCD

EXAMPLE: To add the scalars 91, 77, A, 173, 71 the following sequences are equivalent.

LOAD 91 ⊕ 77 ⊕ A ⊕ 173 ⊕ 71

LOAD 91 ⊕ 77 A 173 RETURN 71 RETURN

4.3.3 TRAILING PREDICATES

It is often an inconvenience for the user to press LOAD every time he wishes to work with a new operand. MOLSF allows the user to implicitly load operands when working with unary operators.

LOAD A SIN is equivalent to SIN A. This can be helpful when constructing a mathematical expression.

EXAMPLE: $\sin^2(A) + \cos^2(A)$

Without trailing predicate:

LI REAL LOAD A SIN SQ STORE T LOAD A COS SQ ⊕ T

With trailing predicates:

LI REAL SIN A SQ STORE T COS A SQ ⊕ T

With trailing predicates and parentheses:

LI REAL SIN A SQ ⊕ (COS A SQ)

4.4 LOADING OF DATA

The primary function of the LOAD key is to explicitly enter numbers or copy data from storage locations into the working registers for the level the user is presently working on. Data may be copied from storage locations on any level (I, II, or III). In many instances, the LOAD key is unnecessary. For example, LOAD Z SIN is equivalent to SIN Z. In the latter case, the loading of data into the working register is implicit, and is the preferred sequence. (See section 4.3.3 on trailing predicates.)

When one loads data into the working register the contents of any specified storage location are not changed. Concisely, LOAD provides explicit, nondestructive recall from temporary data storage to the working registers.

4.4.1 LOAD FOLLOWED BY A NUMBER (a numerical operand)

If the LOAD key is followed by a number, then that number is loaded into every component of the current level's working register.

EXAMPLES:

1) LI REAL LOAD 13 RETURN enters 13 into β_I.
2) LII CMPLX LOAD 3,7 RETURN places 3 + 7i = (3,7) into every component of the $(\alpha_{II}, \beta_{II})$ register.
3) LIII REAL LOAD 2 RETURN enters 2 into every component of the β_{III} register.

4.4.2 GENERAL MOLSF LOAD FORMAT

The general format for the keys which may follow LOAD is:

LOAD [level] $\begin{bmatrix}\text{REAL}\\ \text{CMPLX}\end{bmatrix}$ [location][(component)]

As the brackets indicate, all the keys are optional. The "level" designation is LO, LI, LII, or LIII. A level key need only be included to load data from a level other than the one which is presently active. REAL or CMPLX indicates that the data is to come from the specified mode. "Location" indicates which of the 52 storage locations is to be copied. If "location" is omitted, then the working register is used as the source of the data to be copied. "(component)" may be any level 0 operand and indicates the specific component to be used. The parentheses around the component entry are required. The actual component may be omitted; the online system assumes any missing indices are equal to one. The component entry is necessary when copying data from a higher to a lower level.

4.4.3 LOADING DATA WHILE ON LEVEL I

All of the following examples assume the user is working on level I and explicitly wishes to load data into the appropriate working register. In all cases, the source of the data is not changed.

EXAMPLES:

1) LI CMPLX LOAD Z - The contents of the level I complex scalar Z are copied into the (α_I,β_I) working register.

2) LI REAL LOAD X - The contents of the level I real scalar X are copied into the β_I working register. The contents of α_I are not changed.

3) LI REAL LOAD CMPLX Z - The contents of the real part of the level I complex scalar Z are copied into the β_I working register. The contents of α_I are not changed.

4) LI REAL LOAD LII A(3)- The contents of the third component of the level II real vector A are loaded into the β_I working register.

5) LI REAL LOAD LII (7) - The contents of the seventh component of the β_{II} working register are copied into the β_I working register.

6) LI REAL LOAD LII R(I) - The contents of component I of the level II real vector R are copied into the β_I working register. The value of I is obtained from the integer stored under level 0 I.

7) LI REAL LOAD LIII A (I,J) - The contents of component I,J of the level III real array A are copied into the β_I working register.

8) LI REAL LOAD LIII (,) - The contents of component "(1,1)" of the β_{III} working register are copied into the β_I working register.

4.4.4 LOADING DATA WHILE ON LEVEL II

All of the following examples assume the user is working on level II and explicitly wishes to load data into the appropriate working register. In all cases the source of the data is not changed.

EXAMPLES:

1) LII REAL LOAD G - The contents of the level II real vector G are copied into the β_{II} working register.

2) LII REAL LOAD LI G - The contents of the level I scalar G are copied into every component of the β_{II} working register.

3) LII REAL LOAD LI CMPLX X - The contents of the real part of the level I complex scalar X are copied into every component of the β_{II} working register.

4) LII REAL LOAD LIII A (1,) - The contents of the first row of the level III array A are copied into the β_{II} working register.

5) LII CMPLX LOAD LIII A (2) - The contents of the second row of the level III complex array A are copied into the (α_{II}, β_{II}) working register.

6) LII CMPLX LOAD LIII REAL A(,2) - The contents of the second column of the level III real array A are copied into the α_{II} working register.

4.4.5 LOADING DATA WHILE ON LEVEL III

All of the following examples assume the user is working on level III and explicitly wishes to copy data into the appropriate level III working register. In all cases the source of the data is not changed.

EXAMPLES:

1) LIII CMPLX LOAD A - The contents of the level III complex array A are copied into the (α_{III}, β_{III}) working register.

2) LIII CMPLX LOAD REAL X - The contents of the level III real array X are copied into the α_{III} working register. The β_{III} register is set to zero.

3) LIII REAL LOAD CMPLX X - The contents of the real part of the level III complex array X are copied into the β_{III} working register.

4) LIII REAL LOAD LI Z - The contents of the level I real scalar Z are copied into every component of the β_{III} working register.

5) LIII REAL LOAD LII L - The contents of the level II real vector L are copied into each column of the β_{III} working register.

4.4.6 INCREMENTING THE COMPONENT IN LOAD

The component entry in the general MOLSF LOAD format may be any level 0 operand. Thus it is possible to increment or decrement any index variable in a LOAD format. For a complete list of level 0 operands see section 2.3.1.

In the following examples there is no key sequence which leads the user to believe the keys would be executed more than once. Their explanations are based on the assumption that they are embedded in a user program which is repeated a number of times in the course of solving a problem.

EXAMPLES:

1) LI REAL LOAD LII M(K+) - The variable K is a level 0 operand. The first time the instruction sequence is executed, M(K) is loaded into β_I and K is incremented by 1. The next time the sequence is executed, the entry M(K) for the new value of K is loaded into β_I. K is incremented again each time the sequence is repeated. The level 0 operand can be decremented instead of incremented if the minus sign is used instead of the plus sign. These are the lower keyboard plus and minus signs, not the operator keys ⊕ and ⊖. If the desired increment or decrement is not unity, then the + or - sign should be followed by the desired integer specification.

2) LI REAL LOAD LII L(N+J) where N and J are level 0 operands. If N = 3 and J = 2, then L(3), L(5), L(7), L(9), etc., are loaded into β_I in turn as the instruction sequence is repeatedly executed.

4.4.7 LOADING VECTORS AND ARRAYS WITH VARYING DIMENSIONS

The online system allows complete freedom in loading data of varying sizes into the working registers. Vectors or arrays which have smaller sizes than the current size of the working registers will be completely copied into the working registers. The components already in the working register beyond the data loaded will not be changed. If the size of the vectors or arrays loaded exceeds the current size of the working registers, then only that part of the data up to the limit of the working registers is copied.

EXAMPLE: Suppose the present context of the working register is 51; we wish to combine two vectors F and G each of context 51 such that the resulting vector has the first 46 components of F and the last 5 components of G. Press:

LII LOAD G CTX 46 LOAD F CTX 51

4.5 STORING OF DATA

The STORE key is the antithesis of LOAD. It is used to copy the contents of the working register into a storage location. There are fifty-two unique storage locations (A - Z, α - ω) for each mode on each level, i.e. 52 REAL and 52 CMPLX storage locations. The previous contents of the designated storage locations are replaced by the quantity which is stored. The level specification most recently preceding the alphabetic key will be the one used to determine which storage location is desired. On levels II and III, the context of the storage location is automatically set to that of the working register.

The general format for the keys which follow STORE is exactly the same as that for LOAD:

$$\text{STORE [level]} \begin{bmatrix}\text{REAL}\\ \text{CMPLX}\end{bmatrix} \text{[location] [(component)]}$$

1) "level" is LO, LI, LII, LIII, or omitted.

2) "location" is an alphabetic key A through Z, α through ω.

3) "(component)" is "(i)", or "(i+j)", or omitted, with i and j any level 0 operands.

STORE does not change the contents of the working register.

EXAMPLES:

1) LI REAL LOAD 3.2 STORE A - stores the real scalar 3.2 in level I real A.

2) LI REAL LOAD B ⊙ 5 ⊕ 3 STORE D - stores the real scalar 5B + 3 in level I real D.

3) LI REAL LOAD 9 STORE CMPLX A - stores the complex scalar 9 + 0i in level I complex A.

4) LI REAL LOAD I STORE LII Q(J) - stores the contents of level I real I in component J of the level II real vector Q. J is any level 0 operand.

5) LI CMPLX LOAD 3,1 STORE T - stores the complex scalar 3 + i in level I complex T.

6) LI REAL LOAD 37.2 STORE LIII CMPLX A(,) - stores the complex scalar 37.2 + 0i in the first component (1,1) of the level III complex array A.

7) LI REAL LOAD 8.9 STORE LO C - truncates the real scalar 8.9 to the integer 8 and stores it in level 0 C.

8) LII REAL LOAD 1 STORE S - stores 1 into every component of the level II real vector S.

9) LII REAL ID STORE X - stores the uniformly-spaced discrete domain of the interval $-1 \leq x \leq 1$ in level II real X.

10) LII REAL LOAD A STORE LI B - stores the first component of the level II real vector A in level I real B.

11) LIII CMPLX LOAD A STORE LI B - stores the first component (1,1) of the level III complex array A in level I complex B.

12) LIII REAL LOAD A STORE LII C - stores the first column of the level III real array A in level II real C.

NOTE: When executing a STORE operation into a higher level, the absence of a "location" specification implies storing into the working register element(s) specified by "level" "(component)". For example, LI REAL STORE LII (K) stores the contents of β_I into component K of the β_{II} working register.

(In the above examples, the repeated use of REAL and CMPLX is only for illustration and normally is only required to change from one to the other).

registers is computed. To display vectors on a common scale in dot or dot-dot mode, specify the list of vectors to be displayed in line mode, if any; then push dot or dot-dot followed by a list of vectors to be displayed in that mode.

LII DISPLAY , A.BC..D RETURN

The vector A is displayed normally, B and C in dot mode, and D in dot-dot mode. The scale used for the display is the greatest of the scales of A, B, C, and D. The dot or dot-dot may be placed anywhere in the sequence and may be repeated. Therefore, the sequence LII DISPLAY ,A..B.C..D RETURN is valid. After the comma, a number may be specified to indicate the scale to be used in displaying a curve or a series of curves.

EXAMPLE:

LII DISPLAY ,2A RETURN

The vector A is displayed with a scale of two, regardless of maximum scale.

LEVEL III DISPLAY

Level III display is similar to level II display. With very few conceptual changes the same sequences execute similar operations in two dimensions instead of one.

One important concept that does differ is scaling. In level III display the user has no control over the scale as he does in level II display.

It is to the user's advantage to note that for the higher dimensions displays become cluttered and therefore somewhat difficult to read.

4.6.4 DISPLAY FORMATTING

Numerical and curvilinear dot-dot displays may be formatted. There are three types of formats: integer display format, floating-point display format, and the dot-dot graphical display character.

INTEGER DISPLAY FORMAT ITEM (L0 data, LII and LIII contexts, display scales) - ($n \leq 24$)

In Left justified - leading zeros suppressed

Ln Right justified - leading zeros not suppressed

Sn Right justified - leading zeros suppressed

Xn Right justified - leading zeros not suppressed. Hexadecimal numbers displayed instead of decimal.

n specifies the number of places. Overflow is indicated by an asterisk (*) in the sign position. The default format specification is I10.

FLOATING-POINT DISPLAY FORMAT ITEM - ($n + m \leq 24$)

Dn.m Float - trailing zeros suppressed.

En.m Float - trailing zeros not suppressed

Fn.m Fixed - no exponent displayed, leading zeros suppressed.

n specifies the number of places to the left of the decimal place and m the number to the right. An overflow or underflow in the "F" format is indicated by an asterisk in the sign position. The default floating point format is D1.5.

DOT-DOT FORMAT ITEM

One lower-keyboard character

When dot-dot display is to be done the character specified is used in place of the normal (large) dot. The default dot-dot format item is the dot specified by a period.

The user may change the format on L0 (index), LI, LII, or LIII with a sequence of the form "DISPLAY (format item, format item, ...)". Any format item may be changed on any level and the format items may be specified in any order separated by commas. Should a format of the same type be repeated, the most recent specification is used. If a RETURN appears during format specification, the current format items will be displayed. The format items just specified are not stored until the right parenthesis is pressed.

EXAMPLE: Change the dot-dot format item to a question mark. Press:

DISPLAY (?)

EXAMPLE: Change the dot-dot format item to an asterisk and the floating point format item to fixed form. Press:

DISPLAY (F10.5,*)

EXAMPLE: Display the present format items. Press:

DISPLAY (RETURN

4.7 MATHEMATICAL OPERATORS FOR LEVEL I

The following operand notation is used in describing the mathematical operators:

1) S represents a storage location as defined by an alphabetic operand. (See section 4.3.1)

2) "r", "r_1", and "r_2" represent real numbers, as entered on the numeric keys.

4.7.1 OPERATOR DEFINITIONS FOR LEVEL I REAL (MOLSF)

⊕ , ⊖ , ⊙ , ⊘ followed by S or "r" computes the indicated combination with the number in the β_I register and leaves the result in the β_I register. If one of these operators is followed by any *other* operator, it has no effect.

PWR followed by S or "r" raises the contents of the β_I working register to the specified power and leaves the result in the β_I working register. If PWR is followed by S, then the contents of storage location S are used as the exponent. If PWR is followed by any other operator key, then it has no effect.

SUB K puts the contents of the β_I register into component K of the β_{II} working register. K must be a positive integer or a level 0 operand, not larger than the current context.

EVAL K puts component K of the β_{II} working register into the β_I register. K must be a positive integer or a level 0 operand, not larger than the current context.

(In the following, if the operator key is followed immediately by S or "r", the operand is the value in the REAL storage location S, or the number r, respectively. If the operator key is followed by any other keypush, the operand is the value already in the β_I register. The result is always put into the β_I register.)

SQ squares the operand.

SQRT takes the square root of the operand. The real square root of a negative number is defined to be zero.

NEG negates the operand.

INV takes the reciprocal of the operand.

MOD takes the absolute value of the operand.

SIN, COS, LOG, EXP, ATAN performs the indicated operation on the operand. LOG acts on the absolute value of the operand. LOG of zero gives -183.846.

ARG 0 if operand $\geq$ 0; π if operand < 0.

DEL 0 if operand $\neq$ 0; 1 if operand = 0.

ID sets the β_I working register to 1.

4.7.2 OPERATOR DEFINITIONS FOR LEVEL I COMPLEX (MOLSF)

$\oplus$, $\ominus$, $\odot$, $\oslash$ followed by S or "r_1, r_2" computes the indicated complex combination with the complex number in the (α_I, β_I) register. And leaves the result in the (α_I, β_I) register.

PWR followed by S or "r" raises the contents of the (α_I, β_I) working register to the specified power and leaves the result in the (α_I, β_I) working register. If PWR is followed by S, then the contents of storage location S are used as the exponent. If PWR is followed by any other operator key, then it has no effect.

SUB K puts the contents of the (α_I, β_I) register into component K of the α_{II}, β_{II}) working register. K is a positive integer or a level 0 operand, not larger than the current context.

EVAL K puts component K of the (α_{II}, β_{II}) working register into the (α_I, β_I) register. K is a positive integer or a level 0 operand, not larger than the current context.

(In the following, if the operator key is followed immediately by S or "r_1, r_2", the operand is the value in the complex storage location S, or the complex number $r_1 + ir_2$, respectively. If the next key pushed after the operator key is not S or "r_1, r_2", the complex number in (α_I, β_I) is the operand. The result is always put into the (α_I, β_I) register).

SQ
squares the operand.

SQRT
takes the complex square root of the operand, using the branch of the square root such that the argument of the answer is half the argument (defined by ARG) of the original complex number.

NEG
takes the complex conjugate of the operand.

INV
takes the complex reciprocal of the operand.

REFL
interchanges the real and imaginary components of the operand.

MOD
takes the modulus of the operand, puts it in α_I and puts zero in β_I.

SIN, COS, LOG, EXP, ATAN
performs the indicated operation on the operand; LOG takes the branch provided by ARG. LOG of zero gives -183.846 + 0i and puts it into the (α_I, β_I) register.

ARG or
ARG -
computes the argument in the interval [$-\pi$, π] of the operand, puts it in α_I, and sets β_I to zero. The argument of 0 + 0i is defined to be zero.

ARG +
computes the argument in the interval [0, 2π] of the complex number in the (α_I, β_I) register and puts it in α_I, β_I is set to 0. The argument of 0 + 0i is defined to be zero.

DEL
0 + 0i if operand ≠ 0 + 0i; 1 + 0i if operand = 0 + 0i.

ID
sets (α_1, β_I) equal to 1.0 + 0.0i

4.7.3 ADDITIONAL COMMENTS ON LEVEL I

Data in the working register can be transferred between level I REAL and level I CMPLX by simply changing modes. A real number in β_I on level I REAL becomes the imaginary part of (α_I, β_I) on level I CMPLX. Thus if the real number 6 were in β_I on level I REAL and keys LI CMPLX were pushed, the number would still be in β_I on level I CMPLX. If the contents of α_I were initially 0, the complex number in (α_I, β_I) would now be 0 + 6i. Likewise, when the level is changed from level I CMPLX to level I REAL, the imaginary part of the complex number becomes the real number on level I REAL.

Several simple examples of operations on level I are given below. More detailed examples are presented in Appendix E.

1) LI REAL EXP X DISPLAY RETURN. The number e^x, x the single number contained in X, is calculated and printed on the display scope.

2) LI REAL LOAD Y REPT SIN 3 RETURN. The single number sin (sin (sin y)) is computed. The result is in the β_I register.

3) LI CMPLX LOAD 3,2 LOG DISPLAY RETURN. The principal value of log (3 + 2i) is computed and displayed.

4) LI CMPLX MOD Z DISPLAY RETURN. The modulus of the complex number stored in Z is computed and displayed on the scope. For example, if the number 3 + 4i were in Z, the modulus 5, 0 would be printed on the scope; (i.e., modulus = $\sqrt{3^2 + 4^2} = \sqrt{25} = 5$).

The repeated use of level specifications LI REAL and LI CMPLX occurs in the above examples for the purpose of illustration. In general, such specifications are only used when individual level changes are required.

4.8 MATHEMATICAL OPERATORS FOR LEVEL II

4.8.1 OPERATOR DEFINITIONS FOR LEVEL II REAL (MOLSF)

Throughout the description of the level II REAL operators it is assumed that the vectors needed have previously been defined and are available for use. The general notation adopted for the description of the level I operators is also employed. Note that S, which represents a storage location, now implies a vector $S = (s_1, s_2, \ldots, s_n)$ and "r", which represents a real number, now defines a constant vector of n components.

$\oplus$, $\ominus$, $\odot$, $\oslash$ followed by S or "r" performs the indicated operations componentwise using the vector in the β_{II} register and the real vector in S or "r". Let $(\beta_1, \beta_2, \ldots, \beta_n)$ denote the contents of β_{II} before any of these operations. Then the result, in β_{II}, will be as follows:

$\oplus$ S: $\beta_{II} = (\beta_1 + s_1, \beta_2 + s_2, \ldots, \beta_n + s_n)$

$\ominus$ S: $\beta_{II} = (\beta_1 - s_2, \beta_2 - s_2, \ldots, \beta_n - s_n)$

$\odot$ S: $\beta_{II} = (\beta_1 s_1, \beta_2 s_2, \ldots, \beta_n s_n)$

$\oslash$ S: $\beta_{II} = (\beta_1/s_1, \beta_2/s_2, \ldots, \beta_n/s_n)$

If zeros occur in some, or all, components of the operand vector in division, results for those components will be zero.

PWR followed by S or "r" raises the contents of each component of the β_{II} working register to the specified power and leaves the result in the β_{II} working register. If PWR is followed by S, then the contents of the vector S are used as the exponents.

LS

shifts each component β_k of the β_{II} register into position (k-1) of the β_{II} register, placing the first component in the last position.

$$\beta_{II} = (\beta_2, \beta_3, \ldots, \beta_n, \beta_1)$$

LS K (K a level 0 operand) is equivalent to repeating LS K times.

RS

shifts each component β_k of the β_{II} register into position (k+1) of β_{II} placing the last component into the initial position.

$$\beta_{II} = (\beta_n, \beta_1, \beta_2, \ldots, \beta_{n-1})$$

RS K (K a level 0 operand) is equivalent to repeating RS K times.

ENL

doubles the mantissa of each component of the β_{II} register, for display purposes, and decrements the binary scale by 1 so that the magnitude is not changed. ENL K (K a level 0 operand) is equivalent to repeating ENL K times.

CON

halves the mantissa of each component of the β_{II} register, for display purposes, and increments the binary scale by 1, so that the magnitude is not changed. CON K (K a level 0 operand) is equivalent to repeating CON K times.

NOTE: For LS, RS, ENL, and CON a negative operand implies the inverse operator. For example, LS -3 is equivalent to RS 3.

EVAL

If X is in α_{II} and f(X) is in β_{II}, EVAL followed by S replaces f(X) with f(S). The process is as follows: for each component s_i of S the least upper bound, x_k, and the greatest lower bound, x_j, with respect to the α_{II} register, are found. Linear interpolation then gives the value of $f(s_i)$ as

$$f(s_i) = \frac{f(x_k) - f(x_j)}{x_k - x_j}(s_i - x_j) + f(x_j)$$

If $s_i$is greater than (or less than) all of the α_{II} components, the value of $f(s_i)$ is set equal to the β_{II} correspondent of the maximum (minimum) α_{II} component. If the dimension of S is not equal to the dimension of the β_{II} register, the α_{II} and β_{II} registers containing the result, "X" and f(S), will have the context of S. If followed by "r", EVAL creates a constant vector and proceeds as above. If r is a negative number, it must be enclosed in parentheses.

EVAL + Similar to EVAL except that $f(s_i)$ is replaced by the function of the least upper bound, $f(x_k)$.

EVAL - Similar to EVAL except that $f(s_i)$ is replaced by the function of the greatest lower bound, $f(x_j)$.

ID If the working register length is n, ID places a vector consisting of n equally spaced values from -1 to +1, (beginning with -1, ending with +1) in the α_{II} and β_{II} registers.

$$\alpha_{II} = \beta_{II} = \left(\frac{2k-n-1}{n-1} \middle| \; k = 1, 2, \ldots, n\right)$$

ID X places the vector consisting of n equally spaced values from -1 to +1 in the α_{II} register only. β_{II} is unchanged.

$$\alpha_{II} = \left(\frac{2k-n-1}{n-1} \middle| \; k = 1, 2, \ldots, n\right)$$

X in this case is not an operand.

ID Y places the vector consisting of n equally spaced values from -1 to +1 in the β_{II} register only. α_{II} is unchanged.

$$\beta_{II} = \left(\frac{2k-n-1}{n-1} \middle| \; k = 1, 2, \ldots, n\right)$$

Y in this case is not an operand.

ID ? places a vector consisting of n uniformly distributed random values in the interval [-1, +1] in β_{II}.

ID RETURN

Similar to ID ?, except that the contents of β_{II} are used to compute the random numbers.

SUB

When followed by S, SUB loads the contents of S into the α_{II} working register. When followed by "r", SUB loads "r" into the α_{II} working register. When followed by ( ), puts the contents of β_{II} into α_{II}.

[In the following, if the operator key is followed immediately by S or "r", the operand is the vector in the REAL storage location S, or the constant vector r, respectively. If the next key is not one of these, the operand is the real vector in β_{II}. The result is always put into the β_{II} register.]

SQ

squares each component of the operand.

$$\beta_{II} = (s_1{}^2, s_2{}^2, \ldots, s_n{}^2)$$

SQRT

takes the square root of each component of the operand, assigning the value of zero to each negative component.

$$\beta_{II} = (\sqrt{s_1}, \sqrt{s_2}, \ldots, \sqrt{s_n})$$

NEG

negates each component of the operand.

$$\beta_{II} = (-s_1, -s_2, \ldots, -s_n)$$

INV

computes the reciprocal of each component of the operand.

$$\beta_{II} = (1/s_1, 1/s_2, \ldots, 1/s_n)$$

DIFF

forms the forward difference of the components of the operand, performing a second-order extrapolation to supply the last component in the result.

$$\beta_{II} = (s_2-s_1, s_3-s_2, \ldots, s_n-s_{n-1}, 2s_n-3s_{n-1}+s_{n-2})$$

SUM

forms the running summation of the components of the operand.

$$\beta_{II} = (s_1, s_1+s_2, s_1+s_2+s_3, \ldots, \sum_{k=1}^{n} s_k)$$

PROD — forms the running product of the components of the operand

$$\beta_{II} = (s_1, s_1s_2, s_1s_2s_3, \ldots, \overset{n}{\underset{k=1}{\pi}} s_k)$$

REFL — reverses the order of the n components of the operand.

$$\beta_{II} = (s_n, s_{n-1}, \ldots, s_1)$$

MOD — takes the absolute value of each component of the operand.

$$\beta_{II} = (|s_1|, |s_2|, \ldots, |s_n|)$$

MAX — sets each component of the β_{II} register equal to the maximum component of the operand.

$$\beta_{II} = (\max s_k, \max s_k, \ldots, \max s_k)$$

SIN, COS, LOG, EXP, ATAN

performs the indicated operation componentwise on the operand vector.

$$\left.\begin{array}{l} \underline{\text{SIN}}\text{:}\quad \beta_{II} = (\sin s_1, \sin s_2, \ldots, \sin s_n) \\ \underline{\text{COS}}\text{:}\quad \beta_{II} = (\cos s_1, \cos s_2, \ldots, \cos s_n) \end{array}\right\} \text{(s's in radians)}$$

$$\underline{\text{LOG}}\text{:}\quad \beta_{II} = (\ln s_1, \ln s_2, \ldots, \ln s_n)$$

LOG of 0 gives -183.846.

$$\underline{\text{EXP}}\text{:}\quad \beta_{II} = (e^{s_1}, e^{s_2}, \ldots, e^{s_n})$$

$$\underline{\text{ATAN}}\text{:}\quad \beta_{II} = (\tan^{-1}s_1, \tan^{-1}s_2, \ldots, \tan^{-1}s_n) \quad \text{(results in radians)}$$

SORT — rearranges the components of the β_{II} working register in numerically increasing order. At the same time the integer representing the original position of each component is placed in the α_{II} working register.

SORT A,B

rearranges A using each component of B as an index to designate which component of A will be loaded into α_{II}. The components of B are truncated to integers. Thus

$\beta_{II}(1) = A(B(1))$

$\beta_{II}(2) = A(B(2))$ etc.

If the value of any component of B is less than or equal to zero, then the first component of A is loaded into the indicated component of β_{II}. If the value of any component of B is greater than the context of A, then the last component of A is loaded into the indicated component in β_{II}. The α_{II} working register is not changed. A may be omitted. If it is, then the contents of the β_{II} working register will be sorted as specified.

ARG

assigns the value zero to all non-negative components, the value π = 3.14159 to all negative components of the operand.

DEL

identifies zeros and sign changes in the operand vector as follows:

If $s_k = 0$, $\beta_k = 1$

If $(s_k)(s_{k+1}) < 0$, $\beta_k = 1$ if $|s_k| \leq |s_{k+1}|$

$\beta_{k+1} = 1$ if $|s_{k+1}| < |s_k|$

All other $\beta_k = 0$

CONV

provides a means for obtaining a discrete approximation to the integrals

$\int_{-\infty}^{\infty} K(t - \tau) \; F(\tau) \, d\tau$

$\int_{0}^{t} K(t - \tau) \; F(\tau) \, d\tau$

and

$\int_{0}^{2\pi} K(t - \tau) \; F(\tau) \, d\tau$ K period = 2π

by pressing: LII REAL LOAD F CONV K,J where J is the component number of K(t-τ) which is to be initially aligned with the first element of F. (J any level 0 operand.)

CONVOLUTION

The nature of this discrete approximation is such that the user is expected to reform the kernel to be a distributed kernel (i.e., the weighting factors resulting from the user-selected integration formula are included in the representation of the kernel). In the simplest situation, these weights may all be equal to the step selected for the independent variable. In this case, K would be replaced by K • Δτ before the convolution operator is used.

In the following discussion, these weights will be presumed to be included and K represented by a vector LII REAL K = (k_1, k_2, ..., k_m). The easiest way to describe the convolution computation is by explaining the matrix multiplier derived from K by CONV K,J (J is any level 0 operand which must be less than or equal to m). J defines the matrix extension of K by specifying the upper left hand entry as shown below. The first step of the calculation extends K to the matrix

$$\begin{pmatrix}
K_j & K_{j+1} & \cdots\cdots & K_m & 0 & \cdots\cdots & 0 \\
K_{j-1} & K_j & K_{j+1} \cdots\cdots & & K_m & 0 \cdots & 0 \\
\vdots & & & & & & \vdots \\
K_1 & K_2 & \cdots\cdots\cdots\cdots & & & & K_m \\
0 & K_1 & K_2 \cdots\cdots\cdots\cdots & & & & K_{m-1} \\
\vdots & & & & & & \vdots \\
0 & \cdots\cdots\cdots\cdots & & 0 & K_1 & \cdots\cdots & K_j
\end{pmatrix}$$

This matrix then becomes the multiplier for the column vector F.

To use the convolution operator one must progress through the following steps:

1) Create the kernel as a vector with a context less than or equal to that of the function to be convolved.

2) If the first component of the kernel vector is not equal to zero, multiply the first component by one half. This lessens the error from the convolution computation due to the trapezoidal rule integration formula. To accomplish this one could press:

   ... LI EVAL 1 ⊙ 0.5 SUB 1 LII ...

3) Next reflect the kernel (REFL), then store it in temporary data storage (STORE K).

4) Create the function to be convolved.

5) If the first component of the function vector is not equal to zero, multiply the first component by one half.

6) Make sure the function to be convolved is in the working register, convolve it with the appropriate operand (CONV K,J).

7) If the lower limit of integration is zero, the user may want to set the first component of the resultant vector to zero since:

$$\lim_{t\to 0}\int_0^t dt = 0.0$$

EXAMPLE:

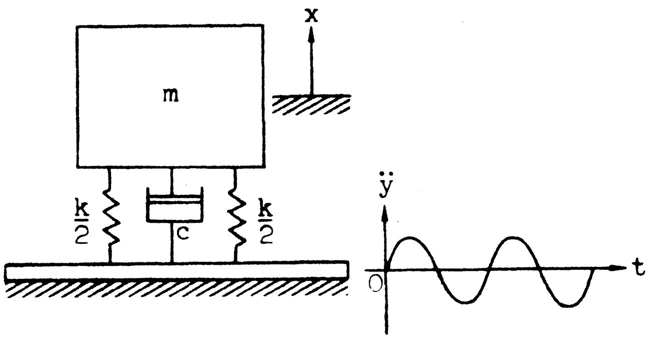
The response of a single-degree-of-freedom linear oscillator initially at rest subjected to a suddenly applied excitation provides an illustration for the application of:

$$y(t) = \int_0^t F(\tau)\ K(t - \tau)\ d\tau$$

$$K(t)\ \&\ F(t) = 0 \text{ for } t < 0$$

This special form of the superposition integral is referred to as the Duhamel integral and results from the differential equation:

$$\ddot{x} + 2\zeta\omega\dot{x} + \omega^2 x = \frac{F}{m} \qquad (1)$$



where $2\zeta\omega$ is the viscous resistance per unit mass (c/m), ω^2 is the spring rate per unit mass (k/m), x is the motion of the mass and y of the base. Such a dynamical system can be subjected to a sudden movement specified by its displacement, velocity or acceleration. The equation of motion (1) is often expressed in terms of the relative displacement between the mass and base such that $z = x - y$ and

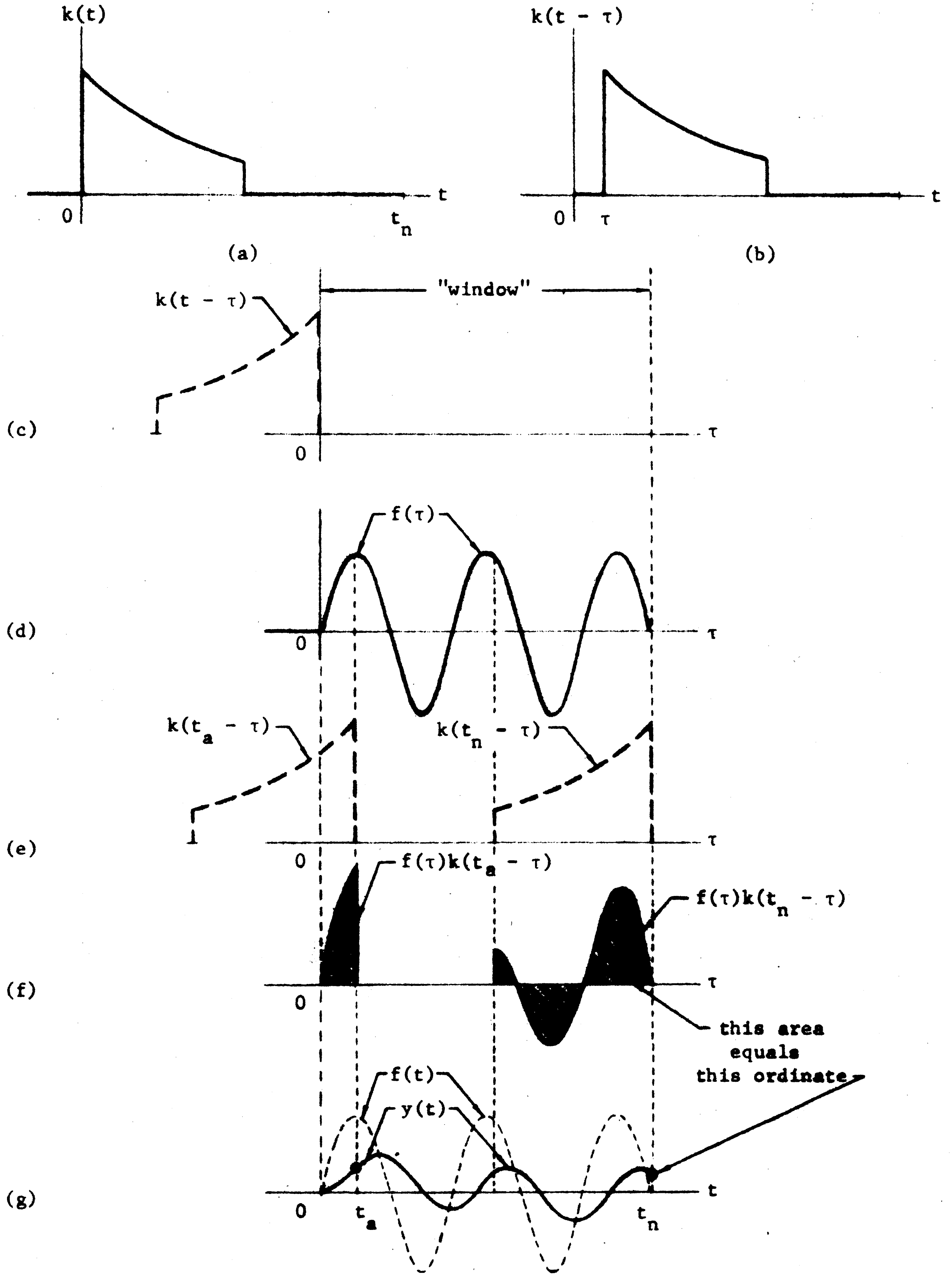
$$\ddot{z} + 2\zeta\omega\dot{z} + \omega^2 z = -\ddot{y} \qquad (2)$$

where the term F/m is replaced by $-\ddot{y}$ or the negative of the base acceleration. For an undamped system ($\zeta = 0$) initially at rest ($z_o = \dot{z}_o = 0$), the solution for the relative displacement is expressed by

$$z = -\frac{1}{\omega}\int_0^t \ddot{y}(\tau)\ \sin\omega(t - \tau)\ d\tau \qquad (3)$$

If $\ddot{y}(t)$ is defined only for a finite time, say $0 \leq t \leq t_p$, then the motion of the system after the pulse terminates at t_p becomes harmonic.

Figure 4.8.1 provides a graphical presentation to assist in visualizing the operations involved in executing the convolution of superposition integral. A given impulse (a) enters the convolution integral in the form of a weighting function $K(t - \tau)$. A "memory" function can be plotted against t with τ treated as a parameter as in (b) or, alternatively, $K(t - \tau)$ can be envisioned as a function of the input time τ with the output time t as a parameter (c). It is observed that the impulse-response-function shape appears reversed in (c) since the integration is to be carried out over the input-time variable τ. An input function, (d), is successively multiplied by the "memory" function (e) to obtain the weighted input (f), with each resulting area under the weighted input curve the value of the output signal (g) at specific output times, such as t_a and t_n. If the multiplication and input-time integration are envisioned as instantaneous operations, then the "memory" function can be considered to slide to the right along the



Convolution of an impulse response (a) with an input function (d), yielding the output function (g).

Figure 4.8.1

input waveform generating as it goes successive instantaneous values of the output signal. The "memory" or weighting function is sometimes referred to as a "window" function since the output (at a given time) is influenced only by that part of the input signal which can be "seen" as it passes through the "window." Let

$$y(t) = \int_0^t F(\tau)\ K(t - \tau)\ d\tau \qquad (4)$$

$$\begin{array}{llll} \text{where} & F(t) = \sin\ 2\pi t, & 0 \leq t \leq 2.50 \\ \text{and} & K(t) = \exp\ (-t), & 0 \leq t \leq 1.25 \\ & = 0 & \ t > 1.25 \end{array}$$

An online solution invoking the convolution integral module for Equation (4) is:

```
LII   CTX   51   ID   ⊕   1   ⊘   2   ⊙   1.25   STORE T
NEG   EXP   ⊙   (DIFF   T)   LI   EVAL   1   ⊙   0.5   SUB   1
LII   REFL   STORE   K   CTX   101   LOAD   0   LOAD   K   STORE   K
ID   ⊕   1   ⊘   2   ⊙   2.5   STORE   T   LOAD   -1   ARG
⊙   2T   SIN   STORE   F   CONV   K,51   STORE   P   DISPLAY
RETURN
```

Note that the function K(t - τ) is simply multiplied by Δτ and reflected to create the weighting function before engaging the integral module. Since the first component is non-zero, it is multiplied by one half. After F(t) is created, the convolution integral is enacted by the sequence CONV K,51 with the function F(t) in the β_{II} register. The designation "K,51" dictates that location K contains the vector or "memory" function and that its 51st element is to be initially aligned with the 1st element of F(t) as illustrated in (c) and (d) prior to commencing the sequence of operators. The convolution module will successively multiply and shift the weighting function across the length of F(τ) as shown in (e) and (f) thereby generating the output function y(t) as sought and depicted in (g) within the defined "window."

4.8.2 OPERATOR DEFINITIONS FOR LEVEL II COMPLEX (MOLSF)

The operands for the level II CMPLX operators are previously defined complex vectors S, complex constant vectors "r_1, r_2", (representing $r_1 + ir_2$), or the contents of the (α_{II}, β_{II}) register. The results of the operations are always complex vectors put into the (α_{II}, β_{II}) working register.

⊕ , ⊖ , ⊙ , ⊘

followed by S or "r_1, r_2", performs the indicated complex combination of the operand vector with the complex vector in the (α_{II}, β_{II}) register. In division, if any or all components of the operand vector are 0 + 0i, the quotient will be set to 0 + 0i for those components.

<u>PWR</u>

followed by S or "r" raises the contents of each component of the (α_{II}, β_{II}) working register to the specified power and places the result in the (α_{II}, β_{II}) working register. If <u>PWR</u> is followed by S, then the contents of the vector S are used as the exponent.

<u>LS</u>

shifts each component (α_k, β_k) of the (α_{II}, β_{II}) register into the (α_{k-1} β_{k-1}) position, placing the component into the last position.

$$(\alpha_{II}, \beta_{II}) = [(\alpha_2, \beta_2), (\alpha_3, \beta_3), \ldots, (\alpha_n, \beta_n), (\alpha_1, \beta_1)]$$

<u>LS</u> K left shifts the α_{II} register K times. <u>LS</u> K_1, K_2 shifts the α_{II} and β_{II} registers separately, the α_{II} register K_1times and the β_{II} register K_2 times. LS , K left shifts the β_{II} register K times. (K a level 0 operand.)

<u>RS</u>

shifts each component (α_k, β_k) of the (α_{II}, β_{II}) register into the

$(\alpha_{k+1}, \beta_{k+1})$ position, placing the last component into the first position.

$$(\alpha_{II}, \beta_{II}) = [(\alpha_n, \beta_n), (\alpha_1, \beta_1), (\alpha_2, \beta_2), \ldots, (\alpha_{n-1}, \beta_{n-1})]$$

RS K right shifts the α_{II} register K times. RS K_1, K_2 shifts the α_{II} and β_{II} registers separately, the α_{II} register K_1 times and the β_{II} register K_2 times. RS , K right shifts the β_{II} register K times. (K a level 0 operand.)

ENL

doubles the mantissa of each component of the α_{II} and β_{II} registers, for display purposes, and decrements the binary scale of each register by 1.

ENL K enlarges the α_{II} register K times. ENL K_1, K_2 enlarges the α_{II} and β_{II} registers separately, the α_{II} register K_1 times and the β_{II} register K_2 times. ENL , K enlarges the β_{II} register K times. (K a level 0 operand.)

CON

halves the mantissa of each component of the α_{II} and β_{II} registers, for display purposes, and increments the binary scale of each register by 1.

CON K contracts the β_{II} register K times. CON K_1, K_2 contracts the α_{II} and β_{II} registers separately, the α_{II} register K_1 times and the β_{II} register K_2 times. CON , K contracts the β_{II} register K times. (K a level 0 operand.)

NOTE: For LS, RS, ENL, and CON a negative operand implies the inverse operator, For example, RS , -7 is equivalent to LS , 7.

ID — places a unit square centered at the origin with vertices at (1, 1), (-1, 1), (-1, -1), (1, -1) in the $(\alpha_{II}, \beta_{II})$ register.

ID . — places a unit circle centered at the origin in the $(\alpha_{II}, \beta_{II})$ register.

[In the following, if the operator key is followed immediately by S or "r_1, r_2" , the operand is the vector in the level II CMPLX storage location S, or the constant complex vector $r_1 + \mathbf{i}r_2$, respectively. If the next key is not one of these, the operand is the complex vector $\alpha_{II} + i\beta_{II}$. The result is always put into the $(\alpha_{II}, \beta_{II})$ register.]

SQ — squares each component of the operand.

SQRT — takes the square root of each component of the operand, using the branch of square root such that the argument of the answer is half the argument (defined by ARG) of the original function.

NEG — takes the complex conjugate of each component of the operand.

INV — takes the complex reciprocal of each component of the operand.

DIFF — forms the complex forward difference of the operand, extrapolating to get the final component of the result.

SUM — forms the running sum of the complex values in the operand, storing the subtotals in the corresponding components of the result.

PROD — forms the running product of the complex values in the operand, storing the subproducts in the corresponding components of the result.

REFL — reflects the operand vector about the 45° line; thus, it interchanges the real and imaginary parts of the operand.

MAX

makes a constant complex vector whose real part is the maximum of the real parts of the operand and whose imaginary part is the maximum of the imaginary parts of the operand.

MOD

evaluates the modulus of each component of the operand vector, stores the answer in the α_{II} register, places zeros in the β_{II} register.

SIN, COS, LOG, EXP, ATAN

performs the indicated operation componentwise, using the values obtained from ARG whenever a function has branches (i.e. LOG and ATAN). If the lower keyboard + follows an operation using ARG then the branch is obtained from ARG + . LOG of zero gives -183.846 + 0i.

DEL

executes LII REAL DEL on the real and imaginary parts of the operand separately, then puts their product into the α_{II} register and sets the β_{II} register to zero.

ARG or
ARG -

computes the argument of each component of the operand vector, assuming that the argument of the first point lies in the interval $(-\pi , \pi)$. Puts the result in the α_{II} resister and sets the βII register to zero. The following values are true arguments based on that branch cut.

ARG +

same as ARG except that the interval for the first component is $(0, 2\pi)$.

4.8.3 ADDITIONAL COMMENTS ON LEVEL II

DISPLAY

In order to generate a display on level II REAL the α_{II} register must contain the desired set of X coordinates in the form of the ID vector or some similar function. Most operations on level II REAL affect only the β_{II} register and leave α_{II} unchanged. An important exception to this is SUB which operates

4.9 MATHEMATICAL OPERATORS FOR LEVEL III

Level III operators provide the ability for a user to manipulate arrays. The number of elements or dimension of an array is restricted by the arrangements made with the Computer Center when the user number is set up.

Arrays are stored on level III under the alphabetic keys, A through Z and α through ω. As discussed earlier the dimensions can be changed by the use of the <u>CTX</u> key.

Level III operators and data are column oriented. Therefore, level III overhead is minimized when the number of rows is greater than the number of columns [i.e., $n > m$ in an (n,m) array].

4.9.1 OPERATOR DEFINITIONS FOR LEVEL III REAL (MOLSF)

Throughout the description of the level III REAL and CMPLX operators it is assumed that the arrays needed have previously been defined and are available for use. The general notation applied to the description of level I is again used. Note that S, which represents an alphabetic key, now implies an array

$$S = \begin{pmatrix} s_{11} & s_{12} & \cdots & s_{1m} \\ s_{21} & \cdots & \cdots & s_{2m} \\ \vdots & & & \vdots \\ s_{n1} & \cdots & \cdots & s_{nm} \end{pmatrix}$$

and "r", which represents a real number, now defines a constant array of n,m components.

$\oplus$, $\ominus$, $\odot$, $\oslash$

followed by S or "r" performs the indicated operation component by component using the array in the β_{III} register and the array S or "r". Let

$$\begin{pmatrix} \beta_{1,1} & \cdots & \beta_{1,m} \\ \vdots & & \vdots \\ \beta_{n,1} & \cdots & \beta_{n,m} \end{pmatrix}$$

denote the contents of β_{III} before any of these operations. Then the result, in β_{III}, will be as follows:

$\oplus$ S :

$$\beta_{III} = \begin{pmatrix} \beta_{1,1}+s_{1,1}, & \beta_{1,2}+s_{1,2} & \cdots & \beta_{1,m}+s_{1,m} \\ \beta_{2,1}+s_{2,1} & & & \vdots \\ \vdots & & & \vdots \\ \beta_{n,1}+s_{n,1} & \cdots\cdots\cdots & & \beta_{n,m}+s_{n,m} \end{pmatrix}$$

The results are formed in a similar fashion for $\ominus$, $\odot$, $\oslash$.

PWR

followed by S or "r" raises the contents of each component of the β_{III} working register to the specified power and places the result in the β_{III} working register. If PWR is followed by S, then the contents of the array S are used as the exponents.

LS

left shifts each column of the β_{III} working register. Each component $\beta_{i,j}$ in the β_{III} working register is shifted into the $\beta_{i,j-1}$ position. The components in the first column are shifted into the last column.

$$\beta_{III} = \begin{pmatrix} \beta_{1,2}, & \beta_{1,3} & \cdots & \beta_{1,m}, & \beta_{1,1} \\ \beta_{2,2} & & & \beta_{2,m}, & \beta_{2,1} \\ \vdots & & & \vdots & \vdots \\ \beta_{n,2} & & \cdots & \beta_{n,m}, & \beta_{n,1} \end{pmatrix}$$

LS K repeats LS K times. (K a level 0 operand.)

RS

right shifts each column of the β_{III} working register. Each component $\beta_{i,j}$ in the β_{III} working register is shifted into the $\beta_{i,j+1}$ position. The components in the last column are shifted into the first column.

$$\beta_{III} = \begin{pmatrix} \beta_{1,m}, & \beta_{1,1}, & \beta_{1,2} & \cdots & \beta_{1,m-1} \\ \beta_{2,m}, & \beta_{2,1} & & & \cdot \\ \vdots & \vdots & & & \vdots \\ \beta_{n,m}, & \beta_{n,1} & & \cdots & \beta_{n,m-1} \end{pmatrix}$$

RS K repeats RS K times. (K a level 0 operand.)

UP

shifts each row of the β_{III} working register up to the next row. Each component $\beta_{i,j}$ in the β_{III} working register is shifted into the $\beta_{i-1,j}$ position. The components in the first row are shifted into the last row.

$$\beta_{III} = \begin{pmatrix} \beta_{2,1}, & \beta_{2,2} & \cdots & \beta_{2,m} \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ \beta_{n,1}, & \beta_{n,2} & \cdots & \beta_{n,m} \\ \beta_{1,1}, & \beta_{1,2} & \cdots & \beta_{1,m} \end{pmatrix}$$

UP K repeats UP K times. (K a level 0 operand.)

DOWN

shifts each row of the β_{III} working register down to the next row. Each component $\beta_{i,j}$ in the β_{III} working register is shifted into the $\beta_{i+1,j}$ position. The components in the last row are shifted into the first row.

$$\beta_{III} = \begin{pmatrix} \beta_{n,1}, & \beta_{n,2} & \cdots & \beta_{n,m} \\ \beta_{1,1}, & \beta_{1,2} & \cdots & \beta_{1,m} \\ \vdots & & & \cdot \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ \beta_{n-1,1} & \cdots\cdots & & \beta_{n-1,m} \end{pmatrix}$$

DOWN K repeats DOWN K times. (K a level 0 operand.)

NOTE: For LS, RS, UP, and DOWN a negative operand implies the inverse operator. For example, UP -7 is equivalent to DOWN 7.

REFL

transposes the β_{III} array, i.e., $\beta_{ij} = \beta_{ji}$.

REFL C

reverses the order of the components in each column of the β_{III} working register.

REFL R

reverses the order of the components in each row of the β_{III} working register.

ID or
ID C

If the working register is of dimension n,m ID places m vectors consisting of n equally spaced values from -1 to +1, (beginning with -1, ending with +1) in the columns of the β_{III} register.

ID R

is the same as ID C except that ID R places the vectors in rows rather than columns.

(In the following, if the operator key is followed immediately by S or "r", the operand is the array in the real storage location S, or the constant array r, respectively. If the next key is not one of these, the operand is the real array in β_{III}. The result is always put into the β_{III} working register.)

ATAN, LOG, EXP, SIN, COS

performs the indicated operation component by component on the operand array. LOG of zero gives -183.846.

SQ

squares each component of the operand array.

SQRT

takes the square root of each component of the operand array.

INV — takes the reciprocal of each component of the operand array.

NEG — negates each component of the operand array.

MOD — takes the absolute value of each component of the operand array.

ARG — assigns the value zero to all non-negative components, the value $\pi = 3.14159$ to all negative components of the operand array.

MAX — sets each column equal to the maximum component in the corresponding column of the operand array.

DEL — identifies zero and sign changes in each column of the operand array as follows:

if $s_{i,j} = 0$ then $\beta_{i,j} = 1$

if $(s_{i,j})\ (s_{i+1,j}) < 0$ then

$\beta_{i,j} = 1$ if $|s_{i,j}| \leq |s_{i+1,j}|$

$\beta_{i+1,j} = 1$ if $|s_{i,j}| > |s_{i+1,j}|$

all other $\beta_{i,j} = 0$ $\quad i = 1, \ldots, n$

$j = 1, \ldots, m$

DIFF — computes the forward difference of each column of the operand array, performing a second order extrapolation to supply the last component in the result.

SUM — computes the running summation of each column of the operand.

PROD — computes the running product of each column of the operand.

4.9.2 OPERATOR DEFINITIONS FOR LEVEL III COMPLEX (MOLSF)

The operands for the level III CMPLX operators are previously defined complex arrays S, complex constant arrays "r" (representing $r_1 + ir_2$) or the contents of the (α_{III}, β_{III}) register. The results of the operations are always complex arrays put into the (α_{III}, β_{III}) register.

$\oplus$, $\ominus$, $\odot$, $\oslash$ — followed by S or "r" performs the indicated complex computation component by component on the operand array with the complex array in the (α_{III}, β_{III}) register. In division, if any or all components of the operand array are 0 + 0i, the quotient will be set to 0 + 0i for those components.

PWR — followed by S or "r" raises the contents of each component of the (α_{III}, β_{III}) working register to the specified power and places the result in the (α_{III}, β_{III}) working register. If PWR is followed by S, then the contents of the array S are used as the exponents.

LS — left shifts each column of the (α_{III}, β_{III}) working register. Each component $\alpha_{i,j}$, $\beta_{i,j}$ is shifted into the $\alpha_{i,j-1}$, $\beta_{i,j-1}$ position. The components in the first column are shifted into the last column.

LS K repeats LS K times. LS K_1, K_2 shifts the α_{III} and β_{III} registers separately, the α_{III} register K_1 times and the β_{III} register K_2 times. LS ,K shifts the β_{III} register K times. (K a level 0 operand.)

RS — right shifts each column of the (α_{III}, β_{III}) working register. Each component $\alpha_{i,j}$, $\beta_{i,j}$ is shifted into the $\alpha_{i,j+1}$, $\beta_{i,j+1}$ position. The components in the last column are shifted into the first column.

RS K repeats RS K times. RS K_1, K_2 shifts the α_{III} and β_{III} registers separately, the α_{III} register K_1 times and the β_{III} register K_2 times. RS ,K

shifts the β_{III} register K times. (K a level 0 operand.)

UP

shifts each row of the (α_{III}, β_{III}) working register up to the next row. Each component $\alpha_{i,j}$, $\beta_{i,j}$ is shifted into the $\alpha_{i-1,j}$, $\beta_{i-1,j}$ position. The components in the first row are shifted into the last row.

UP K repeats UP K times. UP K_1,K_2 shifts the α_{III} and β_{III} registers separately, the α_{III} register K_1 times and the β_{III} register K_2 times. UP ,K shifts the β_{III} register K times. (K a level 0 operand.)

DOWN

shifts each row of the (α_{III}, β_{III}) working register down to the next row. Each component $\alpha_{i,j}$, $\beta_{i,j}$ is shifted into the $\alpha_{i+1,j}$, $\beta_{i+1,j}$ position. The components in the last row are shifted into the first row.

DOWN K repeats DOWN K times. DOWN K_1,K_2 shifts the α_{III} and β_{III} registers separately, the α_{III} register K_1 times and the β_{III} register K_2 times. DOWN ,K shifts the β_{III} register K times. (K a level 0 operand.)

ID

If the (α_{III}, β_{III}) register is dimensioned (n,m), ID places m complex vectors of context n each of which forms a unit square centered at the origin with vertices at (1,1) (-1,1), (-1,-1), (1,-1) in the (α_{III}, β_{III}) register.

ID •

If the (α_{III}, β_{III}) register is dimensioned (n,m), ID places m complex vectors of context n each of which forms a unit circle centered at the origin in the (α_{III}, β_{III}) register.

(In the following, if the operator key is followed immediately by S or "r,r_2", the operand is the array in the level III complex storage location S, or the constant complex array $r_1 + ir_2$, respectively. If the next key is not one of these, the operand is the complex array $\alpha_{III} + i\beta_{III}$. The result is always put into the (α_{III}, β_{III}) working register.)

ATAN, SIN, COS, LOG, EXP — performs the indicated operation component by component using the values obtained from ARG whenever a function has branches. If the lower keyboard + follows an operation using ARG then the branch is obtained from ARG +. LOG of zero gives -183.846 + 0i.

SQ — squares each component of the operand array.

SQRT — takes the square root of each component of the operand array using the branch of square root such that the argument of the answer is half the argument (defined by ARG) of the original function.

NEG — takes the complex conjugate of each component of the operand array.

INV — takes the complex reciprocal of each component of the operand array.

MOD — evaluates the modulus of each component of the operand array, stores the answer in the α_{III} register, places zeros in the β_{III} register.

ARG or
ARG - — computes the argument of each component of the operand array, assuming that the argument of the 1st point lies in the interval (-π, π). Puts the result in the α_{III} register and sets β_{III} to zero.

ARG + — same as ARG except that the interval for the first component is (0, 2π).

MAX — sets each column equal to the maximum value in the corresponding column of the operand array separately for the real and imaginary components.

REFL — interchanges real and imaginary parts of the operand array.

DEL — executes LII REAL DEL on the real and imaginary parts of each column of the operand array separately, then puts the product of the corresponding parts into the α_{III} register and sets the β_{III} register to zero.

DIFF — forms the complex forward difference of each column of the operand array, extrapolating to get the final component of each result.

SUM — forms the running sum of the complex values of each column in the operand array, storing the subtotals in the corresponding components of the result.

PROD — forms the running product of the complex values of each column in the operand array storing the subproducts in the corresponding components of the result.

4.10 OPERATOR DEFINITIONS FOR LEVEL 0 SUB AND EVAL (MOLSF)

Integer arithmetic as described in section 2.3 does not depend on whether the current mode is real or complex. However, the mode is significant in level 0 EVAL which extracts integer data from levels I, II, and III, and SUB, which inserts integer data into levels I, II, and III data structures. Normally, EVAL and SUB extract or insert data from the real part of a number. REAL or CMPLX precedes each definition where the distinction is material. They need not immediately precede the EVAL or SUB key, since the online system remembers whether REAL or CMPLX was last pressed. S is a storage location as defined by an alphabetic key.

EVAL CTX LIII — loads the current level III dimension into the quotient and remainder registers. The number of rows is put into the quotient register, the number of columns into the remainder register.

REAL EVAL CTX LIII S — loads the current dimension of the level III real array S. The number of rows is put into the quotient register, the number of columns into the remainder register

CMPLX EVAL CTX LIII S — loads the current dimension of the level III complex array S. The number of rows is put into the quotient register, the number of columns into the remainder register.

EVAL CTX LII or
EVAL CTX — loads the current level II context into the quotient register.

REAL EVAL CTX LII S or
REAL EVAL CTX S — loads the current context of the level II real vector S into the quotient register.

CMPLX EVAL CTX LII S
or
CMPLX EVAL CTX S

loads the current context of the level II complex vector S into the quotient register.

REAL EVAL 0

loads the value of the β_{II} working register display scale into the quotient register.

CMPLX EVAL 0

loads the value of the α_{II} working register display scale into the quotient register, and the value of the β_{II} working register display scale into the remainder register.

REAL EVAL k

extracts the contents of component K of the β_{II} working register, computes the nearest integer to this value, and places the result in the quotient register. k must be an integer constant.

CMPLX EVAL k

extracts the contents of component K of the α_{II} working register, computes the nearest integer to this value, and places the result in the quotient register. k must be an integer constant.

REAL EVAL + S

computes the least integer greater than or equal to the contents of the level I real scalar S and places it in the quotient register. If S is omitted, then the online system uses the β_I working register as the source of the datum.

CMPLX EVAL + S

computes the least integer greater than or equal to the real part of the level I complex scalar S and places it in the quotient register. If S is omitted, then the online system uses the α_I working register as the source of the datum.

REAL EVAL - S

computes the greatest integer less than or equal to the contents of the level I real scalar S and places it in the quotient register. If S is omitted, then the online system uses the β_I working register as the source of the datum.

CMPLX EVAL - S

computes the greatest integer less than or equal to the real part of the level I complex scalar S and places it in the quotient register. If S is omitted, then the online system uses the α_I working register as the source of the datum.

REAL EVAL S

computes the nearest integer to the contents of the level I real scalar S and places it in the quotient register. If S is omitted, then the online system uses the β_I working register as the source of the datum.

CMPLX EVAL S

computes the nearest integer to the real part of the level I complex scalar S and places it in the quotient register. If S is omitted, then the online system uses the α_I working register as the source of the datum.

REAL SUB 0

stores the contents of the quotient register into the β_{II} working register display scale.

CMPLX SUB 0

stores the contents of the quotient register into the α_{II} working register display scale, and the contents of the remainder register into the β_{II} working register display scale.

REAL SUB k

stores the integer in the quotient register in component k of the β_{II} working register. k must be an integer constant.

CMPLX SUB k

stores the integer in the quotient register in component k of the α_{II} working register. k must be an integer constant.

REAL SUB S — stores the integer in the quotient register in the storage location for the level I real scalar S. If S is omitted, then the online system stores the integer in the β_I working register.

CMPLX SUB S — stores the integer in the quotient register in the real part of the storage location for the level I complex scalar S. If S is omitted, then the online system stores the integer in the α_I working register.

4.11 OPERATOR DEFINITIONS FOR LEVEL V (MOLSF)

LV REAL is a level reserved for operators which are not appropriate to any other MOLSF level and as a means whereby a user with an old keyboard may perform operations such as SORT and CONV. The operators LOAD, STORE, DISPLAY, and DEL interact with a FORTRAN program thru FORTRAN subroutine calls. The calls are explained in Appendix F.

SQRT — Equivalent in operation to LII SORT.

NEG — Equivalent in operation to LII CONV.

DISPLAY jobname RETURN — Displays the status of a background job. Possible responses:

A. "jobname NOT FOUND" if the job is not in execution.

B. "jobname STEP stepname" if the job is in execution but is not currently executing the FORTRAN subroutine FOLS or TOLS.

C. "jobname ASK INPUT n" if the job is executing a "CALL FOLS" for input from the on-line terminal. "n" is the number of components requested of the terminal.

D. "jobname HAS OUTPUT n" if the job is executing a "CALL TOLS" to send output to the on-line terminal. "n" is the number of components made available to the terminal.

DISPLAY jobname ? — All activity at the terminal is suspended until "jobname" executes a CALL FOLS or TOLS. LV LOAD and STORE operators can be preceded by the sequence, thus providing synchronization with the batch job. When the job requests a transfer the succeeding keys are executed. If the job is ready when the sequence is executed, execution of keys proceeds immediately.

LOAD p jobname RETURN

Fetches data from a background job. "p" is a level 0 operand. Possible responses:

A. A and B as described under DISPLAY.

B. "jobname ASKS INPUT (m) n" if the job has requested data from the terminal. "n" components are requested; "p-m" components were successfully transferred in this LOAD operation before the request was made.

C. No response if the transfer operation was completed successfully. Data received from the background job was stored as the first "p" components of the β_{II} working register. "p" may be a positive integer; a level 0 storage location; or the key CTX, in which case the value of "p" is taken to be equal to the current context on level II. If an integer was specified to be transferred in the FORTRAN program it will become the new contents of the level 0 quotient register.

STORE p jobname RETURN

This key sequence transfers data to a background job. "p" is a level 0 operand. Possible responses:

A. A and B as described under DISPLAY.

B. "jobname HAS OUTPUT (m) n" if the job has data to transfer to the terminal. "n" components are offered; "p-m" components were successfully transferred in this STORE operation before the offer was made.

C. No response if the operation was completed successfully. The first "p" components of the β_{II} working register were transferred to the background job. "p" may be a positive integer; a level 0 storage location; or the key CTX, in which case the value of "p" is taken to be equal to the current context on level II. The level 0 quotient register is transferred to the background job if requested by that job.

DEL jobname RETURN

Terminates the background job. Possible responses:

A. A as described under DISPLAY.

B. "jobname STEP stepname" the job is currently in execution and has not issued a CALL FOLS or CALL TOLS. A job cannot be cancelled until it has executed a subroutine call to FOLS or TOLS.

C. No response if the operation was completed successfully. The job is cancelled immediately, terminating with a system completion code of 222.

CTX A,B,C,D RETURN

sets the display window for graphical display. A and B are the coordinates of the lower left corner of the window, and C and D are coordinates of the upper right corner of the window. A, B, C, and D must be floating point scalars $-1.0 \le A,B,C,D \le 1.0$. A scalar greater than one defaults to one; a scalar less than minus one defaults to minus one.

CTX RETURN

returns to the default display window.

4.12 USE OF PARENTHESES

An additional facility which exists on levels I, II, and III is the use of parentheses to specify as an operand an expression which must be computed, thus bringing the programming language much closer to the user's "pencil-and-paper" language. For example, to compute sin X ($-2\pi \leq X \leq 2\pi$), one could use, on LII REAL,

```
SIN (ID ⊙ 6.28)
```

to effect the same computation as

```
ID ⊙ 6.28 SIN
```

Parentheses are extremely useful in both the MANUAL mode of system operation and the construction of user subroutines. For example, if the user desired to evaluate the expression (2X + 1) / (3X + 1) over the range $-1 \leq X \leq 1$ without using parentheses, the required series of button pushes would be

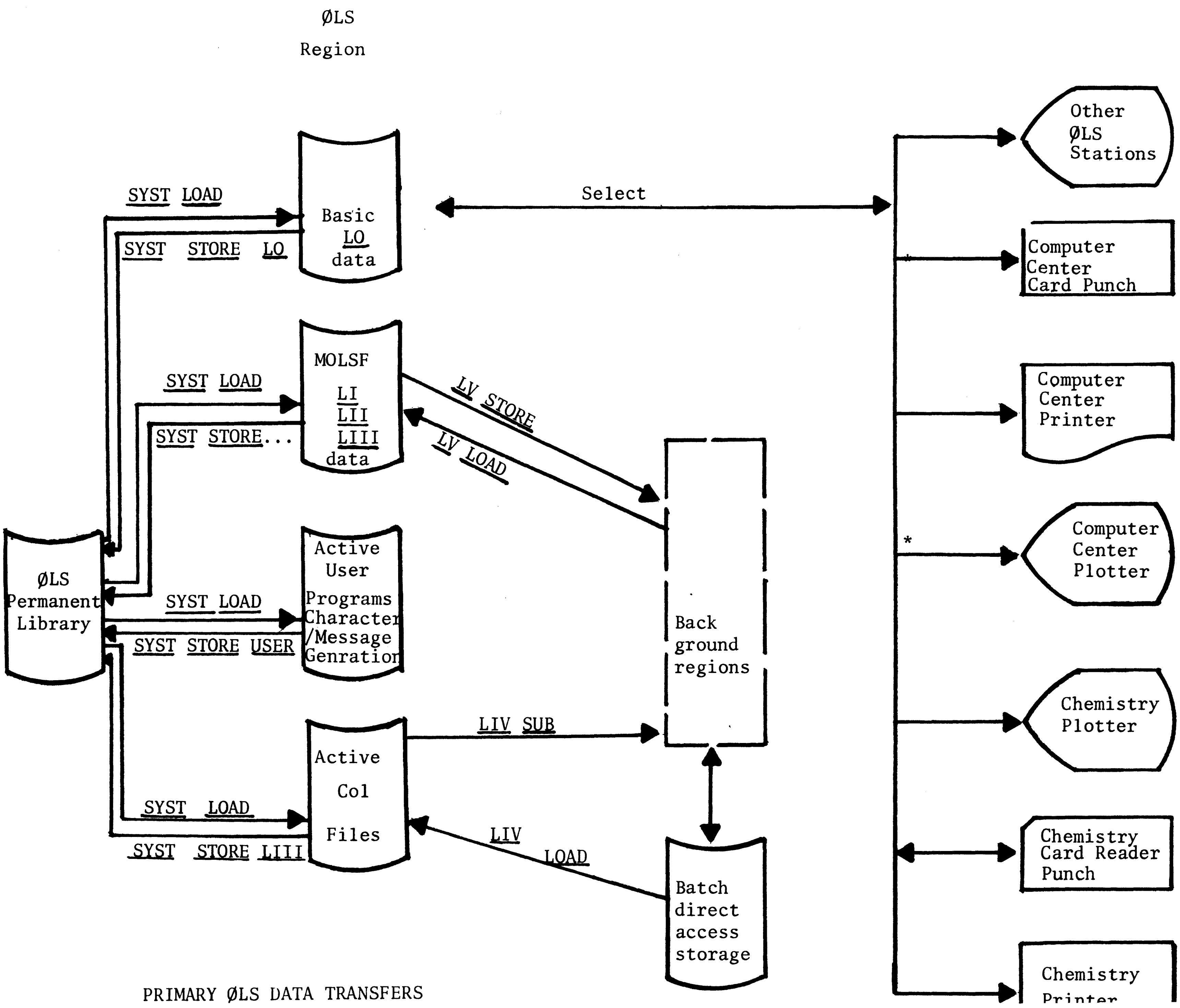
```
LII REAL ID ⊙ 3 ⊕ 1 STORE A
ID ⊙ 2 ⊕ 1 ⊘ A DISPLAY RETURN
```

The instructions ID ⊙ 3 ⊕ 1 STORE A generate the denominator term (3X + 1) and store it under A. The remaining instructions generate the numerator term (2X + 1), divide it by (3X + 1), and display the result.

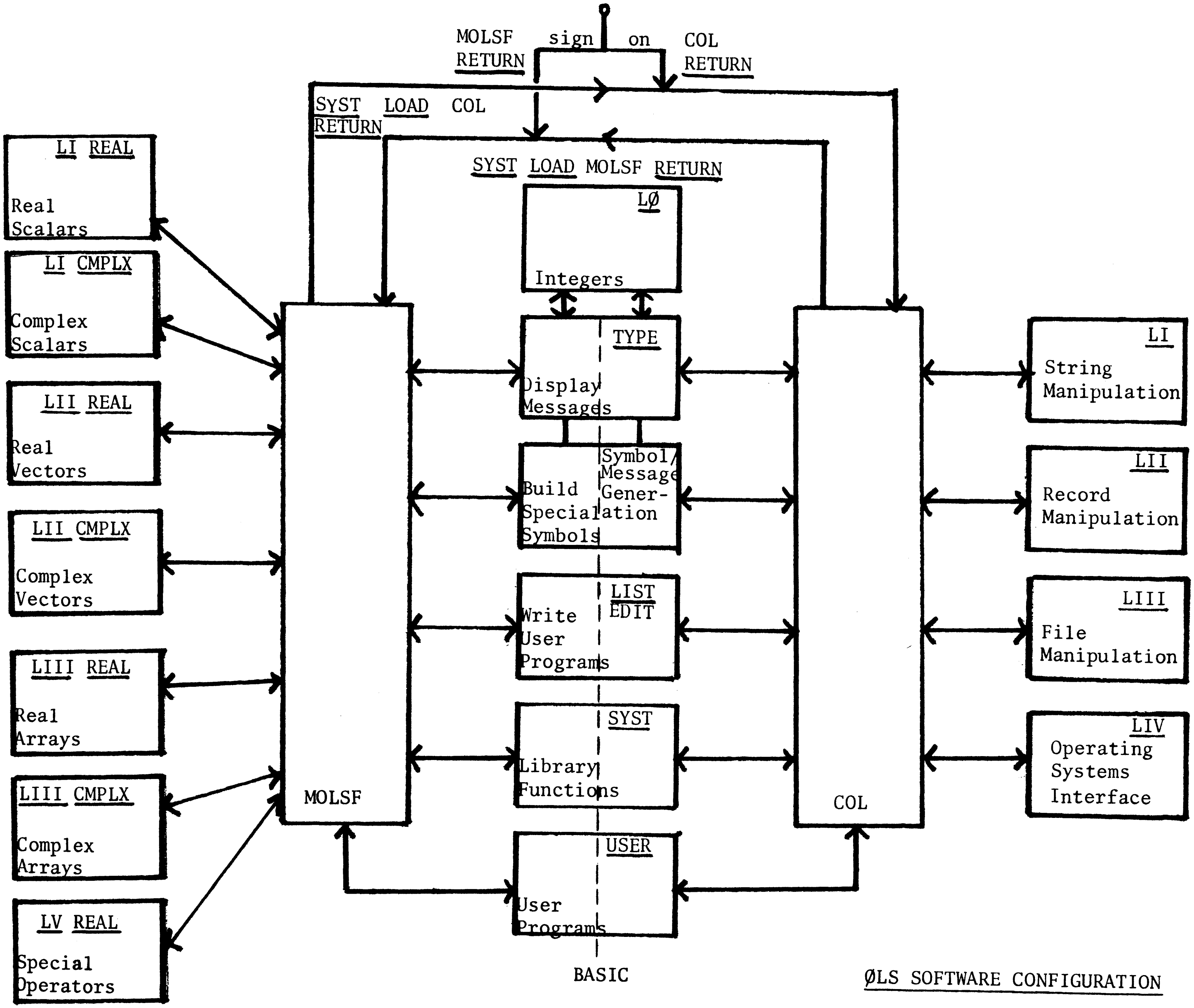
The same program using parentheses would be

Appendix C

ØLS SOFTWARE STRUCTURE & KEYBOARD DIAGRAMS



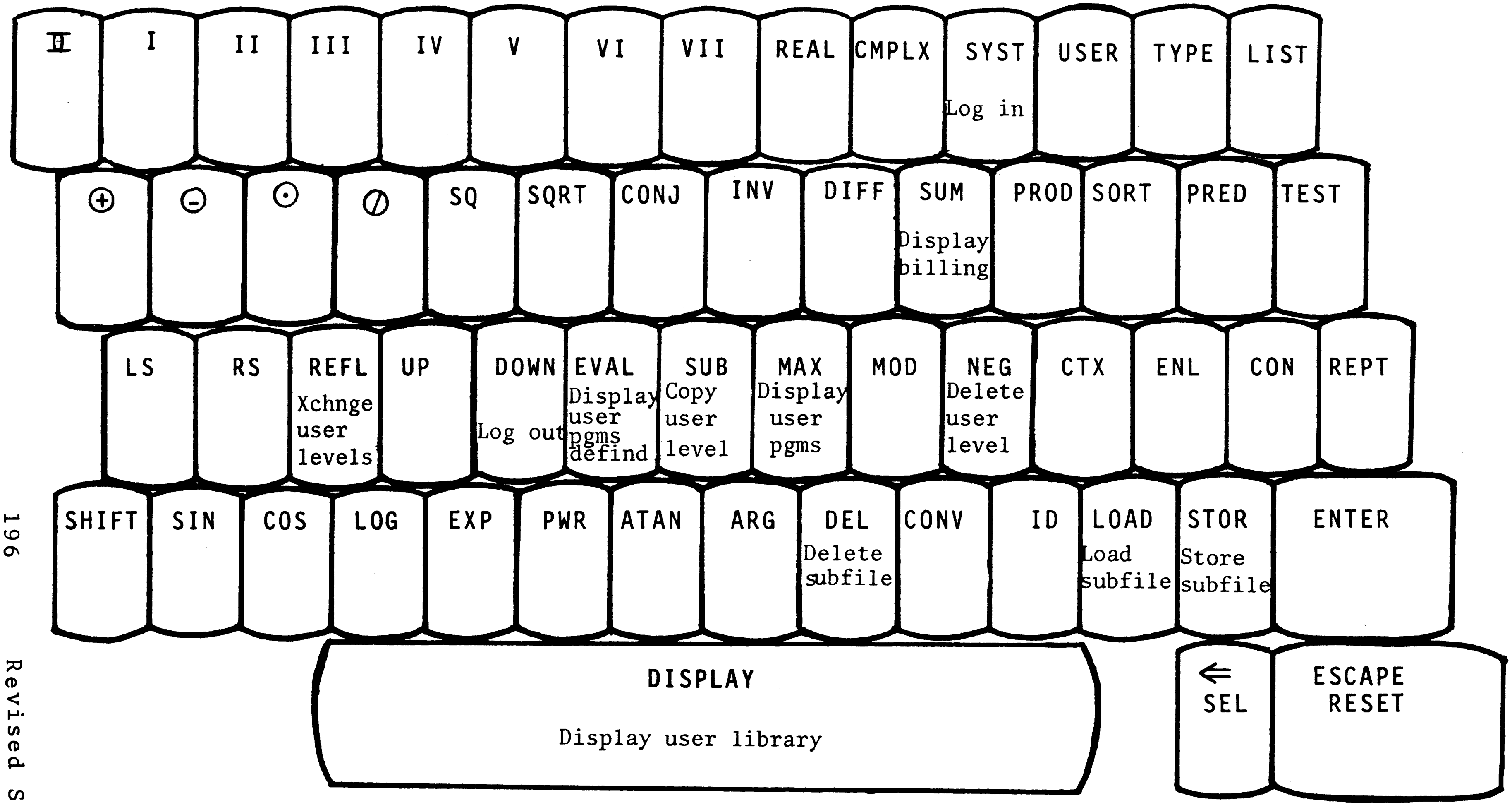
PRIMARY ØLS DATA TRANSFERS



ØLS SOFTWARE CONFIGURATION

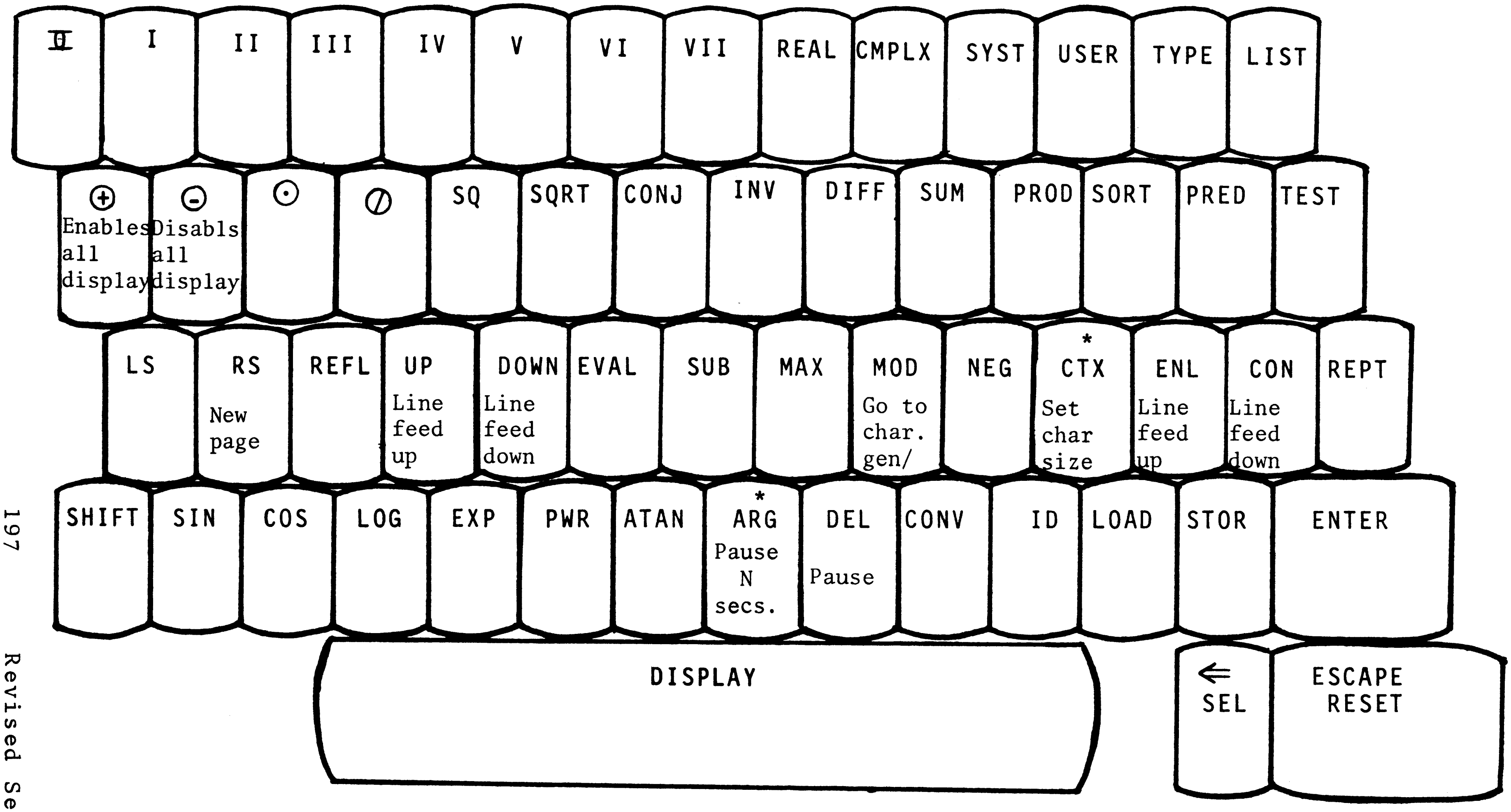
Revised Sept. 1, 1971

BASIC SYSTEM: SYST KEYBOARD



,(after display): Displays all subfiles with common names

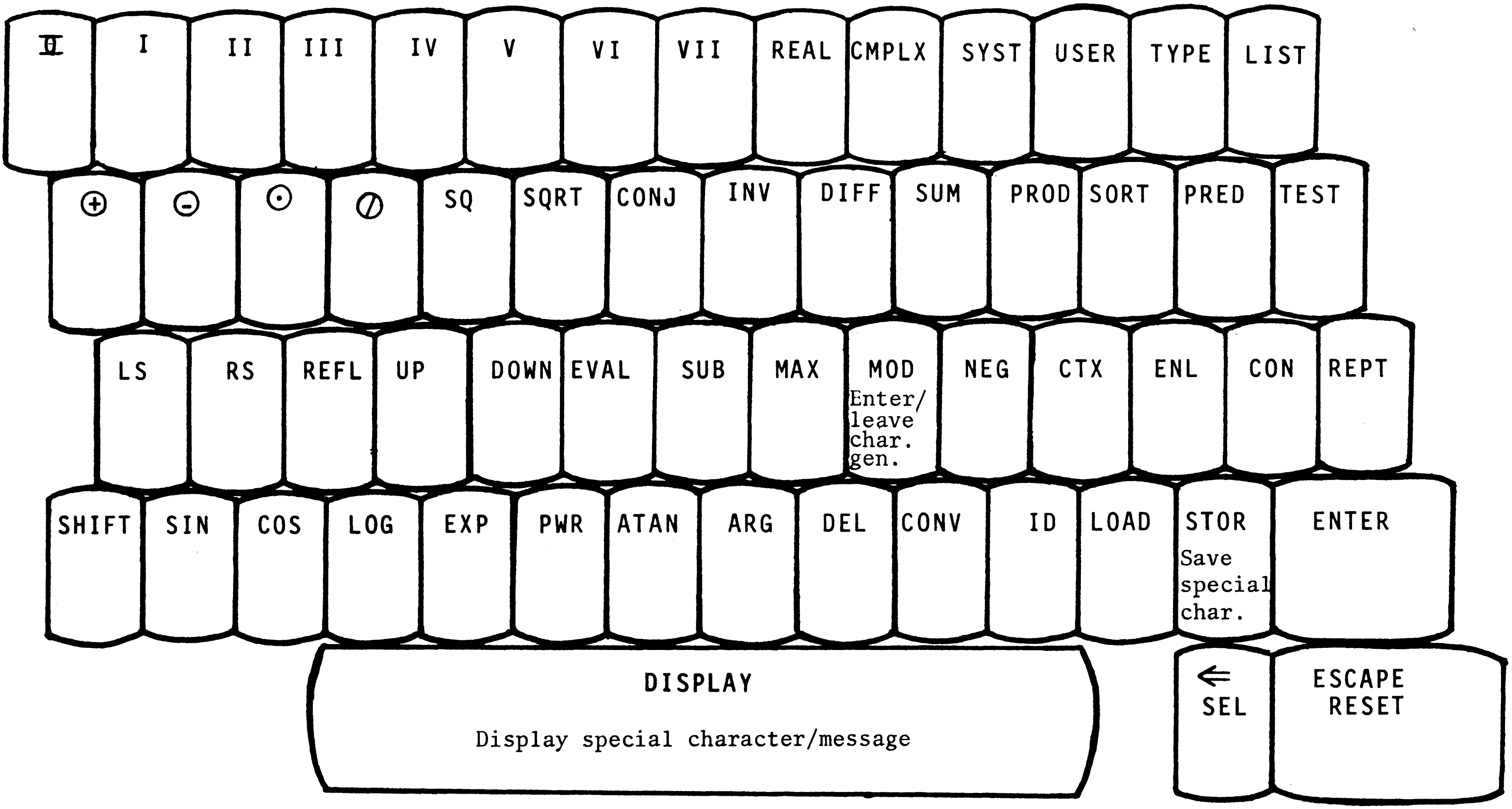
BASIC SYSTEM: TYPE KEYBOARD



Return: Carriage return.
Space : Advance to the right.
Back : Backspace to the left.
Case : Change to case level specified.

Revised Sept. 1, 1971

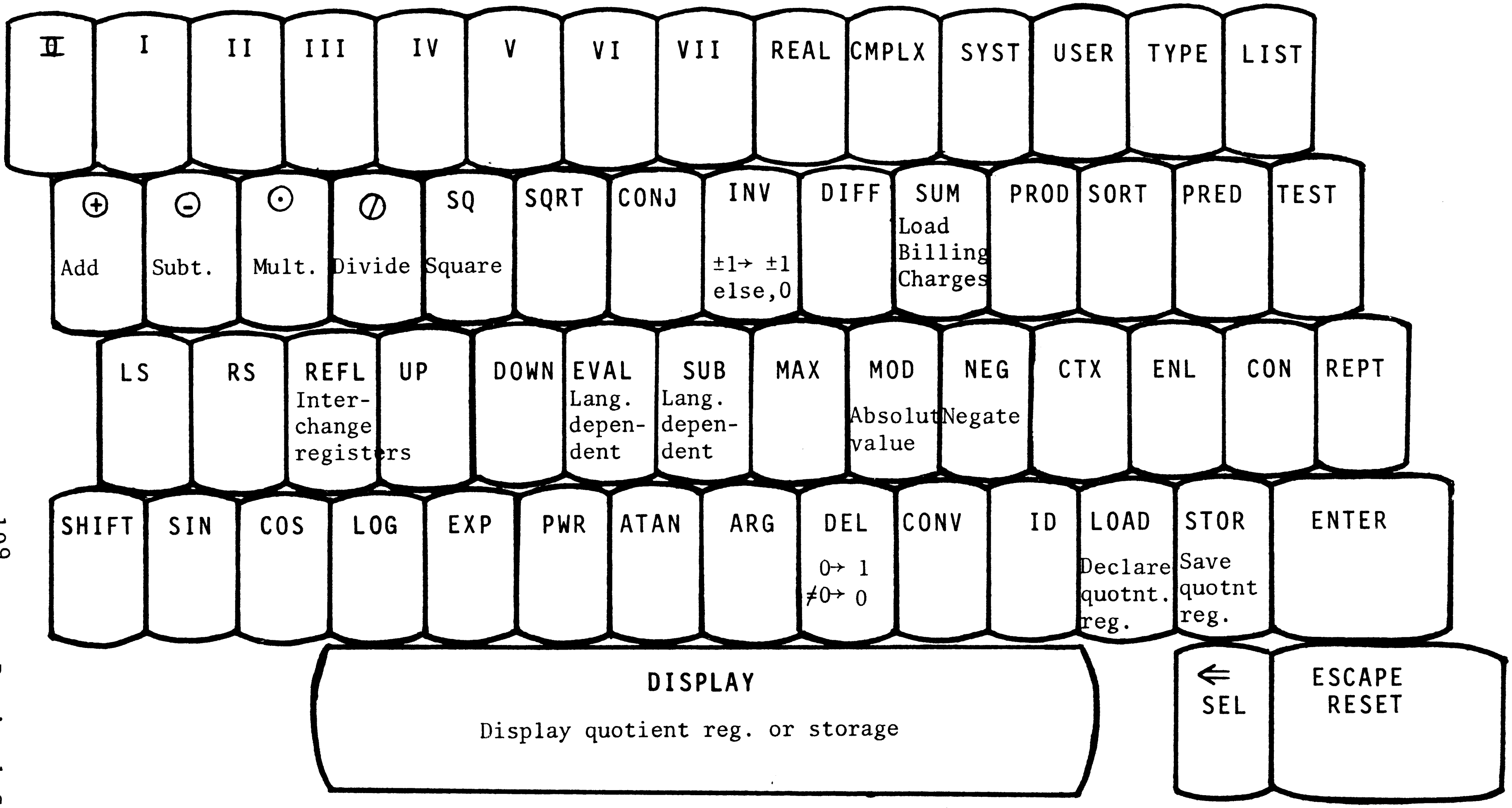
BASIC SYSTEM: MESSAGE/CHARACTER GENERATION KEYBOARD



Back : Delete last direction keypush.
Decimal point: Reposition the dot.

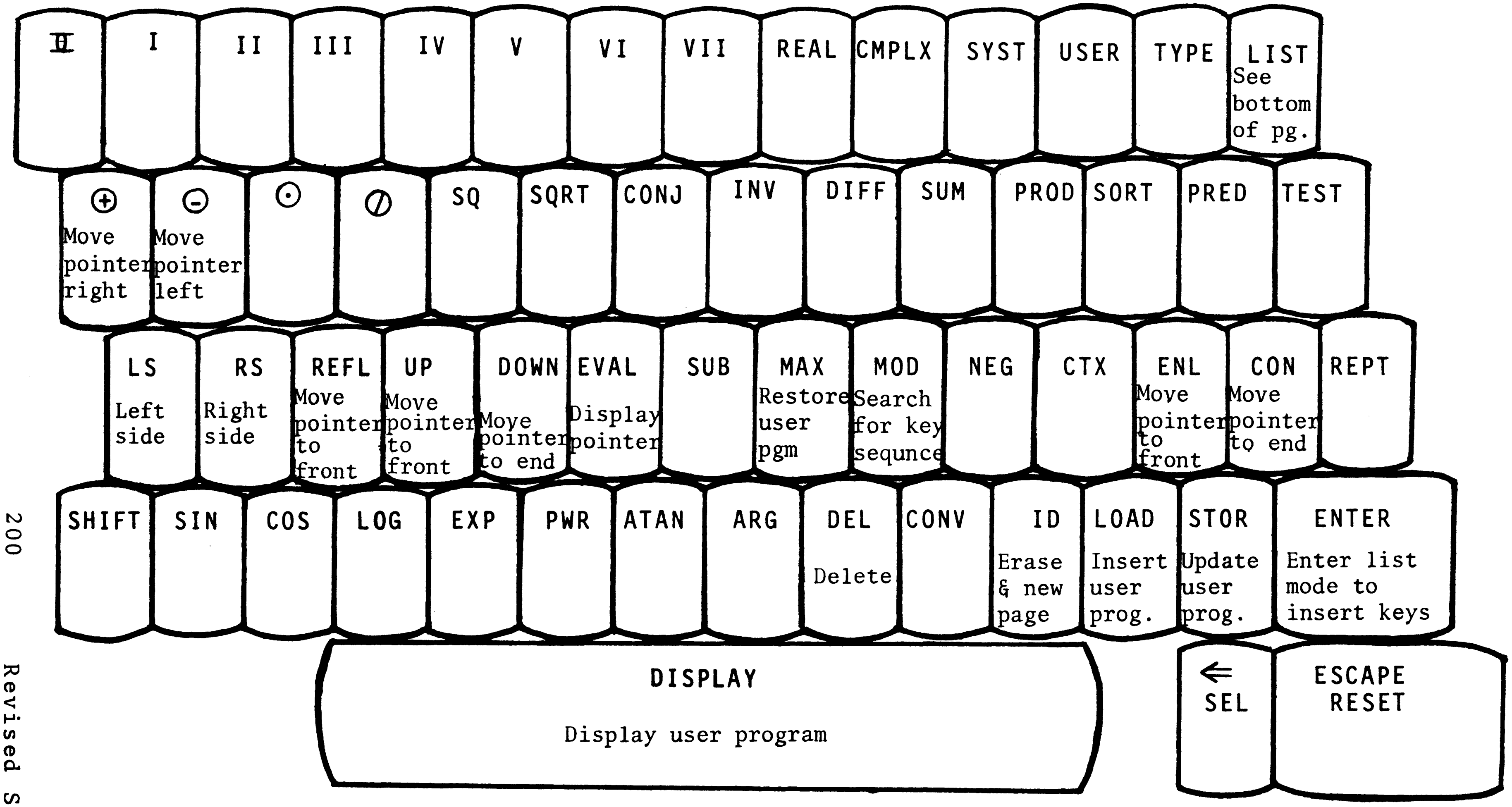
Revised Sept. 1, 1971

BASIC SYSTEM: LO KEYBOARD



Revised Sept. 1, 1971

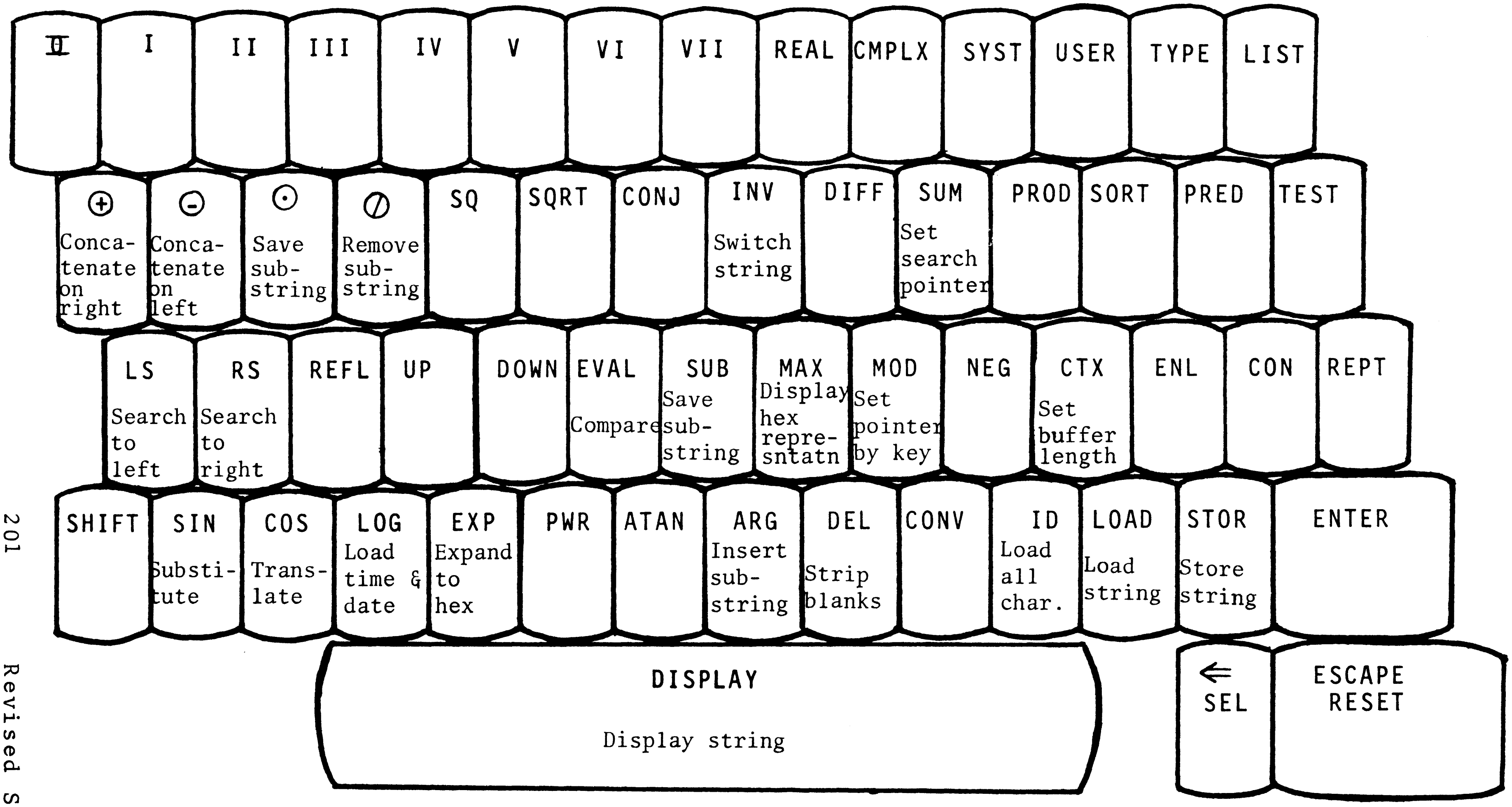
BASIC SYSTEM: EDIT LEVEL KEYBOARD



Back : Delete one key to left of pointer.
Space: Delete one key to right of pointer.
List : Leave EDIT level, enter LIST mode.

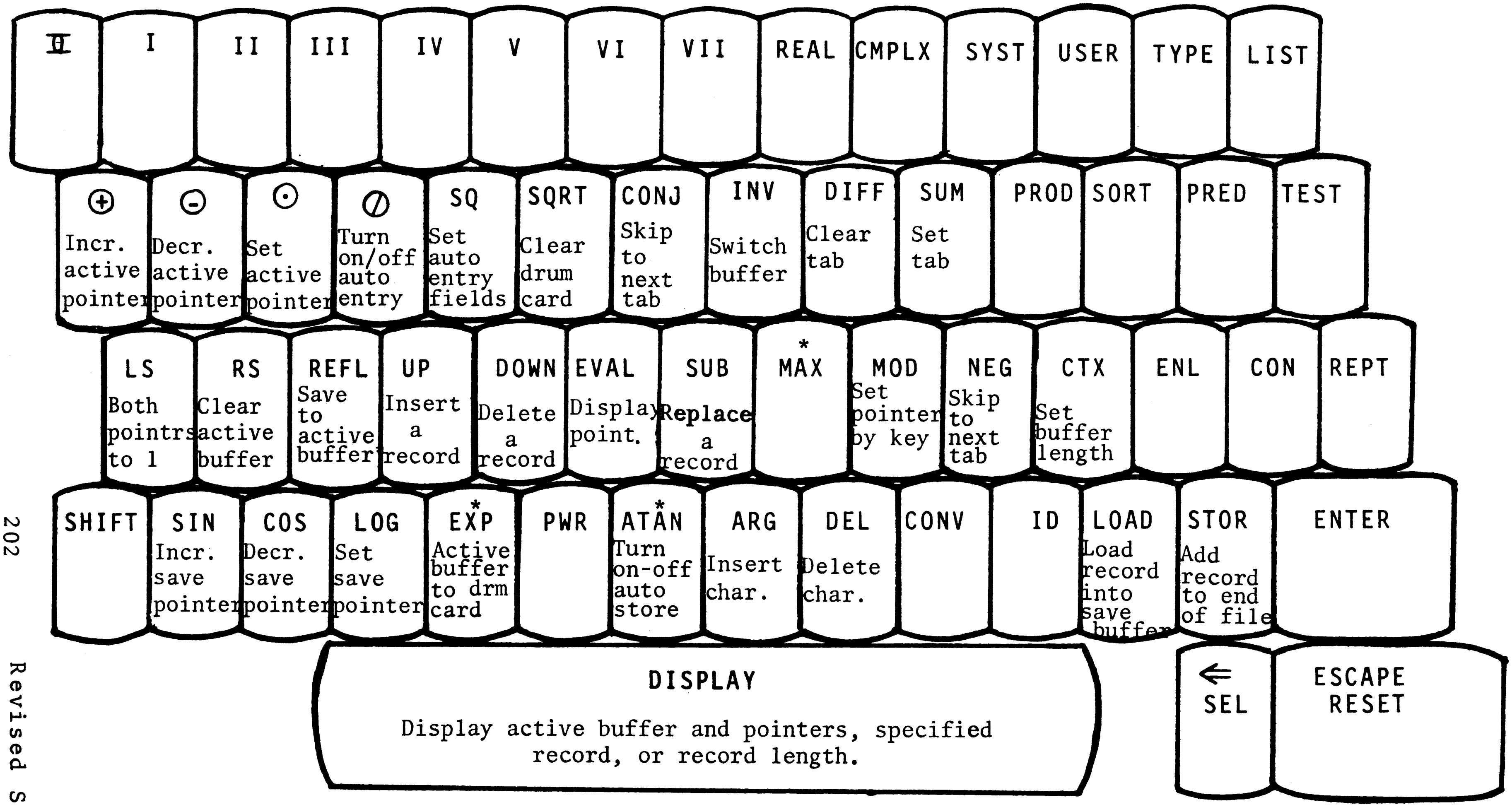
Revised Sept. 1, 1971

COL: LI KEYBOARD



Revised Sept. 1, 1971

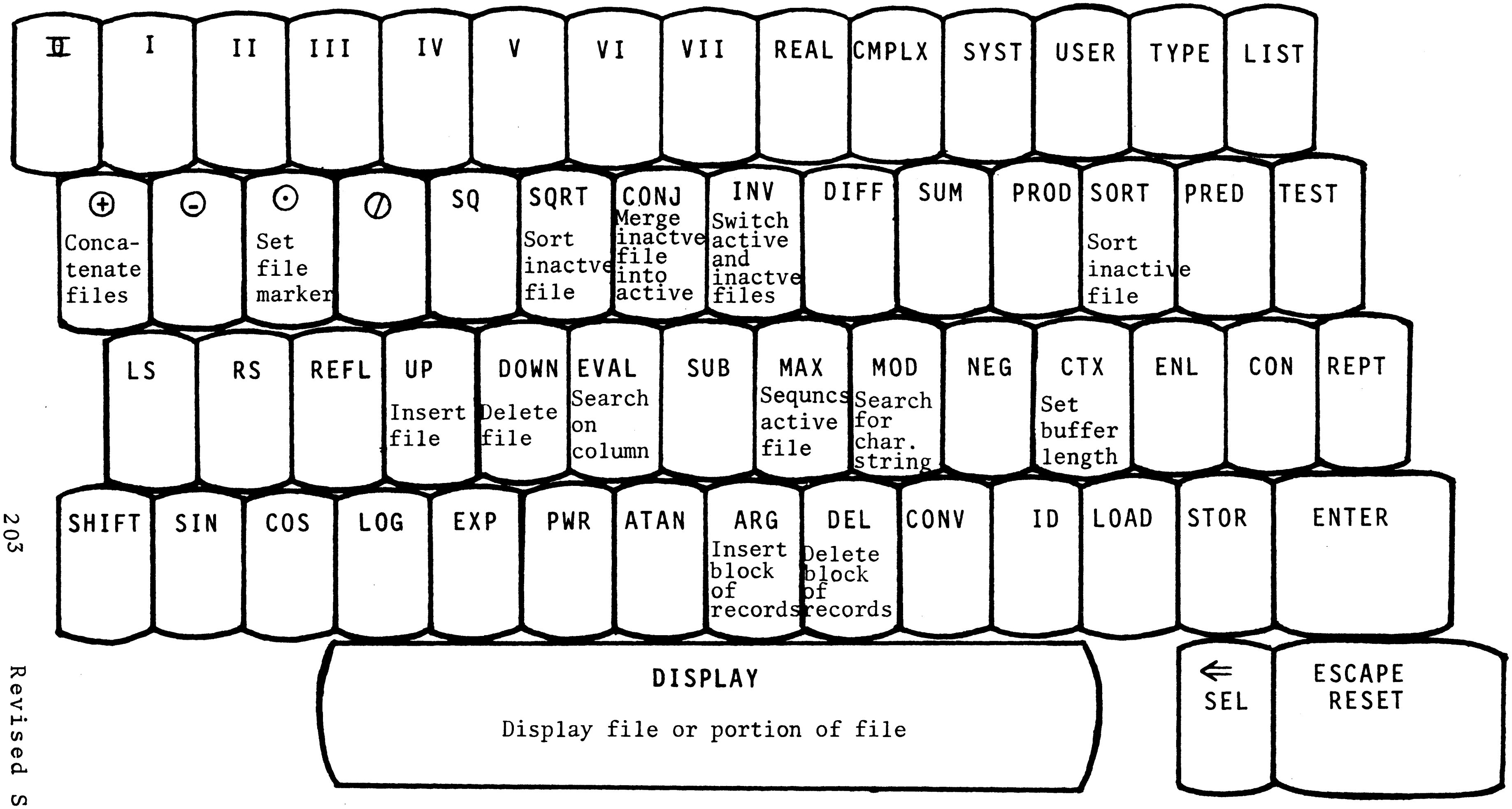
COL: LII KEYBOARD(KEYPUNCH)



Back: Display preceeding record or
 move active buffer pointer left.
Tab : Tab.
Set:: Set tabs.
Clr : Clear tabs.

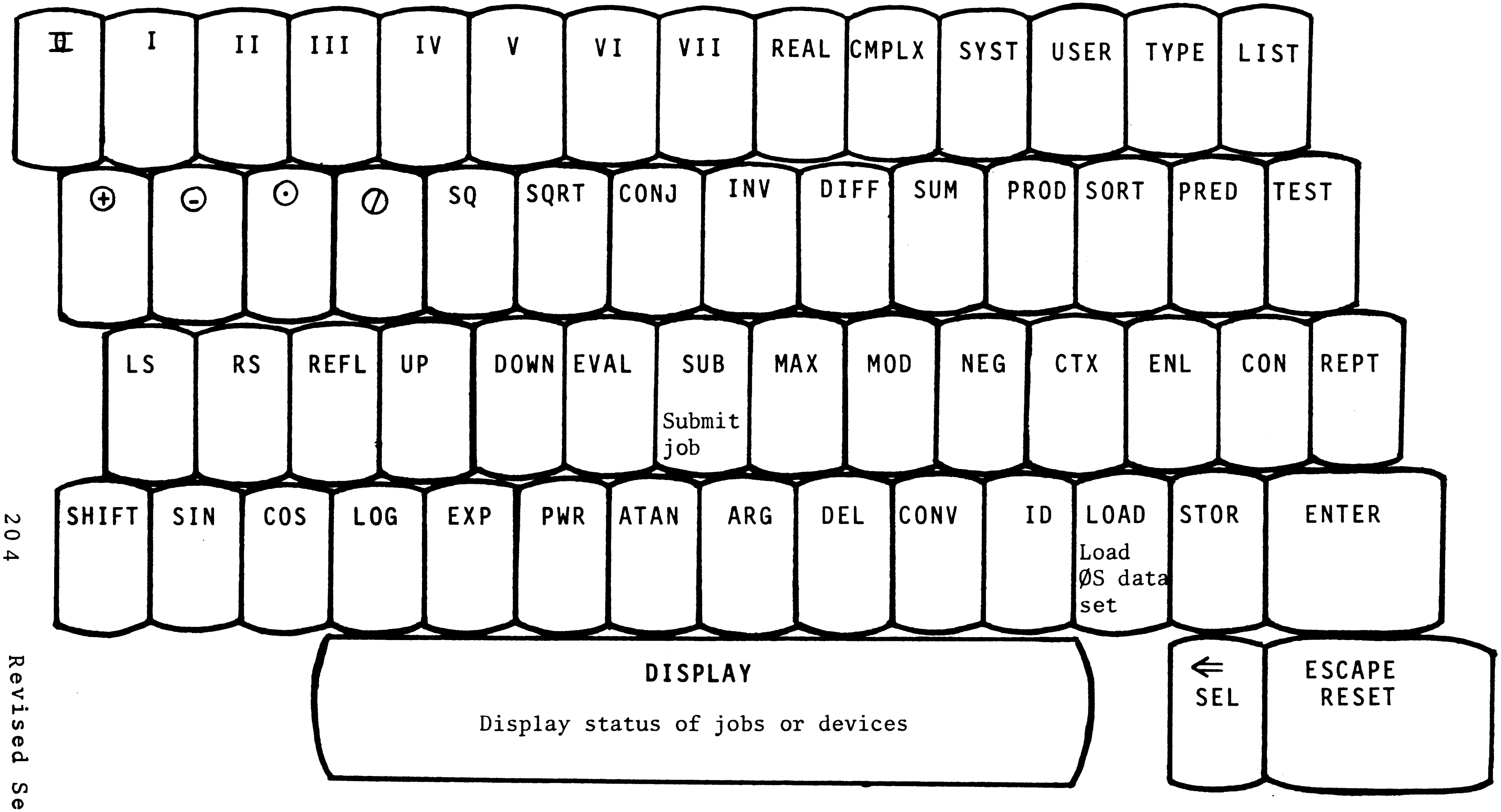
Revised Sept. 1, 1971

COL: LIII KEYBOARD



Revised Sept. 1, 1971

COL: LIV KEYBOARD



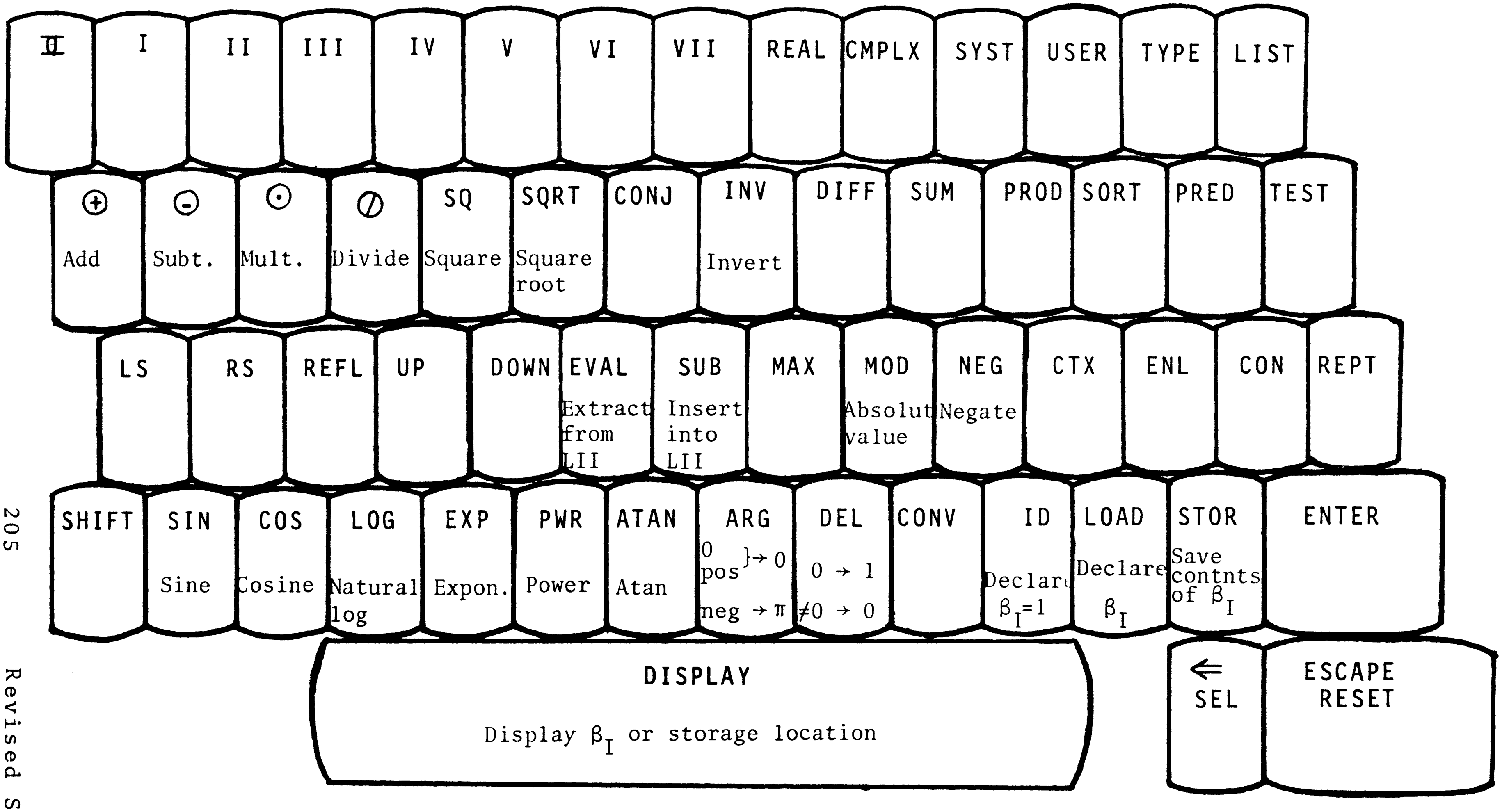
Submit job

Load ØS data set

Display status of jobs or devices

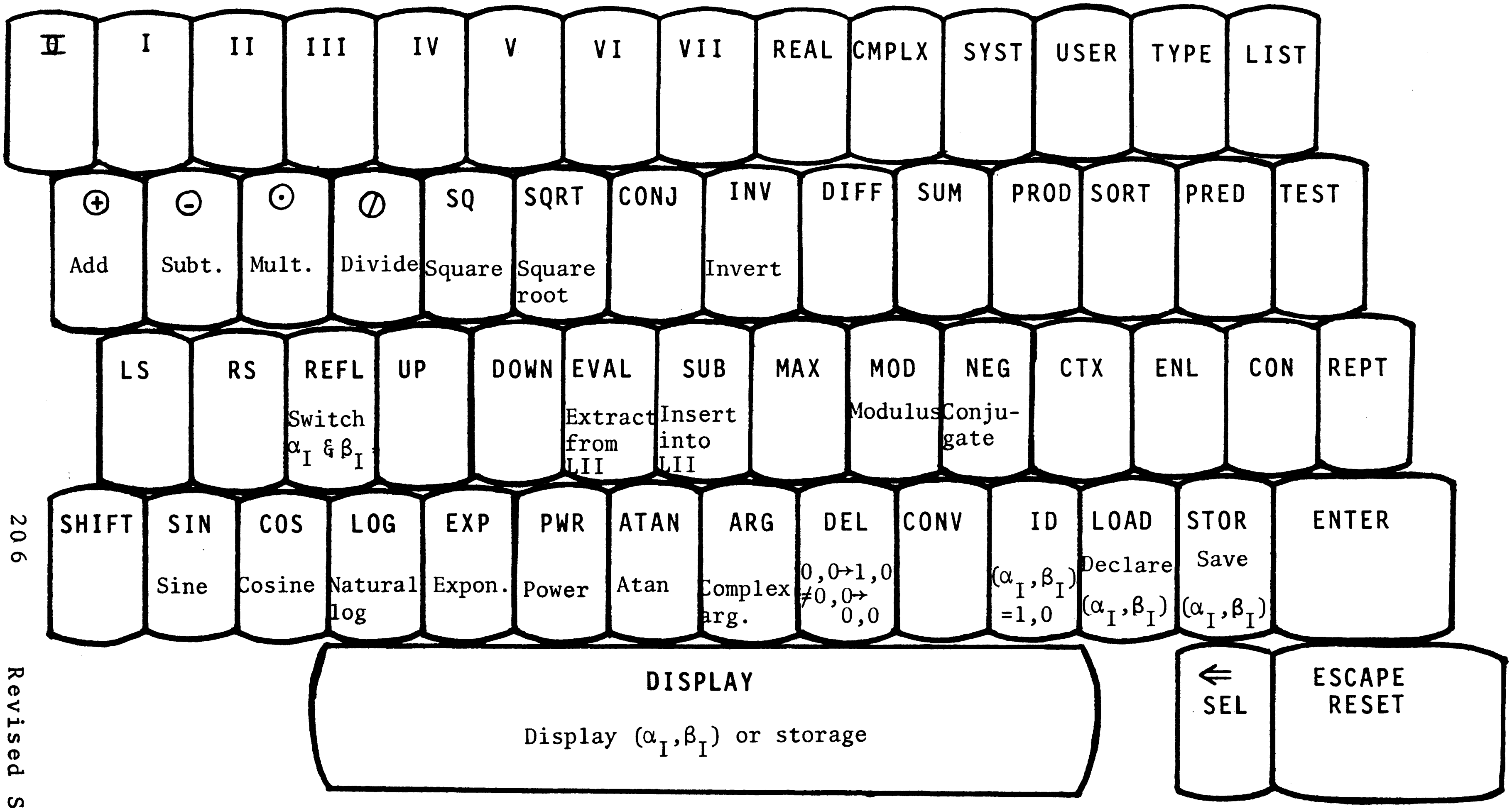
Revised Sept. 1, 1971

MOLSF: L1 REAL KEYBOARD



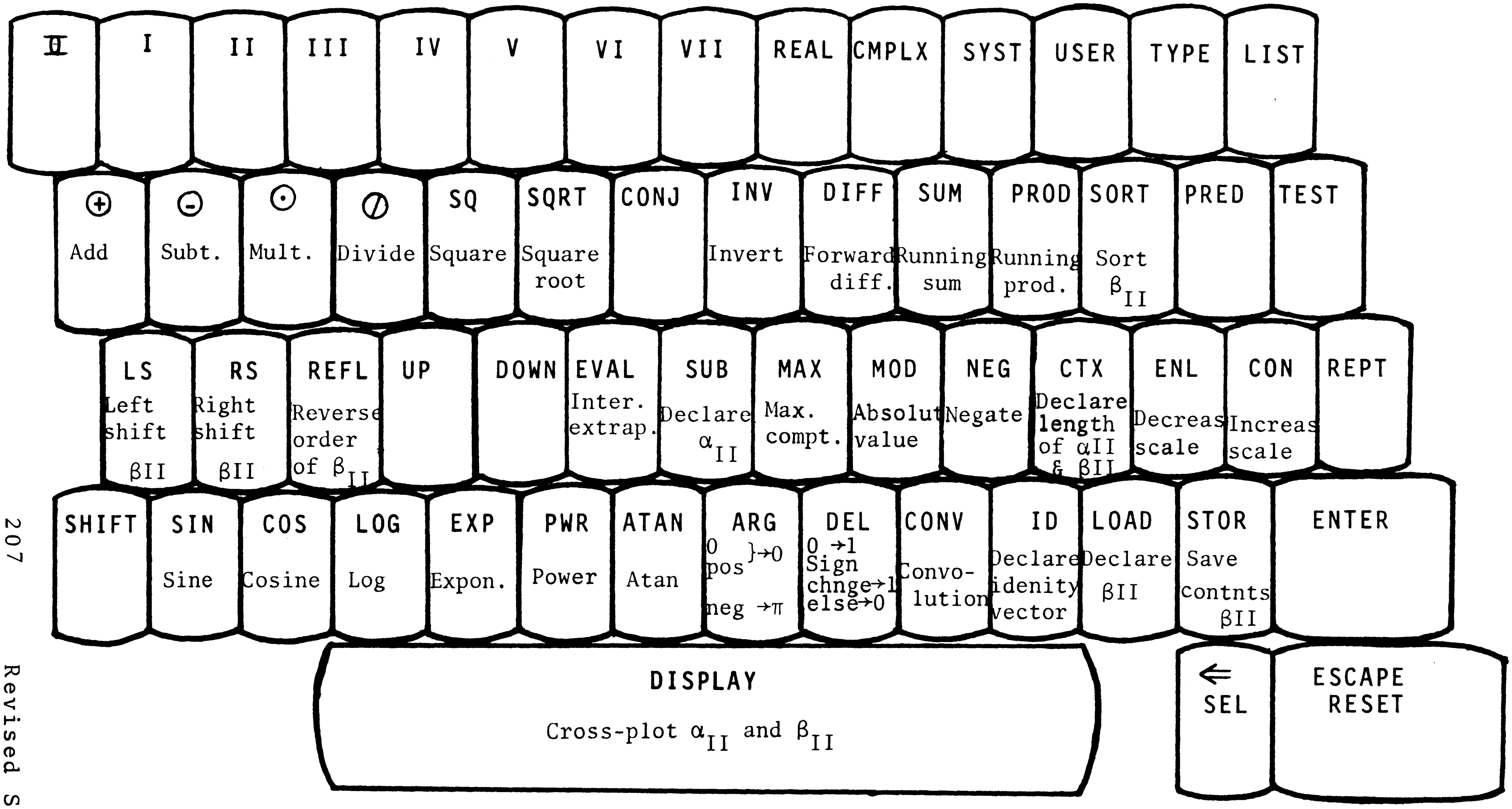
Revised Sept. 1, 1971

MOLSF: LI COMPLEX KEYBOARD



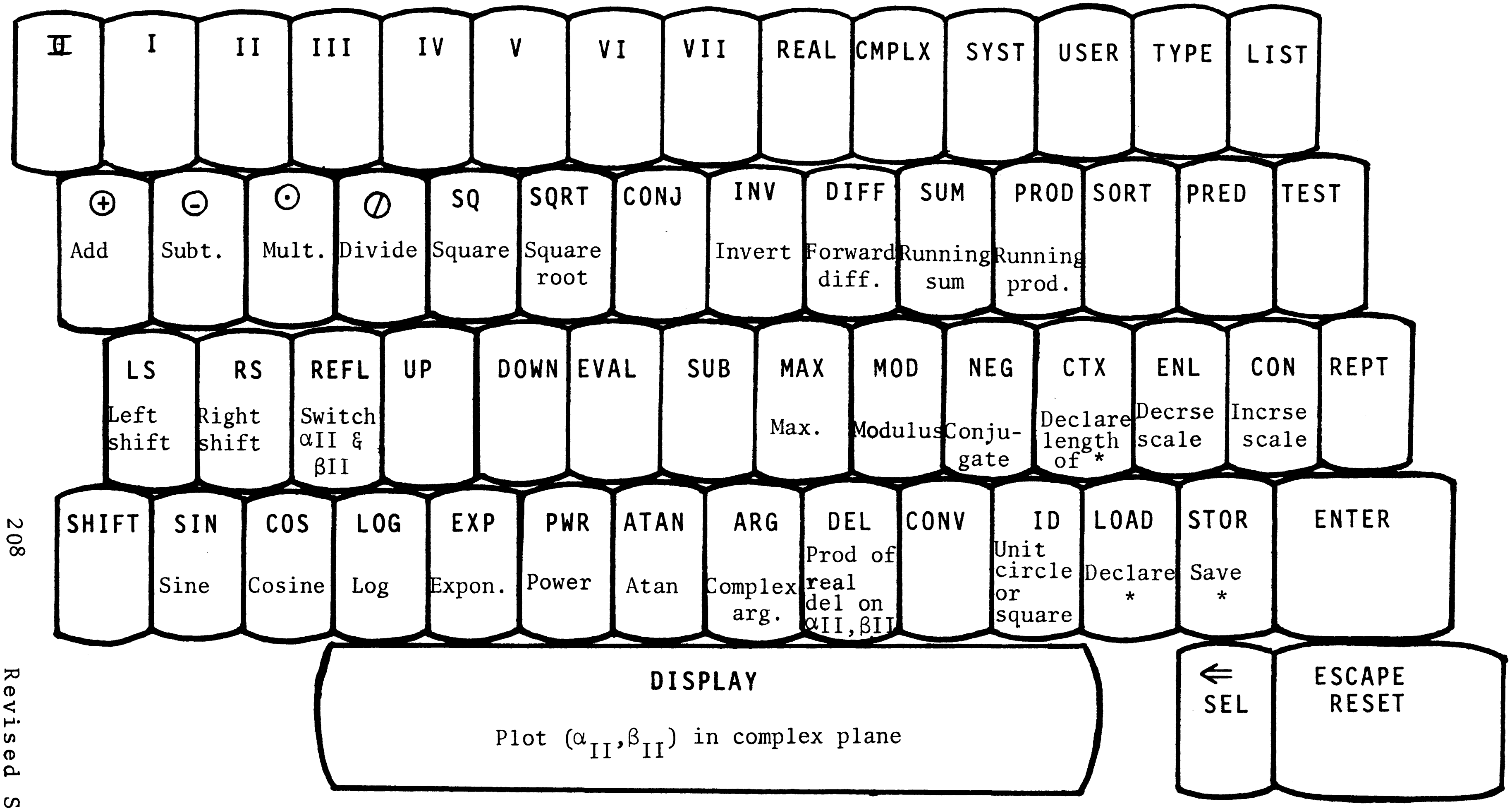
Revised Sept. 1, 1971

MOLSF: LII REAL KEYBOARD



Revised Sept. 1, 1971

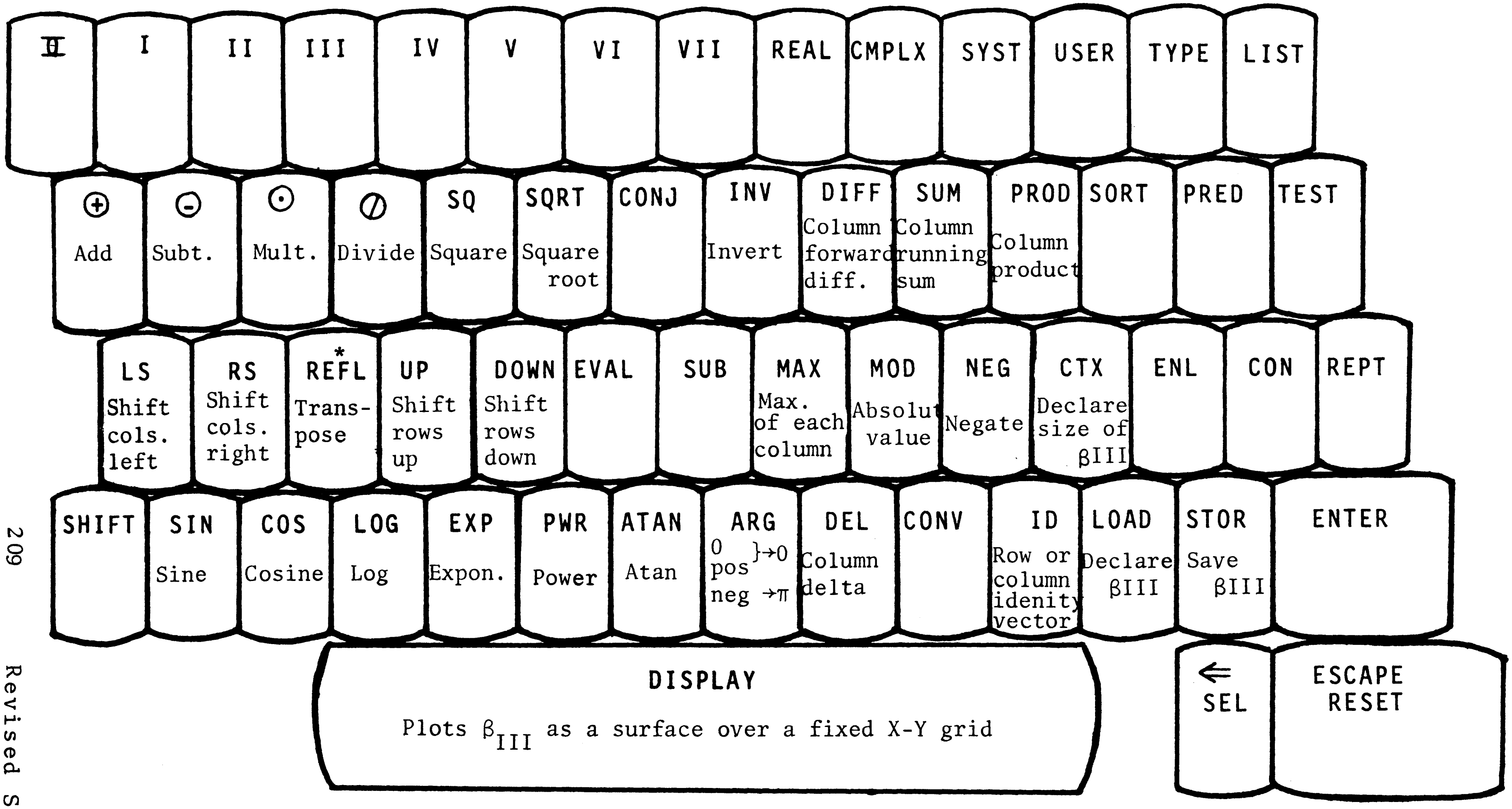
MOLSF: LII COMPLEX KEYBOARD



* (α_{II},β_{II})

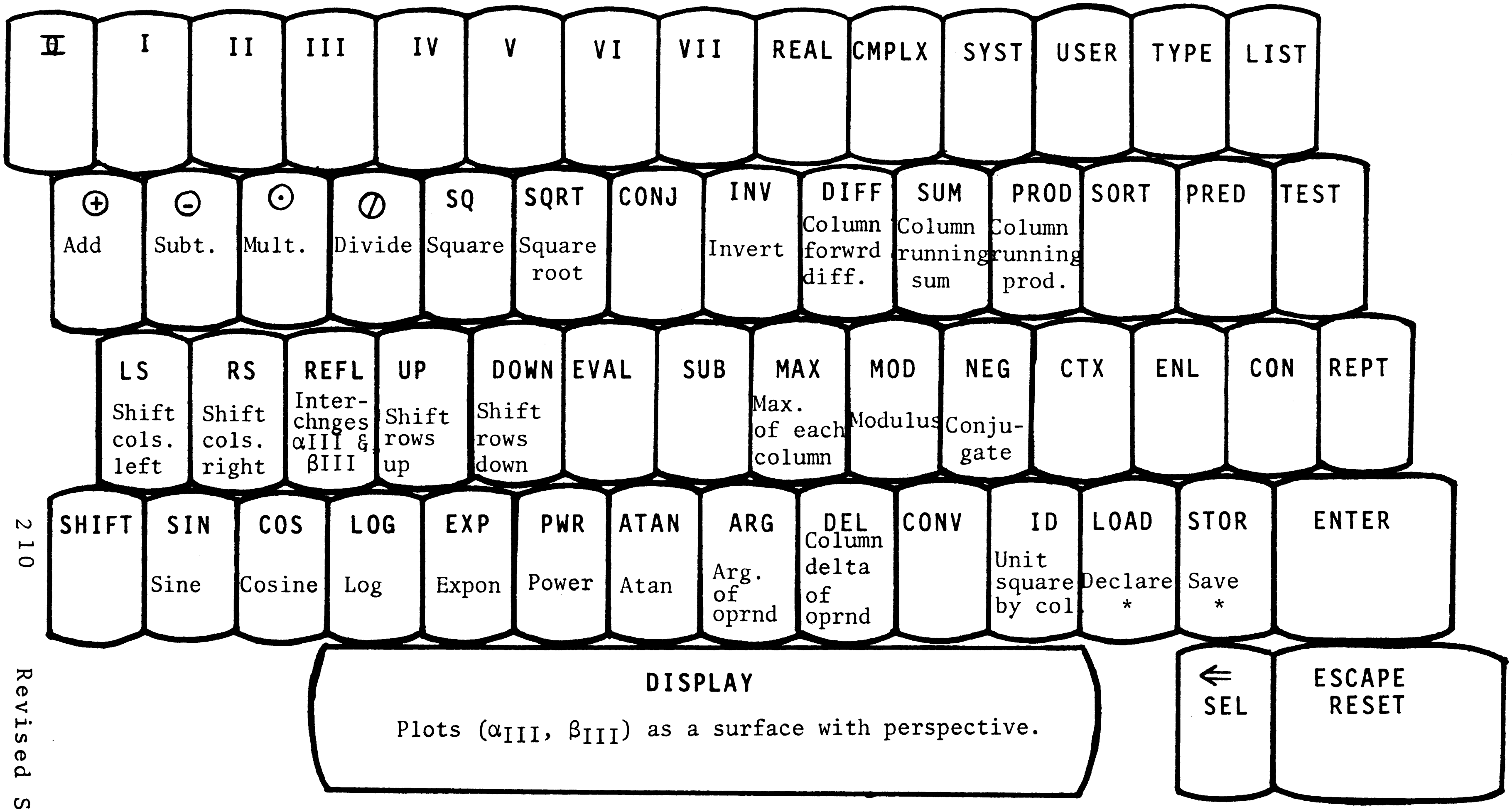
Revised Sept. 1, 1971

MOLSF: LIII REAL KEYBOARD



Revised Sept. 1, 1971

MOLSF: LIII COMPLEX KEYBOARD



0	I	II	III	IV	V	VI	VII	REAL	CMPLX	SYST	USER	TYPE	LIST

⊕	⊖	⊙	⊘	SQ	SQRT	CONJ	INV	DIFF	SUM	PROD	SORT	PRED	TEST
Add	Subt.	Mult.	Divide	Square	Square root		Invert	Column forwrd diff.	Column running sum	Column running prod.			

LS	RS	REFL	UP	DOWN	EVAL	SUB	MAX	MOD	NEG	CTX	ENL	CON	REPT
Shift cols. left	Shift cols. right	Inter-chnges αIII & βIII	Shift rows up	Shift rows down			Max. of each column	Modulus	Conju-gate				

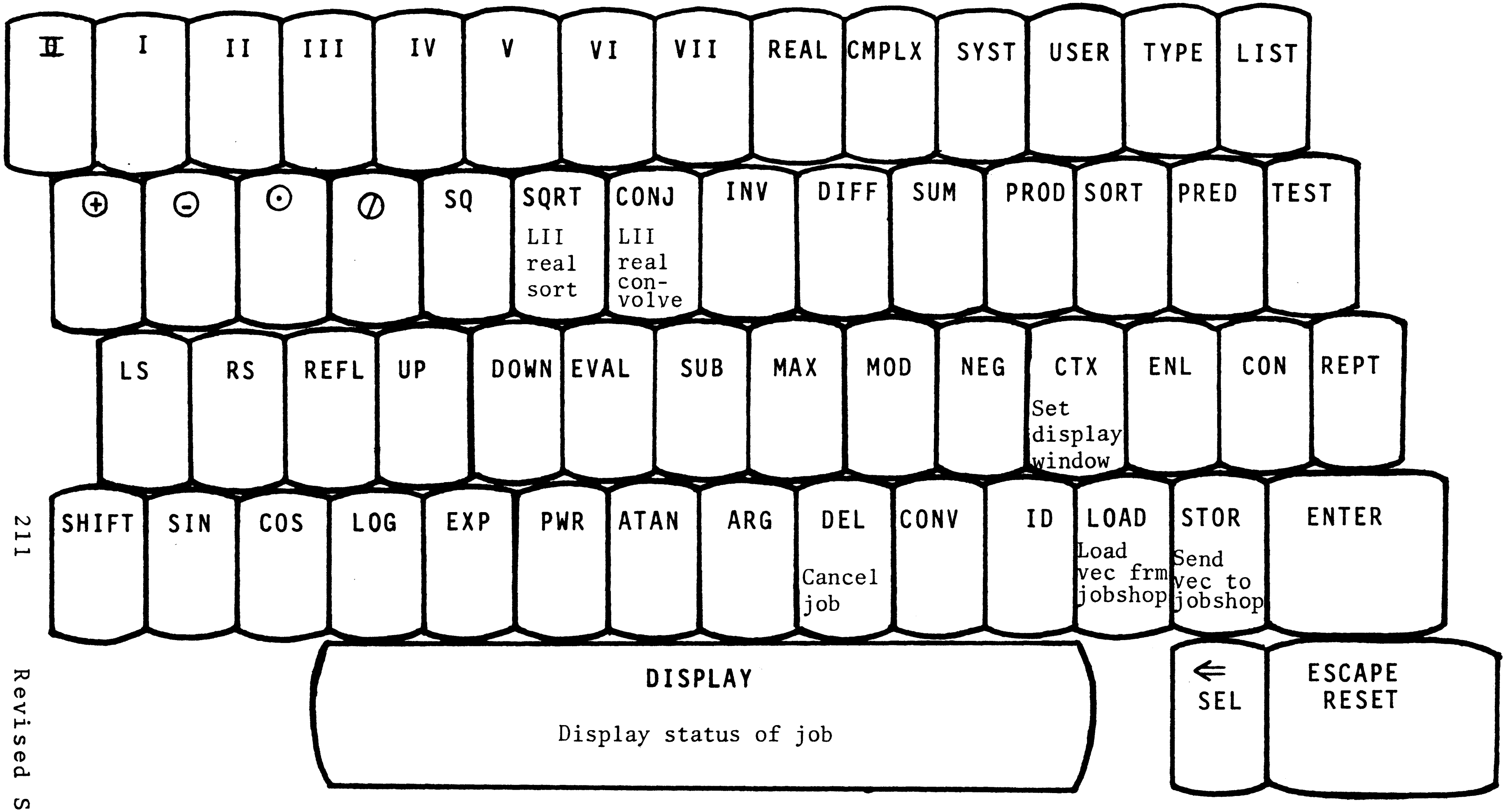
SHIFT	SIN	COS	LOG	EXP	PWR	ATAN	ARG	DEL	CONV	ID	LOAD	STOR	ENTER
	Sine	Cosine	Log	Expon	Power	Atan	Arg. of oprnd	Column delta of oprnd		Unit square by col	Declare *	Save *	

DISPLAY	⇐ SEL	ESCAPE RESET
Plots (α_{III}, β_{III}) as a surface with perspective.		

* (α_{III},β_{III})

Revised Sept. 1, 1971

MOLSF: LV REAL KEYBOARD



Revised Sept. 1, 1971

Appendix D

ON-LINE ERROR AND SYSTEM MESSAGES

The On-Line System displays various system and error messages. The following list explains some of the more common messages. The format for error messages is:

THE ERROR MESSAGE

Key or keys which usually cause the message to be displayed.

An explaination of the message.

Suggested user response.

For system messages the format is:

THE SYSTEM MESSAGE See page ...

AUTOSAVE CODE = number See page 15

CONTEXT ERROR

CTX level 0 operand RETURN (on MOLSF)

You have requested a vector or array dimension (s) that is too large, zero, or negative.

Request a context within the allowed range. If the context is less than 873, then your user number may have a lower context limit and attempts to exceed that limit will result in an error message.

ENTER USER NUMBER See page 15

EXPONENT OVERFLOW

Any sequence of keys on the mathematical levels (on MOLSF).

An operation has caused the exponent of a number to exceed the hardware limitations of the computer.

Wood, R. C. and J. C. Bruch, Jr.: "Teaching Complex Variables with an Interactive Computer System." Article Submitted for review and publication in the *IEEE Transactions on Education*, July, 1970.

Wood, R. C. and J. A. Howard: "An Interactive Computer Classroom." Educational Research and Methods Journal, Vol. 2, No. 4 (June, 1970), 29-31.

Yu, S. Y.: "On-Line Computer Program for Magnetic Hysteresis Loop Characteristics of Permeable Rods." *TRW IOC 3343.3-165*, November 7, 1966.

Yu, S. Y.: "On-Line Computer Program for Solving up to Five Simultaneous Equations." *TRW IOC 66-3343.3-197*, December 6, 1966.

INDEX

INDEX CONTINUED

INDEX CONTINUED

INDEX CONTINUED

INDEX CONTINUED

INDEX CONTINUED

INDEX CONTINUED

INDEX CONTINUED

INDEX CONTINUED

INDEX CONTINUED