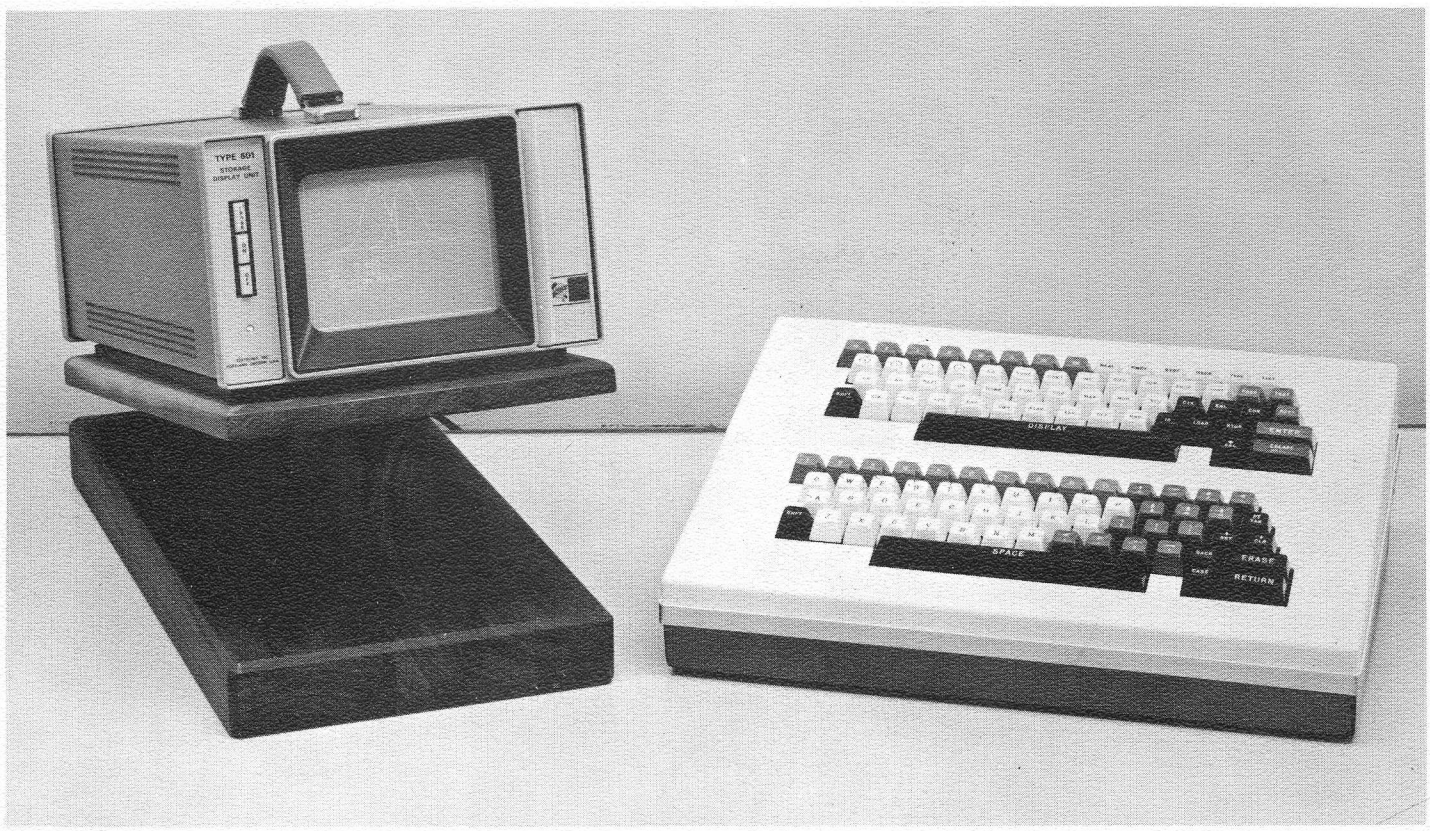
UCSB



on-line system manual

U.C.S.B. ONLINE SYSTEM MANUAL

This manual explains the U.C.S.B. Online System as it exists January 1, 1971. Please report any errors to the online consultant, so that corrections may be included in the updates to this manual.

U.C.S.B. ONLINE SYSTEM MANUAL

TABLE OF CONTENTS

TABLE OF CONTENTS CONTINUED

TABLE OF CONTENTS CONTINUED

TABLE OF CONTENTS CONTINUED

TABLE OF CONTENTS CONTINUED

INTRODUCTION

The UCSB on-line system (OLS) provides the capability for sophisticated mathematical analysis or file and string manipulation for use in solving problems where human interaction is either necessary or desired.

The primary aspect of modern computer systems is that of direct user control of computational processes. An on-line system (OLS) provides interactive facilities by which a user can exert deterministic influence over a series of computations. A time-sharing system provides a means by which partial computations on several different problems may be interleaved in time and may share facilities according to predetermined sharing algorithms. Placing a single user in direct control (i.e., on-line) of a large scale digital computer is impractical from an economic viewpoint, while a small-scale, less expensive computer generally does not provide the computational power required for significant scientific applications. Consequently, on-line computing has come to depend upon time-sharing as its justifiable mode of implementation. The UCSB OLS described in this manual is a time-sharing on-line system. The number of users that can be supported at any one time is limited only by the hardware capabilities of the given computer. At the present time a limit of sixty-four users has been set by the Computer Center.

The fact that a user is in direct control of a computational process, giving commands from his own console, is only superficially

analogous to his being his own operator. The important distinction lies in the program power for direct control associated with his console. This interactive capability of OLS is provided by the unique operator (controlling process) - operand (objects affected) software structure of the system.

It is the objective of this manual to provide the user with the fundamentals of the OLS languages and to indicate how these basics can be used.

The programming capabilities of the software underlying the OLS are quite extensive. It is important for the user to recognize that he doesn't have to completely understand all the system capabilities in order to solve a particular problem. A good approach is to pay detailed attention to only that part of the system which is required for the solution of the problem at hand.

The user of this system must basically provide the mode of problem solution himself by creating and applying his own user's language to the problem under consideration. If difficulty is encountered, the user should insure that he has properly interpreted the operators and operands which constitute the user's language he has constructed. Sample problems are provided in the appendix to aid the user in the use of his console. These problems may also assist in clarifying the correct utilization of a given OLS instruction. If the above approach fails to remedy the trouble, call upon some other user for assistance.

Any attempt to delineate the limitations of the system's applicability would be unfair, since such limits are largely determined by one's ingenuity in using the system. Mathematical simulation, on-line control of experimental systems, string and file manipulation, and data analysis are but four examples of areas outside that of classical mathematical analysis in which the system has been applied. In many instances, a problem which appears to be completely inappropriate for OLS can be resolved by employing some facet of the OLS structure in a slightly different fashion.

A simplified block diagram of the interconnections between an OLS user's console and the IBM 360/75 digital computer is depicted in Figure 1.1. The console is comprised of the keyboard, which acts as the input device to the computer, and an output device. One such output device is the display scope.

During system operation, each time the console operator depresses a keyboard button a binary number (i.e. a series of zeroes and ones) uniquely corresponding to that button is transmitted to the 360 and stored in its memory. The program within the computer analyzes this number, then responds in accordance with conditions already set up by the antecedents of this key.

Transmission back to the user of the results of his key-pushes is provided by the output device(s) selected by the user. The output device can present either alphanumeric display (numbers and alphabetical characters) or a curvilinear display (graphical), as controlled by the user.

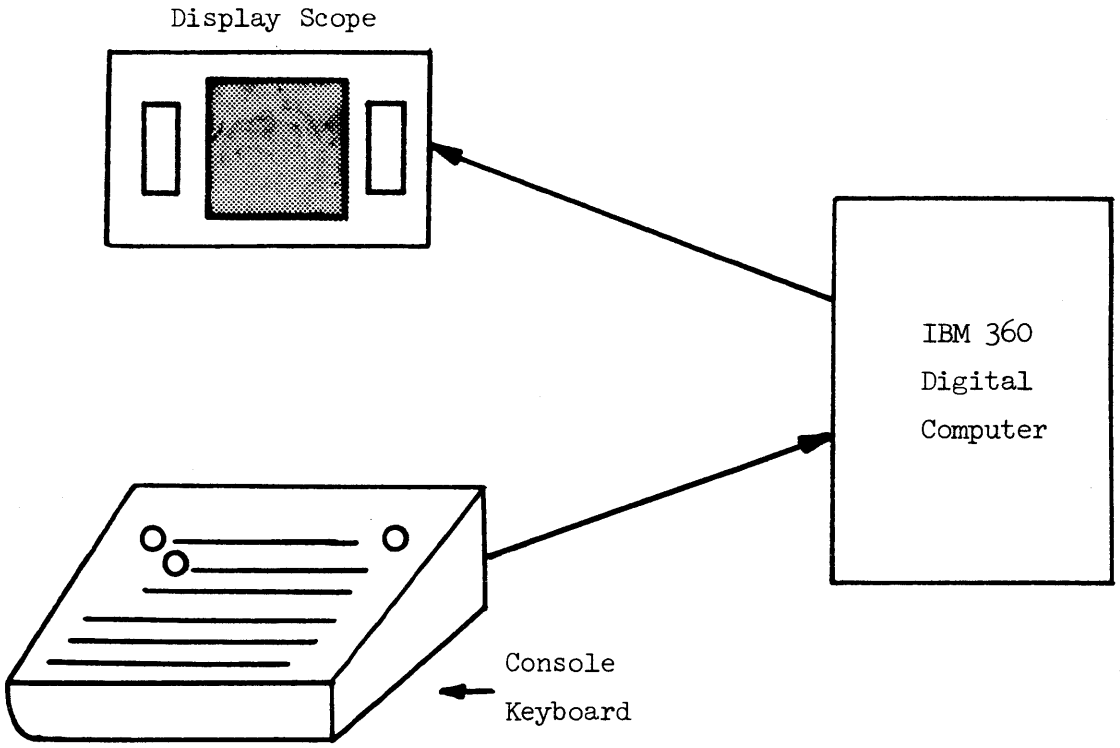


Fig. 1.1 Simplified block diagram of interconnections between a user console and the IBM 360 Digital Computer.

The underlying notion of a time-sharing system is that several users are on the air simultaneously. The computer responds to only one user at any given instant, but all active stations are constantly being monitored and serviced in turn. This can lead, during times of heavy system usage, to a delayed response by the computer to a given OLS station. During this delay period every button you push will be recorded by the computer and, when control is returned to you, all of these instructions will be performed.

Delayed response time is not only a function of the number of users, but it is also determined by the length and complexity of the task given to the computer by the user.

Control of the computer by the user is provided by the console keyboard (shown in Figure 1.2). One should first observe that the keyboard is divided into an upper and lower half. The halves are designated as the operator and operand portions, respectively.

The operator keys are grouped according to their system properties and typical uses. The operand keys are also grouped according to their usage.

In general, the green, black, and red keys provide program and console control. On the upper keyboard, they furnish program controls such as a command to stop a program, to store data, and to repeat an operation. On the lower keyboard, they provide a miscellany of punctuation marks, the plus and minus signs for positive or negative numbers, and typewriter controls.

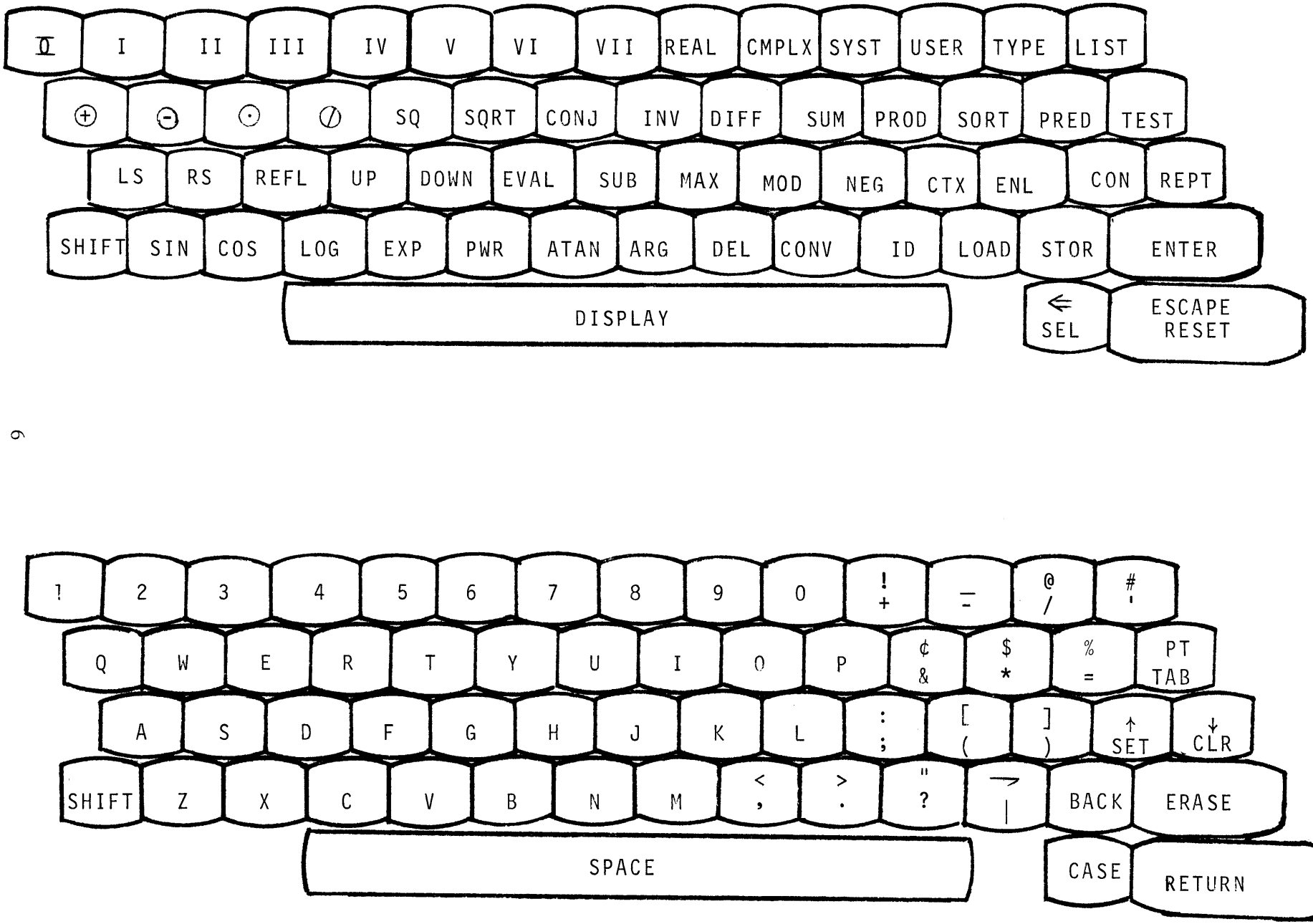

0 I II III IV V VI VII REAL CMPLX SYST USER TYPE LIST
⊕ ⊖ ⊙ ⊘ SQ SQRT CONJ INV DIFF SUM PROD SORT PRED TEST
LS RS REFL UP DOWN EVAL SUB MAX MOD NEG CTX ENL CON REPT
SHIFT SIN COS LOG EXP PWR ATAN ARG DEL CONV ID LOAD STOR ENTER
DISPLAY
SEL
ESCAPE RESET
! 2 3 4 5 6 7 8 9 0 + - / '
Q W E R T Y U I O P ¢ & $ * % = PT TAB
A S D F G H J K L : ; [( ]) ↑ SET ↓ CLR
SHIFT Z X C V B N M < , > . " ? | BACK ERASE
SPACE
CASE RETURN

Figure 1.2

The blue and yellow keys on the upper keyboard permit access to the various operator levels. The operator levels of OLS are described in later chapters. On the lower keyboard, the blue keys provide a means for typing numbers and entering numerical data.

The white keys labeled A through Z on the lower portion of the keyboard are used for typing alphabetic and Greek characters or as storage locations for data: single numbers, vectors (i.e., lists of scalars), arrays, or character strings. Whenever a quantity is stored in a particular location on a particular level, it replaces the previous contents of that location on that level. Thus, if the user stores a number under the B key and wants to retrieve it at a later time, he must not store anything else under B on that level in the interim. Once a quantity is stored in a given location, it will be available for use indefinitely unless it is replaced by another piece of data.

The white keys on the upper half keyboard are used for mathematical or string operations, depending upon the language used, or for storage of USER (created) programs. The upper keyboard storage buttons are like those on the lower keyboard in that storage cell information is replaced when a new quantity is stored.

The 16 green slashes or scribe marks on the alphabetic white keys are used for the construction of symbols not appearing on the keyboard. Symbols are constructed by guiding a dot around the scope so that it traces out the desired pattern. The

slashes indicate the 16 possible directions in which the dot can be moved under control of a string of vector instructions. Symbol generation is discussed in Chapter 2.

The output devices permit visual monitoring of the operations carried out by the computer. One such device, the display scope, consists of a storage tube and the appropriate circuitry to create high-quality alphameric and graphical displays.

The user is supplied two languages to apply to his particular problem, MOLSF and COL. Underlying these, however, there is a Basic language common to both.

The Basic System is composed of several levels of operators:

1) System level - SYST

   This level provides access to the on-line system, the ability to load and store files and subfiles, and billing information.

2) Type level - TYPE

   This level provides the ability to generate messages on the output device(s) and create symbols not already available on the keyboard.

3) Level 0 - L0

   Level 0 provides integer mathematical operations.

4) List mode and Edit level - LIST

   LIST mode enables a user to create programs. The EDIT level enables a user to modify USER programs.

MOLSF (mathematically-oriented language, single-precision floating point) provides a degree of numerical accuracy suitable for many scientific computations. Numbers represented in this system have up to seven decimal digits accuracy (single precision), with magnitudes ranging from 10^{-75} to 10^{+75}. The keys on the

upper keyboard, when pressed, will initiate operations. The lower keyboard provides access to data and storage areas.

MOLSF provides the ability to work with real and complex scalars, vectors, or arrays. The display can be either numerical or graphic.

COL (card-oriented language) is a non-mathematical language for the creation and manipulation of character strings and files. A file is defined as an ordered set of records or strings; a record or string, in turn, is defined as an ordered string of characters (alphameric and special).

Through COL, the user can:

1. access data-sets residing on the installation's storage devices.
2. submit programs coded in any language supported by UCSB (Fortran, PL-1, etc.) for batch processing, and access output generated by them.
3. transfer vectors between an OLS console and a program submitted for batch processing.(in conjunction with MOLSF).
4. obtain punched or printed copy of files.
5. manipulate character strings.

Conceptually, COL and MOLSF differ only in the definitions they assign to the upper keyboard operators.

To produce the desired machine response the user's problem must be logically and unambiguously defined. Error messages are generated by the system in response to illogical sequences. The control emphasis is informal and casual rather than programmatic. Consequently, the system makes computational meaning from a great variety of key sequences. By recognizing initiation

operators as operators that override partially completed sequences, it allows the user to start a new sequence immediately without producing system error messages.

As mentioned earlier the Basic system provides the ability to create USER programs. This facility is called LIST mode as opposed to Manual mode.

In Manual mode, the computer reacts in direct computational response to each user request (keypush). The display of its efforts may be requested by the user at any time; the machine does not automatically display its work.

LIST mode allows the construction and subsequent storage of a list of keypushes, thus providing a procedure for constructing new programs from those operators already defined in the system. When construction of a USER program is desired, the user simply enters LIST mode and then pushes any sequence of keys just as though he were operating the on-line system in Manual mode. However, instead of responding to these key pushes in the usual way (i.e., by initiating the corresponding subroutines), the computer records and displays the list of the keys which have been pressed. After leaving LIST mode it is possible to alter the list of keys and then to store them under an upper keyboard white key, for execution at a later time.

These USER programs can be executed at any time. It is also possible for one USER program to call another creating a pyramiding feature, which makes it possible to construct programs of virtually unlimited complexity.

Before and after storing a USER program it can be displayed, edited, and restored.

Research and developmental work on OLS will continue for some time to come. This work is usually done on a Developmental System, as opposed to the Day System on which users generally operate.

The Developmental System is identified by the message:

"DEVELOPMENTAL SYSTEM"

before the log-in procedure. Depending on the nature of the development in progress, the operator may allow users to sign on and use the Developmental OLS; however, user libraries are not accessible and the system may be cancelled or reloaded at any time. To offset these liabilities, there is no charge for using the Developmental System.

When the above message does not appear, the User or Day System is up. Libraries are accessible and the Computer Center is endeavoring to maintain the maximum system stability possible. When the User system is up, user activity is billed.

Regular schedules for the User and Developmental Systems are posted at the beginning of each quarter.

Users with suggestions for additions and improvements or who have discovered "bugs" should contact the Computer Center.

When new facilities are available they will be described in News and Notes, published bi-monthly by the Computer Center. When the manual is updated, a notice will appear in News and

Notes. It is the user's responsibility to contact the Computer Center to acquire the latest updates. Updates will be dated, thus a user will be able to secure all past updates at any time.

THE BASIC SYSTEM

The Basic system provides: access to the on-line system routines, a common interface between the languages, services to the language processors, and library upkeep. The Basic system has several special operators and five levels. The special operators are: REPT, SELECT, TEST, LIST, ENTER, and PRED. The levels are TYPE, L0, USER, SYST, and EDIT. The facilities of the Basic system are always available when one is signed on the air. Thus the facilities of the Basic system augment whatever language one is working with.

The Basic system has been designed so that languages may be added or deleted with one small change to the Basic system. To the user this means that the operator definitions for the Basic system do not change when one changes languages. NOTE: L0 SUB and EVAL are not part of the Basic system, but are part of the individual language processors.

The TYPE level enables the user to type messages on his output device. Special characters and frequently used sequences of characters may be generated, stored, and recalled for use in typing messages.

The SYST level enables the user to alter his library; display billing charges; load or change language processors; load, store, display or delete files; and terminate access to the system.

L0 provides integer arithmetic and indexing facilities. Simple operators (⊕ , ⊖ , ⊙ , ⊘ , SQ, and DEL) on L0 give

integer results in the quotient and remainder registers. The contents of the quotient register may be saved under any of the letters A through Z and alpha through omega. Operators that require integer operands on both the Basic system and the language processors, will accept a L0 operand. L0 operands may consist of an integer or a storage location name. Optionally, the L0 operand may specify that the contents of the storage location be automatically incremented or decremented after execution of the operator.

The LIST mode, entered by pressing LIST, allows one to define programs containing any key except LIST or RESET. Every key pressed while in that mode is displayed and added to the program definition until the LIST key is pressed the second time. The user is then placed automatically on the EDIT level, where he may modify and store his programs.

The USER levels (USER L0 through USER LVII) are for user defined operators. Programs created using LIST mode and the EDIT level may be associated with a level and an operator or a name. The CTX key may be used between the level and operator to add further to the number of user defined operators. The total number possible at one time is 8(number of levels) x 2(use of CTX) x 31(number of operators) or 496 programs.

The special operators add flexibility to the system. REPT allows the user to repetitively execute a series of keys with, optionally, a control variable. TEST allows for branching in user programs. The special user program operators PRED and ENTER control the data upon which a program will operate.

2.1 SYST

2.1.1 ACCESS TO OLS

The SYST key notifies OLS that a user wishes to sign-on. 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 to OLS which program library is to be used, how many of the OLS facilities this user may access, whether a problem name is required, and the funds remaining in his account. The procedure for obtaining a user number is outlined in Appendix A.

PROCEDURE FOR SYSTEM INITIATION

As an illustration of the procedure to activate the system, assume that a user has been assigned a user number, ID code, user name, and that a problem name is required for this user number. After turning on the equipment, if necessary, he presses the SYST key.

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

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 should OLS fail 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

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 the next section.

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 will cause 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.

2.1.3 WARMSTART

Although every effort is made to provide uninterrupted service to users during those hours of the day when service is promised, occasional interruptions occur. Irrecoverable system failures do occur; furthermore, since OLS and batch processing time-share a single computer, additional interruptions are sometimes required to keep the latter "on the air". Users will be notified, if possible, of scheduled interruptions before they occur, by the appearance on the user's display scope of a message indicating that an interruption is imminent. The user should then store any material he wishes to save. When service is restored (i.e. when SYST yields a response), the user must re-initiate the sign-on procedure.

To minimize the inconvenience of service interruptions, especially those which are unpredictable, a procedure called "warmstarting" is provided. Warmstarting is an option available at sign-on to the user after an interruption, and is initiated as follows:

KEYBOARD ENTRY	OLS QUERY/RESPONSE
SYST (user number) RETURN	ENTER USER NUMBER (user no.)
(ID number) RETURN	ID NUMBER =
(name) RETURN	USER NAME = (name)
	AUTOSAVE CODE = 6 (new autosave code)
	LOAD
RETURN	ENTER AUTOSAVE CODE
5 RETURN (old autosave code)	RESTART COMPLETE

Warmstarting recovers all user programs, special characters, and data in the condition they were when the user last pressed SYST or RESET prior to the interruption. The user must warmstart with the same user number and same autosave code, but not necessarily the same terminal, as he was using when the failure occurred. When restart is complete the user no longer has his old autosave code; he must use the new one given him during sign-on. In this example the user would no longer use autosave code 5 - the one he had when the system went down - but must now use autosave code 6, the one he was just assigned.

2.1.4 USER LIBRARY ORGANIZATION

Every user number has associated with it at least one user library. If a user number is further qualified by user name, there is a separate library for each user name; otherwise, all those who use the same user number share the same user library. A library contains files, each with one or more subfiles, which are composed of programs and special characters or a single data level. A file is associated with the language which the user was using when he stored his file. All data subfiles must be of the same language.

A subfile is either a data system or a user system. There may be only one user system per file because this contains all basic system data, i.e. user programs, and special characters. There may be a data subfile for L0 and one for every level for which data may be stored. A subfile may be defined for any of

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 LO 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 presently supported subfiles are shown below.

PRESENTLY SUPPORTED SUBFILES

The Basic System subfiles

USER	User programs and special characters. Both are contained in one subfile.
LO	LO storage locations.

COL subfiles

LI	LI 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.

The STORE and LOAD operations both accept predicate lists. Predicate lists tell the system what kind of subfile is to be created or loaded.

Predicate List Formats

Specification	Subfile to be loaded/stored
USER RETURN	All user subfile data (USER L0-LVII, CASE 3-9).
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 Cases 3 through Cases specified.
L0	L0 data.
lvl (REAL or CMPLX)	Data of level specified.
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.

NOTE: A not sign ("¬") after the USER when defining a section of a subfile tells OLS to create/load everything except the specification.

lvl = [L0,LI,...,LVII] and number = [3,4,...,9]

2.1.5 STORING A FILE IN YOUR USER LIBRARY

The procedure for storing a file in your user library is:

KEYBOARD ENTRY	OLS QUERY/RESPONSE
STORE (predicate list)	STORE
RETURN (name) RETURN	FILENAME= (name)
(protection code) RETURN	PROTECT CODE= (protection code)
	DONE

The first time a user stores a file, 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 supplies 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.

If a file with a user level-case level subfile was previously stored and a portion of a user subfile was stored now, the new subfile replaces the entire user subfile. 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
STORE USER RETURN	STORE
VENICE RETURN	FILENAME=VENICE
GONDOLA RETURN	PROTECT CODE=GONDOLA
	DONE

2.1.6 LOADING FROM ANOTHER USER NAME ON YOUR USER NUMBER

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 log in, 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 the whole subfile (LOAD USER RETURN) it replaces the current subfile, but does not purge parts of the current subfile not overstored by corresponding parts of the new subfile.

LOADING A SUBFILE

A subfile may be loaded from a file separately, by the following procedure:

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

LOADING FROM ANOTHER USER NUMBER

A file may be loaded from another user number if you know the ID code. The procedure is:

KEYBOARD ENTRY	OLS QUERY/RESPONSE
LOAD USER (user number) RETURN	LOAD (user number)
(ID no.) RETURN	ID NUMBER =
	LOAD

You may now load a file or subfile as explained in the above sections. The file loaded, however, will be from the user library specified. On user numbers which are subdivided into user names, OLS will request a user name after entering the ID code.

LOADING FROM ANOTHER USER NAME ON YOUR USER NUMBER

One may load a file from another user name by the following procedure:

KEYBOARD ENTRY	OLS QUERY/RESPONSE
LOAD USER (name) RETURN	LOAD
	USER NAME = (name)
	LOAD

You may now load a file or subfile from the user library specified.

2.1.7 DELETING A FILE OR SUBFILE FROM YOUR USER LIBRARY

If you wish to delete a file from your library, the procedure is as follows:

KEYBOARD ENTRY	OLS QUERY/RESPONSE
DEL (name) RETURN	DELETE (name)
(protect code) RETURN	PROTECT CODE = (protect code)
	FILE PURGED

To delete just one subfile from a file in your library specify a predicate list as in STORE:

KEYBOARD ENTRY	OLS QUERY/RESPONSE
DEL (predicate list)	DELETE
(filename) RETURN	FILENAME= (filename)
(protect code) RETURN	PROTECT CODE = (protect code)
	FILE PURGED

NOTE; OLS will not allow one to delete part of a subfile, i.e. USER LI only.

2.1.8 DISPLAYING YOUR USER LIBRARY

The library which you are using may be displayed by pressing the display button. One file is displayed thereafter, as long as there are files left, every time RETURN is pressed. Each file is displayed 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. The subfiles displayed (if there are any) appear below the first subfile. No message is displayed to indicate all subfile or files are displayed. The first RETURN causes a new file to be displayed, the commas all of the subfiles. The example below shows a typical user library display:

```
TEST              MOLSF LII C
                        USER
INTEGRAL          MOLSF USER
                        L0
                        LII R
PUNCH             COL   LIII C
PUNCH2            COL   USER
                        LIII R
```

If your user number is subdivided into user names the sequence "DISPLAY USER name RETURN" will display the library for the user number that you are on, and name that you specified. Your library is displayed without specifying a user name.

2.2 THE TYPE LEVEL

On the TYPE level the lower keyboard keys function like those on a normal typewriter. The keys RETURN, BACK, and SPACE provide "carriage" control just as on a regular typewriter. The upper keyboard keys ENL and CON "roll" the carriage up one line and down one line respectively. The RS key positions the display to the upper lefthand corner of the display device. On the TYPE level, the CASE key operates like its typewriter counterpart, moving the keys to a different set of characters. In the normal case, which is CASE 1, the lower keyboard buttons type the symbols appearing on their faces. By pressing either CASE 2 (okb. SHIFT 2) or holding down the SHFT key (nkb. only) the alphabetic keys type the Greek alphabet and the numeric keys type superscripts. The punctuation keys type other symbols, such as "=", as shown in Table 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 computer remains at that level until a new level is defined or the TYPE or DISPLAY key 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 will be

Table 2.1 Symbols Available on the Lower Keyboard in the TYPE Mode

CASE 1	CASE 2	NAME (CASE 2)
A	α	alpha
B	β	beta
C	χ	chi
D	δ	delta
E	ε	epsilon
F	Π	capital pi
G	γ	gamma
H	θ	theta
I	ι	iota
J	Σ	capital sigma
K	κ	kappa
L	λ	lambda
M	μ	mu
N	η	eta
O	o	omicron
P	π	pi
Q	ϕ	phi
R	ρ	rho
S	σ	sigma
T	τ	tau
U	υ	upsilon
V	ν	nu
W	ω	omega
X	ξ	xi
Y	ψ	psi
Z	ζ	zeta

CASE 1	CASE 2	NAME (CASE 2)
(	[	square bracket
)	]	square bracket
,	_	underline
.	.	uppercase dot
?	'	apostrophe
0	0	superscript 0
1	1	superscript 1
2	2	superscript 2
3	3	superscript 3
4	4	superscript 4
5	5	superscript 5
6	6	superscript 6
7	7	superscript 7
8	8	superscript 8
9	9	superscript 9
+	=	equality sign
-	/	division

displayed on the output device. Thus the key sequence might be this (new keyboards):

TYPE RETURN A+B = LO LOAD

A + B DISPLAY SPACE RETURN

The first RETURN after the TYPE moves the "carriage" all the way to the right so that "A+B=" is left-adjusted. These key-pushes with A=2 and B=-3 will cause the computer to print on the screen:

```
A+B=-1
```

When the right-hand margin of the output device is reached while the computer is on the TYPE level, a carriage return occurs automatically. The next symbol appears at the left-hand margin of the next line. If the user leaves the TYPE level and returns to it later, typing will begin at the place where the last typing ended, unless the user positions it otherwise.

The ⊕ and ⊖ operators may be used to enable and disable, respectively, all display (including curvilinear displays). RESET always enables display. These operators are particularly useful when one must do SYST level operations in user programs without display.

Words that appear on operator keys should be spelled out on the lower keyboard when typing messages referring to them.

SUMMARY OF TYPE LEVEL OPERATORS

⊕	enables display.
⊖	disables display.
RS	the current display position is set to the upper-left-hand corner of the screen.
ENL	the current display position is raised by one line.
CON	the current display position is lowered by one line.
DEL	a delay of approximately one-half second is made. This operator is often used in programs after an erase so that the characters do not fade.
CASE	(okb SHIFT) followed by a L0 operand changes the case to the L0 operand. CASE 1 is the regular character set. CASE 2 is the Greek alphabet. Case 3 through 9 are user defined special characters and messages.
RETURN*	positions the next character at the extreme left of the next line.
BACK*	moves the display position one position to the left.
DISPLAY	places the user on CASE 1.

2.2.1 MESSAGE AND SYMBOL GENERATION

The message and symbol generation level is reached by the sequence "TYPE MOD". Once on that level the user may select either message or symbol generation. To select symbol generation

*Theses keys are in reality operands, however on the TYPE level they act as operators.

the user should specify initial coordinates as explained in the section on symbol generation. If a valid operand is not received the user is placed on message mode.

MESSAGE GENERATION

When creating large user systems, messages begin to take large portions of USER programs. Message generation allows the user to create and store messages that are used frequently. After entering message generation mode the user may type letters, numbers, punctuation, and the keys SPACE, BACK, DISPLAY and RETURN. Finished messages may be stored with the sequence MOD STORE CASE followed by a number from three through nine representing the CASE level and a letter (Roman or Greek) where the message will be stored.

SUMMARY OF OPERATORS FOR MESSAGE GENERATION

DISPLAY RETURN	displays message generated thus far.
BACK	removes last character specified.

SYMBOL GENERATION

When typing messages, the user may often discover he needs symbols which do not appear on the keyboard (e.g., ∂, ∇, $\int$, etc.). To meet this requirement, the on-line system provides the user with SHIFT levels 3 through 9 in the TYPE mode for storage of manually generated symbols and messages. The user constructs these symbols by guiding a dot on the display scope to trace the desired pattern.

For purposes of character generation, the display scope is partitioned into a 4096x4096 grid. The distance between any two vertical or horizontal lines is called a unit. Standard-sized keyboard letters are each centered on a grid 160 units wide by 224 units high. Thus, a page of typing held by the display scope consists of 18 lines of 25 spaces each as depicted in Figure 2.2. The current display position is defined as the lower right-hand corner of the last character typed and is referred to as the origin (0,0).

Suppose, for example, the user presses "F" in TYPE mode and the screen illuminates "F" in line 2, space 2 (Figure 2.3). He now changes levels by pressing LII CMPLX, performs some computations and graphical display, but does not print any numbers or letters, and then returns to the TYPE level with TYPE. Since nothing has been typed after "F", even though many computations, level changes, and graphical outputs may have been executed, the carriage position is still the lower right-hand corner of space 2 in line 2 and is indicated in Figure 2.3 by the large dot with (0,0) beside it. The dot does not physically appear on the screen unless the user executes instructions which explicitly call for it, as explained in the following paragraphs.

Assume the user is now in TYPE mode and he wants to generate the symbol "∇" and store it in CASE 5 D. He first presses MOD to get into the message/character generating mode and then starts symbol generation by positioning the dot at some initial point on the grid with the instruction:

$\cdot\ k_1,\ k_2$ RETURN

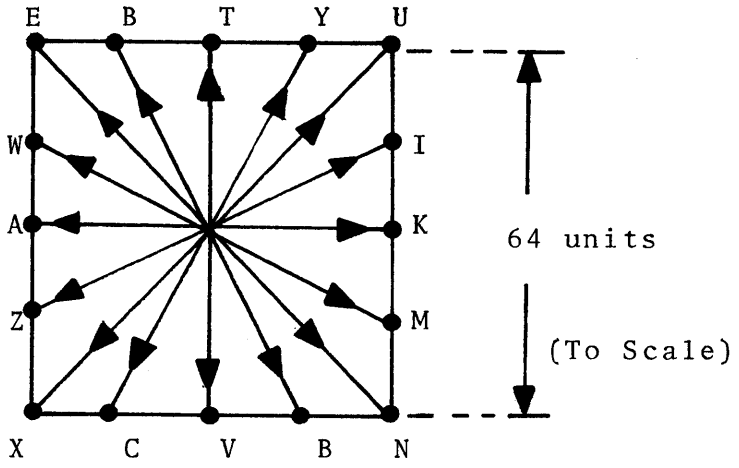
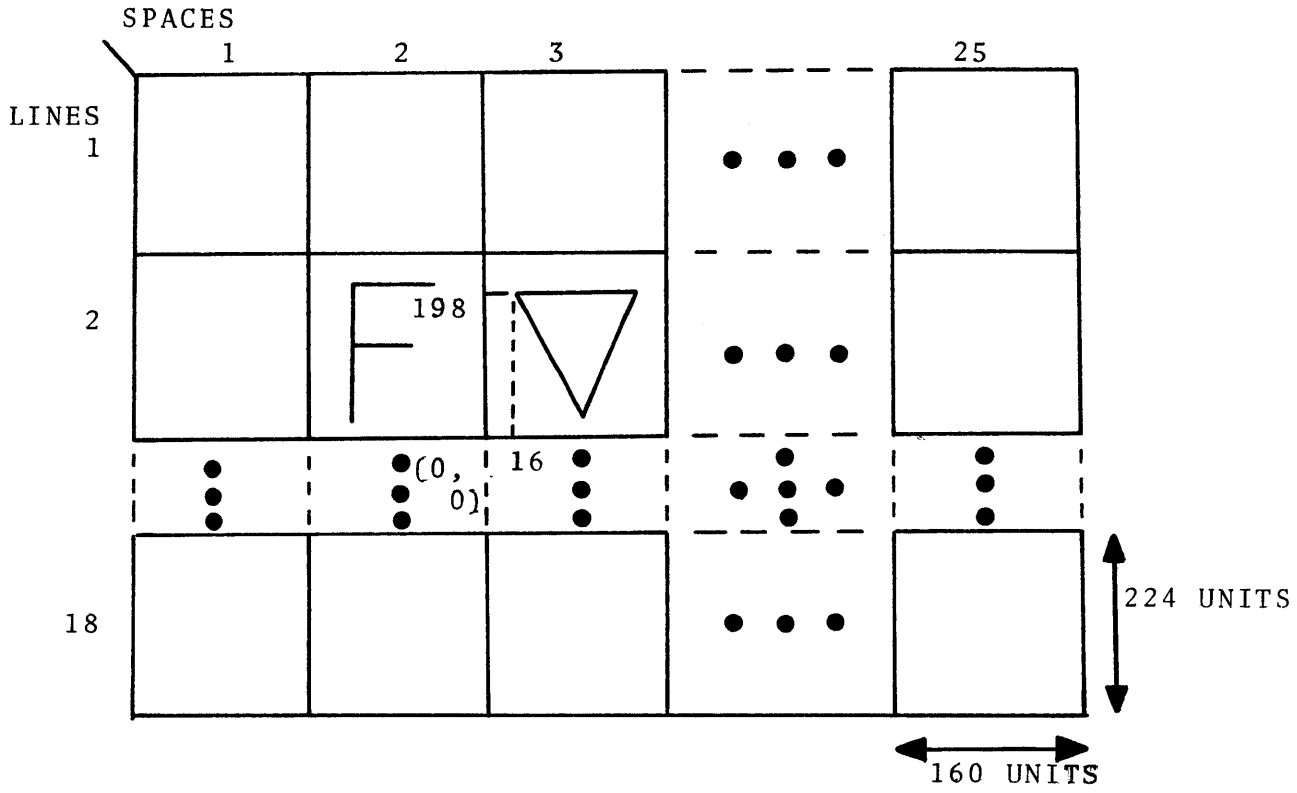


Figure 2.2 Sixteen possible directions and distances in which the dot can move.



(NOT TO SCALE)

Observe that there are 4096-18x224 = 48 units margin on the vertical scale and 4096-160x25 = 96 units margin on the horizontal scale.

Figure 2.3. Display Screen's Reference Grid Structure

This "dot instruction" moves the dot k_1 units horizontally and k_2 units vertically from the current origin determined by the carriage position. When RETURN is pressed the dot appears on the screen and becomes the first point stored in the character vector. Once the dot is positioned, it may be moved approximately 40 units at a time in one of 16 possible directions. The directions are marked by slashes on the keys A,Z,X,C,V,B,N,M,K,I,U, Y,T,R,E and W. Figure 2.2 indicates geometrically the direction and distance the dot moves for each key. When a direction key is pushed, the new dot position is connected to the old by a straight line. A sequence of direction keys pushed by the user determines the dot's locus. The user's task is to select the proper direction keys for the sequence to realize the desired pattern; for "$\bar{V}$" such a sequence is KKKKCCCCRRR.

If the user wants to start a new portion of a symbol which is not continuous with what he has already generated, he begins again with a dot instruction:

$.k_3,k_4$ RETURN

This instruction now moves the dot k_3 horizontal and k_4 vertical units from the last dot position. If the user wants to return to the first starting point, he simply presses RETURN.

To complete character generation, the user presses MOD for the second time and then stores the character in the desired CASE level and location. For the example, the instruction sequence would be MOD STORE CASE 5 D.

The entire instruction sequence, for the example, from start to finish is:

TYPE MOD . 16, 192 RETURN

KKKKCCCCRRRR MOD STORE CASE 5 D

After storage, the computer replies:

DONE

After the user has constructed a symbol and stored it, he may type it at any time on the TYPE level by pressing CASE followed by the level and the letter under which the symbol is stored. The starting point of the typed symbol is located at (k_1, k_2) relative to the present carriage position. The carriage automatically "moves" one space after a key is typed, even if the typed key is a created symbol which covers more than one space. Unless the user spaces the carriage appropriately a multi-space symbol will be overlapped by succeeding typed keys. A multi-space character may also extend into lines above and below the current one.

While the user is in character generation mode (i.e., after he pushes MOD the first time on the TYPE level and before he pushes MOD the second time) the following operations are available:

. k_1, k_2 RETURN	relocates the dot to a point k_1 horizontal and k_2 vertical units relative to the current terminal point in the character vector list, and displays the dot.
ERASE	erases the tube.
DISPLAY RETURN	displays the character currently in the character vector list.

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

L0 (okb. LI SHIFT), 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. L0 operators operate on two registers, the remainder and quotient registers. Most operators operate only on the quotient register, however, the ⊙ , the ⊘ , INV and the REFL operators modify the remainder register in addition to the quotient register.

2.3.1 L0 OPERANDS

(In the explanation below the following symbols are used: N - an unsigned integer, L - a storage location referenced by an alphabetic character.)

A. L — Indicates the use of L0 storage location L.

B. L+ — Indicates the use of L0 storage location L. L is incremented by one and stored after the operation. NOTE: This increment/decrement facility is particularly useful when indexing through arrays, vectors, tables, files, and strings.

C. L+N — The same as above except that the operand is incremented by N.

D. L- — The same as B except that the storage location is decremented by one.

E. L-N — The same as B except that the storage location is decremented by N.

F. N — The value N.

G. -N — The value minus N.

2.3.2 SUMMARY OF OPERATIONS

(A LO operand is represented by a T)

⊕ T, ⊖ T	performs the indicated operation on the quotient register.
⊙ T	multiplies 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.
⊘ T	divides the quotient register by T. 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".
SQ	squares the quotient register.
NEG or CONJ	negates the quotient register.
INV	inverts the quotient register leaving the remainder in the remainder register.
REFL	interchanges the remainder and quotient registers.
MOD	takes the absolute value of the quotient register.
DEL	places a one in the quotient register if it was zero, otherwise the register is set to zero.
LOAD T	loads the operand into the quotient register.
STORE L	the contents of the quotient register are placed in storage location L.
DISPLAY L	the contents of storage location L are displayed.
SUB, EVAL	the SUB and EVAL operators are discussed in the sections pertaining to the language which one is using i.e. COL, MOLSF, etc.

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, and user workspace in main storage is transferred to auxiliary storage to allow warmstart. The message "RESET COMPLETED" is displayed to signal successful completion of the reset operation. The user is then placed 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

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 L0 operand". A series of keys, including special operators, may be repeated with the sequence "REPT (keys) L0 operand". In both forms the L0 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 L0 operand may be replaced by another operand of the form "A=I,J,K RETURN", where A is a L0 storage location and I, J, and K are L0 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 (L0 DISPLAY A) A=1,7,2 RETURN

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

2.5 USER PROGRAMS - LIST MODE

Typically, the OLS user interacts manually with the primary operators defined by the on-line 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 on-line 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 the subsection on loading and storing files (subsection 2.1.2). 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 L0, 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 can be stored on any one USER system. This maximum

may be further restricted, however, due to the storage limitations placed on each of the eight user levels.

Once a USER program is stored, it can be treated like any other operator in the on-line system. Assuming we have a USER program stored in storage location USER LI COS, the storage location USER LI COS constitutes an operator which is composed of a key sequence stored at this address. In other words, a USER program is a user defined operator; and a USER system is a set of user defined operators. Like any of the OLS defined operators, a USER program is executed by pressing the key(s) which defines its storage address. The only difference is that a USER program must be preceded by the USER key and a level designation before the operator key is pressed. The level designation defaults to the previous USER level, if it is omitted.

EXAMPLE: To execute the USER program stored at the storage location LI INV. One would press:

USER LI INV

If one now wished to execute the USER program, USER LI DEL, he need only press:

USER DEL

2.5.2 LIST MODE

LIST mode is the means for constructing a USER program. A USER program may contain any key on the keyboard, except RESET or LIST. This means the user may use any operator he wishes and may execute other USER programs from a controling USER program.

To construct a USER program, the user presses the LIST key. The on-line system responds with the message "START LIST" and places the user in LIST mode. Until the on-line system is removed from LIST mode by pressing the LIST key a second time, all keys pressed are recorded in the exact order that they are pressed, but none of the operators are executed. As the user constructs his program, the keys he presses are displayed on the output device. This facility allows the user to check for errors in his program as he types it. When the USER program is completed, the USER presses the LIST key a second time to signal the on-line system that his program is completed. This puts the user on the EDIT level. Once on the EDIT level the user can store or modify his program.

Each key pressed in LIST mode becomes part of the USER program. Any keys pressed in an attempt to correct an error while in LIST mode will become part of the USER program. A USER program cannot be edited until the on-line system is on the EDIT level.

EXAMPLE: Write a USER program which evaluates e^{-x^2} and displays the function on the output device, for values of x stored in a REAL vector X.

LIST

LII REAL LOAD X SQ NEG EXP

DISPLAY RETURN

LIST

STORE USER LII EXP

NOTE: Storing a USER program is explained in subsection 2.6.2. When the keys USER LII EXP are pressed, all keys in the program are executed as if the user were manually pushing them. This user program may be included in another user program, such as:

LIST

LII REAL ID ⊙ 5 STORE X USER LII EXP

LIST

STORE USER LII LOG

When constructing a USER program, some considerations concerning length should be applied. The system programming that supports the running and embedding of subroutines by and in other subroutines is designed to allow USER programs to be short and easily combined for extensive computations. One should tend to think of a USER program as expressing one computational thought rather than solving a complete mathematical problem. Also, it is often useful to display a user program; therefore the size of the program should be limited to that which will fit on the output device. The system ultimately limits the length of a program, but that limit is, in most cases, beyond the desirable user-controlled limits mentioned above.

2.6 USER PROGRAMS - EDIT LEVEL

When writing USER programs, the user will occasionally push the wrong key or desire to alter an existing program. To avoid the tedious and frustrating job of rewriting an entire program until it is perfect, an editing level is provided. On the EDIT level a user can:

1. Store a USER program or name program.
2. Insert keys anywhere in the program.
3. Delete keys anywhere in the program.
4. Delete a block of keys anywhere in the program.
5. Transfer a block of keys from one part of a program to another.
6. Insert one USER program into another program.
7. Delete a program.

2.6.1 ACCESS TO THE EDIT LEVEL

The on-line system automatically enters the EDIT level when a user:

1. Leaves LIST mode, i.e. when he presses LIST to signal the on-line system that he has finished his program and the post list ":" symbol has appeared.
2. Has stored his program.
3. Has displayed a program.
4. Has loaded a program.

Thus errors can be corrected when a program is first written or a program may be called back later for modification.

2.6.2 STORING A USER PROGRAM

Once on the EDIT level a user stores his program by the sequence:

STORE USER ("level") ("operator")

The USER key identifies the program as a user defined operator, the level key indicates which level it is stored on, and the operator key is the final qualification necessary to locate the program. As discussed in the section on the "STRUCTURE OF THE USER SYSTEM" the level may be qualified by the CTX key. In this case the sequence to store a program is:

STORE USER ("level") CTX ("operator")

EXAMPLE: Assume a user has just finished a program by pressing a second LIST and wishes to store it on USER level I under the DIFF key. To accomplish this the user would press:

STORE USER LI DIFF

To store the program on the CTX level of qualification the user would press:

STORE USER LI CTX DIFF

2.6.3 DISPLAYING OR LOADING A PREVIOUSLY STORED USER PROGRAM

A user program is displayed by the sequence:

USER ("level") DISPLAY ("operator")

or if it is stored on the CTX level of qualification by:

USER ("level") DISPLAY CTX ("operator")

NOTE: the USER key and the level designation must be pressed before the DISPLAY key is pressed.

EXAMPLE: Display the program stored under USER LIV SQ. The user would press:

USER LIV DISPLAY SQ

EXAMPLE: Display the program stored under USER LIII CTX INV. A user would press:

USER LIII DISPLAY CTX INV

Displaying a USER program puts the on-line system on the EDIT level. The user may now modify or store the programs.

One may load a USER program without displaying it by the sequence:

USER ("level") LOAD ("operator")

or if it is stored on the CTX level of qualification by:

USER ("level") LOAD CTX ("operator")

The user may now modify or store the program.

2.6.4 MODIFYING A USER PROGRAM

Each editing operation must take place at a specific point in the program, which must be manually specified by the user. As the name LIST implies, any on-line program is a list of

keypushes. Each key pressed is one item in a list which defines a program. Thus the USER program "LO LOAD 1 STORE IJ USER LI DEL" may be thought of as the list of keys:

LO	LOAD	1	STORE	I	J	USER	LI	DEL	:		

where ":" is the post list mark. To modify any program, a user must position a pointer called the edit pointer to the position where he wishes to modify the program. All modifications to a program are entered before the key delineated by the edit pointer.

When the program is written for the first time or a previously stored program is displayed, the edit pointer is automatically located at the end of the program. The present position of the edit pointer can be displayed at any time by pressing EVAL. The location of the edit pointer is indicated by underscoring the key to the right of the edit pointer.

EXAMPLE: Assume that the program above has just been entered and the user has pressed LIST the second time to signal the on-line system that he has finished his program. Pressing EVAL would underscore the post list marker, because the edit pointer is automatically positioned to the end of the program.

The following operators move the edit pointer to specified positions:

⊕	moves the edit pointer one key towards the end of the program.
⊕ N RETURN	moves the edit pointer N keys towards the end of the program. N must be an integer.

⊖	moves the edit pointer one key towards the beginning of the program.
⊖ N RETURN	moves the edit pointer N keys towards the beginning of the program. N must be an integer.
REFL or UP or ENL	positions the edit pointer to the beginning of the program and underscores the first key.
CON or DOWN	positions the edit pointer to the end of the program and underscores the post list marker.

At the conclusion of any of these operators the on-line system is still on the EDIT level.

To locate a key sequence within the program one may use the MOD operator. MOD is used to search for a unique key sequence. If a nonexistent key sequence follows the MOD operator, the diagnostic "NONEXISTENT STRING" is displayed. MOD must be followed by as many keys as necessary to locate a unique key sequence. When a unique sequence is found, the first key in the sequence is underscored and the edit pointer is positioned before it. If the user wishes to stop the search before a unique key sequence has been designated, he may press LIST; LIST aborts the search. The on-line system remains on the EDIT level.

2.6.5 INSERTION OF KEYS IN A USER PROGRAM

Once a user has properly positioned the edit pointer, he presses the ENTER key to enter LIST mode to insert new keys. Each new key is entered before the edit pointer. The edit

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

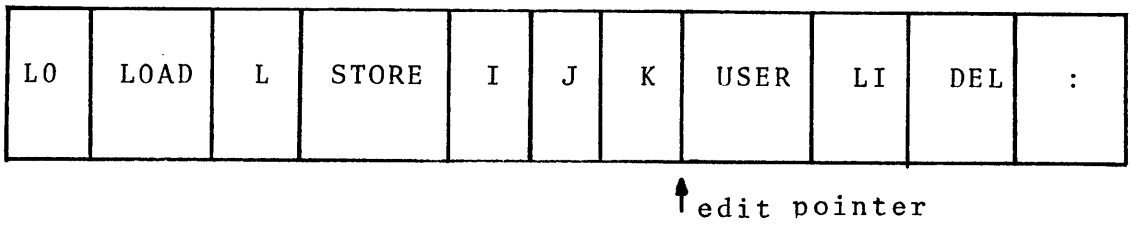
ID DISPLAY RETURN

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:



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.
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	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, deletes n successive keypushes. If no integer is given, one keypush is deleted for each depression of SPACE or BACK. The deleted key is blotted out on the output device.

The 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.
BACK	deletes CMPLX.
⊕ 2	locates the edit pointer between SQ and ⊘ .
EVAL (optional)	displays the location of the edit pointer.
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 OLS 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.
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 II 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 a pointer will appear on the output device. If that key appears more than once, then succeeding keys must be pressed until identification of the pointer location is uniquely determined. A pointer will not be displayed until 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 EDIT mode.
⊕	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 displays it.
CON or DOWN	moves the edit pointer to the end of the program.
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 pointer in a temporary location called the right side save (RSS) area.

STORE LS — stores everything to the left of the edit pointer in a temporary location called the left side save (LSS) area.

STORE RETURN — stores left side and right side.

STORE USER (level) (operator)

stores contents of list buffer in specified storage location.

LOAD RS — inserts into the program at the left of the edit pointer the keys stored in the right save (RSS) area.

LOAD LS — inserts into the program at the left of the pointer the keys stored in the left side save (LSS) area.

LOAD USER (level) (operator)

inserts into the program at the pointer's left the specified user program.

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 on-line 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. The number being tested (henceforth denoted by N_T) is dependent upon the current level as follows:

1) Level 0 - N_T is the number in the level 0 quotient register.

2) Level I REAL - N_T is the number in the β_I working register.

3) Level I CMPLX - N_T is the number in the α_I working register.

4) Level II REAL - N_T is the first component of the β_{II} working register.

5) Level II CMPLX - N_T is the first component of the α_{II} working register.

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

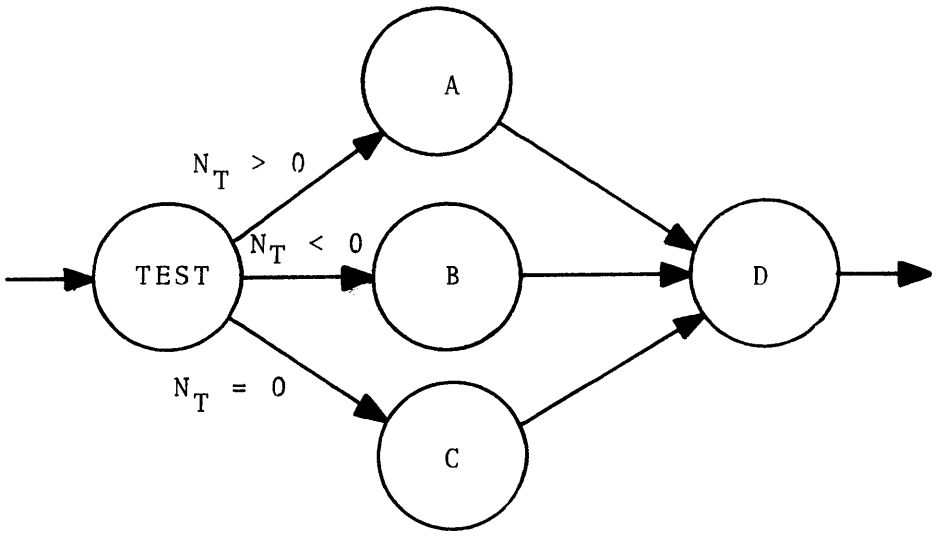


Figure 2.7.1 Flow chart branching for TEST program:
TEST + (A) - (B) 0 (C) D

As indicated in the figure the basic format provides a three-way branch for the program depending on the test condition. The key-

push sequence in these branches are executed and then control is returned to the main program.

Assume the following keypush sequence for A, B, C, and D respectively.

A = (TYPE POSITIVE RETURN USER LI ⊕)

B = (TYPE NEGATIVE RETURN USER LII SIN)

C = (USER LI SQ)

D = (TYPE RETURN END)

The key sequences executed for the various conditions are as follows:

1) $N_T > 0$, execute TYPE POSITIVE RETURN USER LI ⊕ followed by TYPE RETURN END.

2) $N_T < 0$, execute TYPE NEGATIVE RETURN USER LII SIN followed by TYPE RETURN END.

3) $N_T = 0$, execute USER LI SQ followed by TYPE RETURN END.

It is not necessary to specify branches for all three conditions; the first unconditional sequence occurring after the TEST key signifies the end of the TEST operation. For example, the instructions

TEST + (A) - (B) D

would result in the branching depicted in Figure 2.7.2. In this case for the condition $N_T = 0$, control is returned to the main program without execution of an intermediate instruction set.

Multiple conditions may preceed the key sequence to be executed. Thus TEST + - (A) 0 (C) D is a valid TEST operand.

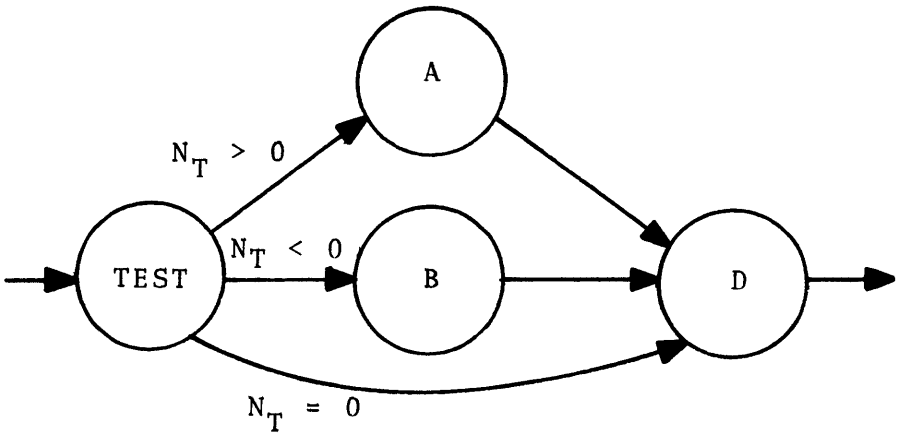


Figure 2.7.2 Flow chart for: TEST + (A) - (B) D

THE TEST OPERATOR RS

When the upper keyboard button RS is inserted between the condition (+, -, or 0) and the conditional sequence, it acts as a RESET operator which clears the execution list of all pending buttons and causes only the keypushes in the parentheses immediately following RS to be executed. Typically, RS might be employed as shown in the following TEST program

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

and the flow diagram of Figure 2.7.3. For this case the branching is the same as in Figure 2.5.1, except when $N_T > 0$. Under this condition all keypushes after A are cleared and only A is executed.

If the same sequences are used for A, B, C, and D as in the subsection covering the basic test format the following

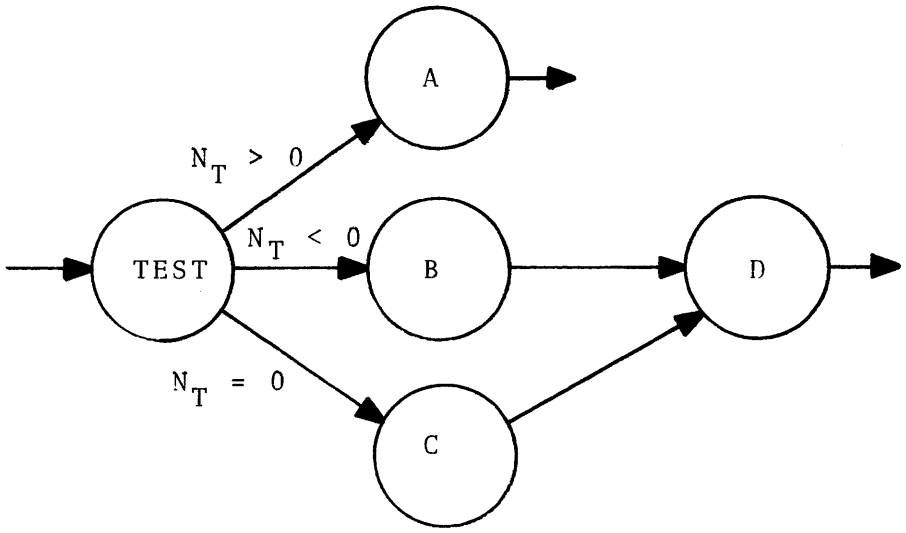


Figure 2.7.3 Flow Chart for: TEST + RS (A) - (B) 0 (C) D

branches occur:

1) $N_T > 0$, reset the execution list (clear all pending buttons) and execute

   TYPE POSITIVE RETURN USER LI + .

2) $N_T < 0$, execute TYPE NEGATIVE RETURN USER LII SIN followed by TYPE RETURN END.

3) $N_T = 0$, execute USER LI SQ followed by TYPE RETURN END.

THE TEST OPERATOR NEG

The TEST operator NEG allows a series of keypushes to be suppressed until a specified sequence occurs. If the test condition is met, the NEG operator is interpreted as "skip to" the sequence matching the sequence in parentheses. Consider the example shown below and flow charted in Figure 2.7.4

X TEST - NEG (Y) WYZ

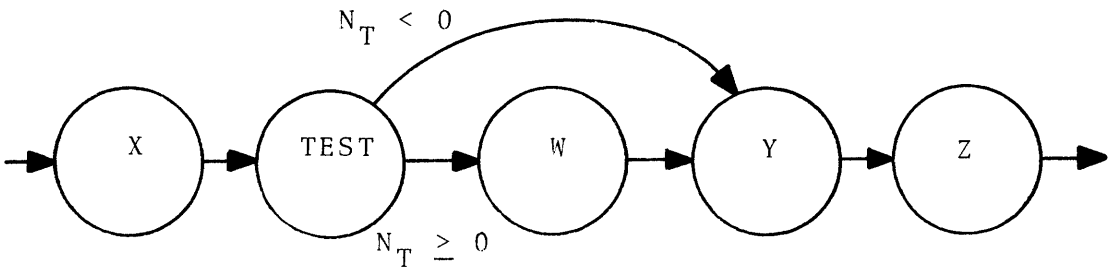


Figure 2.7.4 Flow Chart for: X TEST - NEG (Y) WYZ

As indicated in the figure, if $N_T \geq 0$ all sequences X W Y Z will be executed as if TEST - NEG (Y) did not exist. If N_T is less than 0, keypush sequences XYZ will be executed.

This operator makes it easy to avoid repetition of a sequence which is to be executed for two of the three test conditions. The sequence in the example above accomplishes the same purpose as the longer sequence:

X TEST + (W) 0 (W) YZ

NOTE: the following sequence is also acceptable.

X TEST + 0 (W) YZ

If, after the TEST NEG* operator, there is no keypush sequence in the execution list to match the sequence in parentheses, the diagnostic "TEST ERROR" is displayed on the output device.

LO OPERANDS WITH TEST

Integers or level 0 operands can be used with TEST to provide the capability of skipping a number of keypushes in a program. For example, in the program

TEST + 3 - 6 USER LI RS USER LI LS USER LI SQ

the following will occur depending on the TEST conditions.

1) $N_T = 0$, all keypushes will be executed.

2) $N_T > 0$, the first three keypushes USER LI RS will be skipped and the sequence USER LI LS USER LI SQ executed.

3) $N_T < 0$, the first six keypushes will be skipped and the series USER LI SQ executed.

Level 0 operands may be used instead of integers, and be changed, manually or automatically, between executions of the TEST program, providing a convenient means for changing a computational sequence. NOTE: with this form of TEST the lower keyboard button 0 is always interpreted as a test condition and thus may not be used as part of an operand.

USE OF PARENTHESES WITH TEST

The set of possible keypushes within these parentheses includes all buttons except RESET or LIST, with the requirement that for every embedded left parenthesis "(", there must be a closing or right parenthesis ")". This allows one to employ embedded TESTs in conjunction with the TEST operators to provide more sophosticated branching capabilities in a program.

As an illustration of the use of parentheses with the TEST operator, a TEST program is shown below and flow diagrammed in Figure 2.7.5.

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

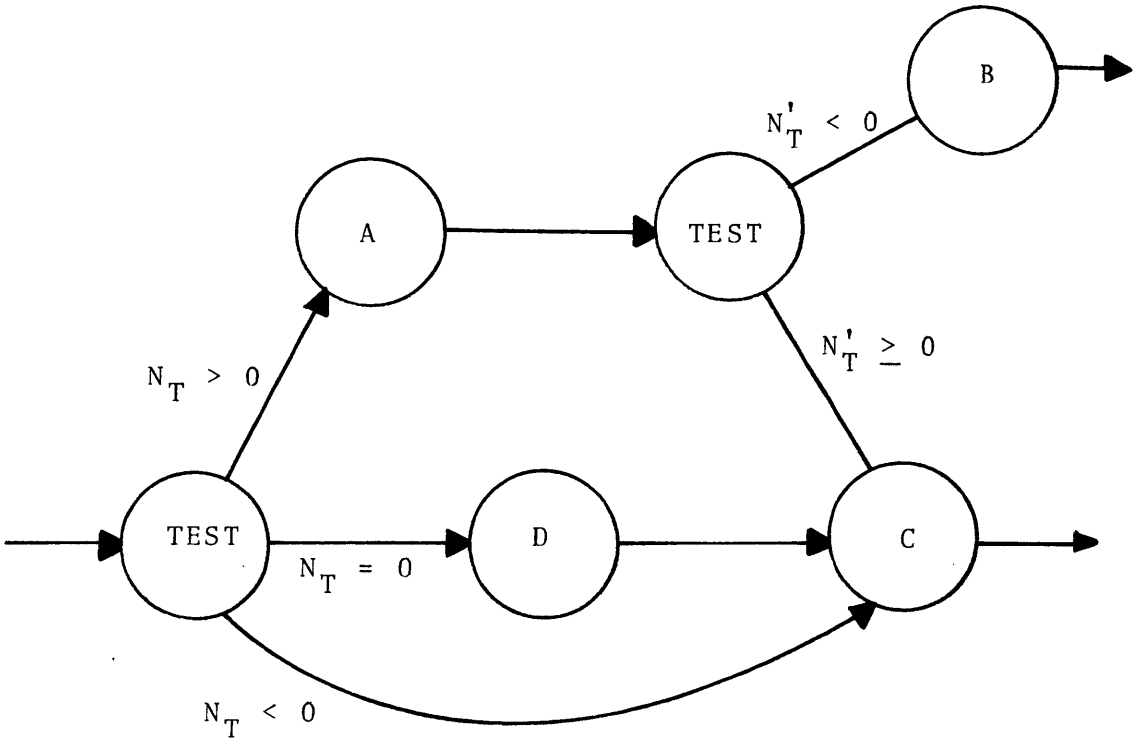


Figure 2.7.5 Flow Chart for: TEST + RS (A TEST - RS (B) C) 0 (D) C

As shown in the flow chart the program branches to D and then executes C if $N_T = 0$. If $N_T < 0$, the program branches directly to C. When $N_T > 0$ the RS operator suppresses C and the button sequence to A is executed, followed by a second TEST instruction. The TEST parameter N_T' for the second test is determined by the sequence A. If $N_T' < 0$, the RS operator suppresses C again, and B is executed. If $N_T' \geq 0$, C is executed. (In this example, even though the sequence C occurs twice, it cannot be executed more than once.) Note that in the embedded TEST sequence, execution is initiated starting from the outer set of parentheses.

2.7.3 THE PRED KEY

PRED is a special user program operator which allows arguments to be passed to a user program. The PRED button followed by a single non-zero digit or L0 (index) data location name tells the system the number of the argument which is to replace PRED and its operand. Arguments are passed to a user program by specifying a parenthesized list of the form (arg_1, arg_2, arg_3, ...) following the user program call. Each argument may consist of any series of keys except RESET, LIST, ")", or ",". An argument may be omitted; however, the preceding comma must be pushed to assure correct numbering of arguments. An omitted argument which has its preceding comma included will be replaced with the null string. Before execution of a program containing PRED the system replaces the PREDs and their operands with the corresponding buttons passed in the argument list. Since the replacement proceeds from the end of the user program to the beginning, PREDs may be nested, for example "PRED PRED 1". The sequence "PRED 1" is replaced first, creating a new operand for the first PRED. If the argument list is not supplied to a user program containing PRED, the message "PARAMETER LIST NOT FOUND" will be displayed. If an individual argument is not supplied the message "UNLOCATABLE PARAMETER(S)" will be displayed. The latter message will also occur if the PRED operand is invalid. NOTE: a paired set of parentheses may appear in an argument; a comma may appear in a parenthesized argument.

EXAMPLE: The following program computes the indefinite integral of a real function using the formula:

$$\int F dT = \Sigma(F + \Delta F/2 - \Delta^2 F/12)\Delta T$$

The function F and independent variable T can be specified at execution time.

LIST

LII REAL LOAD PRED 1 ⊕

(⊖ (DIFF ⊘ 6) DIFF ⊘ 2) ⊙ (DIFF

PRED 2) RS LI LOAD 0 SUB 1

LII SUM:

STORE USER I SUM

The button sequence USER LI SUM (F,X) would then compute the indefinite integral ∫FdX.

EXAMPLE: This example converts a decimal integer to any base less than or equal to 16.

USER LI ⊕

TYPE RETURN BASE ? L0 LOAD ENTER DISPLAY SPACE

RETURN STORE B TYPE RETURN NUMBER ? L0 LOAD

ENTER DISPLAY SPACE RETURN STORE N TYPE RETURN RETURN

BACK USER LI REPT ⊖ 25 RETURN :

USER LI ⊖

L0 LOAD N ⊘ B REFL ⊕ 1 STORE A USER LI ⊙

(0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F) L0 REFL STORE N USER :

USER LI ⊙

TYPE PRED A BACK BACK :

This program calculates a number base B by printing:

$$N \text{ MOD } B^{24} \quad N \text{ MOD } B^{23} \quad \ldots \quad N \text{ MOD } 1$$

USER LI ⊕ has the user enter the base and the number after which it repeats USER LI ⊖ 25 times. USER LI ⊖ calculates the number to be printed and then calls USER LI ⊙ . USER LI ⊙ prints the A^{th} argument. The loop is then repeated.

2.7.4 REPETITION OF PROGRAMS; LOOPING

Of particular importance in the use of user programs is looping capability so that a program can be repeated a number of times, if desired. The simplest method of looping and the least desirable is to program a user subroutine so that it calls itself. For example, the following program stored under USER LI ⊕

TYPE RETURN YEA USER LI ⊕

references itself. Consequently the program will go into an infinite loop and continue to type the word "YEA" on the display scope until the RESET button is pushed. Programs of this form are to be avoided since they use computer time wastefully. Care should be taken in the construction of subroutines to insure that this situation does not occur.

The REPEAT key (REPT) can be employed to cause programs to be repeated a specific number of times. For example, the following

program:

TYPE RETURN BOO USER LI

stored under USER LI COS will type the word "BOO" on the display scope five times when the key sequence

USER LI REPT COS 5 RETURN

is typed. This use of the REPT key is completely consistent with the description in Section 2.4.3. The appropriate level (USER LI) is made current before the REPT key is pressed, and the operator is COS. Note that the program ends with USER LI. It is important to remember than when the operator to be repeated is a user program the current level is changed during execution and must be restored at the end of the program. Thus, since

USER LI REPT COS 5 RETURN

is equivalent to

USER LI COS COS COS COS COS,

the second COS will be executed as an operator on the level current at the end of USER I COS, etc.

The TEST operator can be used very effectively to control looping, either (a) to repeat a program a specified number of times, as in the example above, or (b) to iterate a computation until some value in the computation reaches a particular limit.

a) To have the program in USER LI COS, above, repeat itself k times, where k is a non-negative integer assigned to INDEX level K, the program USER I COS would read

TYPE RETURN BOO LI SHIFT LOAD K ⊖ 1

STORE K TEST + (USER I COS)

b) To iterate a program, say USER LII ⊕ , until two successive values R_j and R_{j-1}, stored in LI REAL R and S, differ from each other by less than a constant value stored in C, the program would end

. . . LI REAL LOAD R ⊖ S MOD ⊖ C

TEST + (LOAD R STORE S USER LII ⊕)

2.7.5 NAME PROGRAMS

As a mnemonic aid, the user may give a USER program a descriptive name. Once a user has named a USER program, he may refer to the program either by its name, or by the operator key where it is stored.

To construct a name program the user presses LIST followed by any of the alphabetic or numeric characters on the TYPE level. He finishes the program by pressing LIST. He stores the name program by the sequence:

STORE USER (level) DISPLAY (operator)

EXAMPLE: Assume there is a program which evaluates integrals stored under USER LV MOD and that we wish to name the USER program "INTEGRAL". The name program would be constructed and stored as follows:

LIST INTEGRAL LIST

STORE USER LV DISPLAY MOD

The user can now display the name program by the sequence:

USER LV DISPLAY DISPLAY MOD

or equivalently:

USER LV DISPLAY DISPLAY INTEGRAL RETURN

He could now execute the integral program USER LV MOD with the key sequence:

USER LV INTEGRAL RETURN

The above examples use RETURN to terminate a name; however, any key which may not be included in a name program will terminate the call. The name of the USER program may be used anywhere the operator key may be used. If one makes an error typing in the program name, he may erase the preceding key by pressing BACK, or start the name over by pressing SPACE.

When the console is in LIST mode and a reference is made to a USER program which has a name program associated with it, the name is displayed instead of the operator key. If this is the only purpose for which one wishes to use a name program, one may include special characters in the name program. A name program used to call a USER program may not contain any special characters.

Just as a USER program may end with a call to the USER levels, a name program may end with a call to a USER level. A name program's call to a USER level is not related to execution; it is solely for display purposes in LIST mode. When a name program ends with a call to USER level and one is in LIST mode,

the on-line 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 ORIENTATED 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 LI as it requires knowledge of the operators on levels II and III.

3.1 BASIC CONCEPTS

COL has four levels. On each level the operators manipulate a specific data structure. COL utilizes three data structures: files, records, and character strings. They can be defined as:

File ≡ ordered set of records

Record ≡ a character string with a declared length

Character string ≡ an ordered set of characters

Valid characters are Latin alphabet, Greek alphabet, digits, and special characters. Level IV operators provide an interface with the operating system facilities. Files can be submitted to the operating system for processing. Files can be created from operating system data sets. Level III operators are for manipulating files. Level II operators are for creating and manipulating records. Level I operators are for creating and manipulating character strings.

Associated with each data structure are work areas, markers, and pointers. There are two working files, the REAL file and the COMPLEX file. When one is using COL one of these files is active and the other is inactive. The user declares which of these files he wants to be the active file by pressing either the REAL key or the CMPLX key. It's important to keep track of which file is active, because the active file is the destination for all file manipulation operators, and it is the source and destination for many record and string manipulation operators. If the active file is not explicitly declared, it is assumed

to be the REAL file. Associated with each file is a marker, called the active file marker. The active file marker is designed to aid the user in loading, storing, displaying, and operating on files and records without having to explicitly declare which record within a file he wishes to access. Use of the active file marker will be discussed below.

There are also two record work areas, the active buffer and the save buffer. As its name implies the active buffer is where records are created or modified. It is also the source for storing records in the active file. The save buffer can be thought of as a holding area. When a record is loaded from the active file, it is placed in the save buffer. It can then be transferred in whole or part to the active buffer for modification and storage. Each buffer has its own pointer, called the active buffer pointer and the save buffer pointer, respectively. The active buffer pointer locates the character position where the next character will be inserted or deleted. The save buffer pointer locates the first character to be transferred to the active buffer when part of the save buffer is transferred to the active buffer.

Both level I and level II use the active buffer as their work area. On level II the active buffer has a declared length, set by the user. On level I the character string has a varying length. (NOTE; this is the key distinction between a record and a character string. A record has a declared length, while a character string has a varying length, which may be zero.)

When a user signs on COL the record length on level II is set to 80 characters. The user may change the record length for his own purposes. The record length can be set to any value between 1 and 254. 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.

Intermediate level I storage is provided by the fifty-two alphabetic storage locations, A-Z and α-ω. Each one of the storage locations is initially set to the null string. When a string is stored its length as well as its 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.

EXAMPLE: Valid literal operands

'BROWN,J.C. RETURN

'MY SPACE FAIR SPACE LADY RETURN

The latter operand would be displayed as "MY FAIR LADY".

NOTE: No closing apostrophe is required; in fact, another apostrophe would be treated as a valid character and added to the character string.

Alphabetic operands are used to reference the fifty-two storage locations defined by the lower keyboard keys. The standard form for an alphabetic operand is:

alphabetic [,character number] [,number of characters]

(NOTE: brackets indicate optional keys.) The alphabetic key describes which storage location is to be used. The character number and number of characters enable one to extract a substring from the indicated storage location. If the character number is omitted it is assumed to be one. If the number of characters is omitted the remainder of the string is used.

Interlevel operands are used to reference records stored in the active file. The standard form for an interlevel operand is:

LII record number [,column number] [,number of characters]

The record number defines which record in the active file is to be used. The column number and number of characters enable one to extract a substring from the indicated record. The record number, column, number, and number of characters may be integers

or level 0 operands. If the column is omitted, it is assumed to be one. If the number of characters is omitted, the remainder of the record is used.

Examples of the operand forms are given in the next subsection.

3.2.2 DISPLAYING, LOADING, AND STORING OPERANDS

Most operators affect only the contents of the string buffer. Thus it is necessary to move the appropriate string into the string buffer before manipulating it. LOAD and DISPLAY enter a character string into the string buffer. They provide non-destructive recall from the level I storage locations or the active file. LOAD moves the indicated operand into the string buffer; the previous contents of the string buffer are lost. DISPLAY loads data into the string buffer and displays it. The following examples all use LOAD, however DISPLAY could be substituted for LOAD.

LOAD 'BROWN,J.C. RETURN

The literal operand "BROWN,J.C." is entered into the string buffer.

LOAD A RETURN

The contents of storage location "A" are entered into the string buffer. The contents of "A" are not altered.

LOAD A,7,3 RETURN

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 N^{th} 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 M^{th} 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 will start with the first character of the string buffer and search right for the specified operand. LS followed by any level I operand searches the string buffer starting at the end of the string buffer. MOD followed by a character string searches for the character string until a unique character string is found. If MOD cannot find the character string, then the diagonistic "NO SUCH OCCURRENCE" is displayed. For all three operators, if no match is found the level I search pointer is set to zero. If a match is found the search pointer is set to the character number of the first character of the match. If there are multiple occurrences of the string the level I operand may be followed by a level 0 operand to specify which occurrence of the string is desired.

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 collating sequence. COL uses the IBM 360 collating sequence. This sequence can be viewed by pressing:

ID DISPLAY RETURN

ID loads a string consisting of all valid characters into the string buffer. The string buffer and operand are compared 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 terminated 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 GEORING 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

within the active buffer at which the next character will be stored. Each time a character is stored the value of the active buffer pointer is automatically incremented by one. Thus successive characters are stored in successive locations within the active buffer.

The user can exercise manual control over the active buffer pointer, thus facilitating correction of errors, insertion of characters, and deletion of characters. BACK replaces the preceding character with a blank and decrements the active buffer pointer by one. LS sets the active buffer pointer to 1. The key sequence:

⊕ N RETURN

increments the active buffer pointer by N. The key sequence:

⊖ N RETURN

decrements the active buffer pointer by N. The key sequence:

⊙ N RETURN

sets the active buffer pointer to N. Finally, the operator MOD followed by a character string will search the active buffer for that character string. The user must continue pushing characters until a unique character string is found or no match is found. When a unique character string is found, the active buffer pointer is set to that value and the value is displayed. If no match is found, the diagonistic message "NO MATCH" is displayed.

Once the active buffer pointer is properly positioned, previous characters can be replaced, new characters can be inserted, or characters can be deleted. To replace existing characters enter lower keyboard keys exactly as if one were creating a record. To insert a string of characters between existing characters, press ARG followed by the characters to be inserted. RETURN or any upper keyboard key will terminate the character string being inserted. The active buffer pointer is not changed. To delete a character string press DEL followed by the number of characters to be deleted from the active buffer. If no number follows DEL the remainder of the active buffer is set to blanks. DEL also shifts the remaining characters in the active buffer to the left to fill in for the deleted characters. DEL is terminated by RETURN or any upper keyboard key. RS sets the active buffer to blanks; it does not change the active buffer pointer.

The contents of the active buffer may be displayed at any time by the key sequence:

DISPLAY RETURN

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

EXAMPLE: Constructing a record destined for inclusion in a FORTRAN source program. The desired record is:

```
REAL*8 X,Y,Z(100)
```

Assuming one has signed on COL, he must first access the record creation level and then start the new record in position 7 (by convention the first non-blank character in a FORTRAN declaration must appear in or after column 7). One possible sequence:

LII ⊕ 6 RETURN REAL*8 SPACE X,Y,Z(100)

Note, since the default record length is 80, the user did not have to change it. Secondly, the initial value of the active buffer pointer is 1, so to set it to 7 one simply adds 6. An equivalent key sequence is:

LII ⊙ 7 RETURN REAL*8 SPACE X,Y,Z(100)

To verify the results the user can press DISPLAY RETURN.

3.3.3 AUTOMATIC SKIP, DUPLICATE, OR LEFT ZERO FEATURE.

An automatic numeric entry feature is available on LII which will help create records. This feature is initiated and terminated by the ⊘ operator. (This operator corresponds to the automatic skip-dup switch on a keypunch machine.) It has two operands.

1) ⊘ + Initiates the automatic entry.

2) ⊘ - Terminates the automatic entry.

When the system is in automatic entry it uses a "drum card" to determine if and what automatic service should be accomplished. There are three possible services:

1) Skipping a particular field of the active buffer.

2) Duplicating a particular field from the save buffer into the active buffer.

3) Allowing a number to be entered that is right adjusted, padded with zeroes, and placed in a field of the active buffer.

The "drum card" contains a template that defines each field, Each field begins in a certain column and continues for a certain length Whenever the system finds that it is at the beginning of a field it begins the service. The fields are defined by using the SQ operator. This operator requires three operands separated by commas and followed by a RETURN:

1) The column number in which the field is to begin.

2) The length of the field.

3) The type of field abbreviated by its initial: S for skip, D for duplicate and N for numeric.

The SQRT operator clears the drum card and thus nullifies any previous SQ operations. When the system encounters the beginning of a numeric field it requires an entry be specified in one of the following ways:

1) A series of lower keyboard blue buttons followed by a RETURN that specifies the number.

2) A lower keyboard letter followed by a RETURN that specified a L0 operand which is to be used as the entry in that field.

3) A RETURN that specifies that the L0 accumulator is to be used as the number.

EXAMPLE: Problem - Prepare a file of ten cards. Each card should contain the word NUMBER in columns one through six. In columns 10 through 12 put a number padded with zeroes. This

number should be 1 on the first card, 2 on the second, etc.

Solution 1: Push the following: (NOTE: STORE is defined in the next section).

```
LII SQRT SQ 1,6,D RETURN
SQ 7,3,S RETURN
SQ 10,3,N RETURN
RS LS NUMBER Ø +
1 STORE 2 STORE 3 STORE 4 STORE 5 STORE
6 STORE 7 STORE 8 STORE 9
STORE 10 STORE Ø -
```

Solution 2:

```
LII SQRT SQ 1,6,D RETURN
SQ 7,3,S RETURN
SQ 10,3,N RETURN
RS LS NUMBER Ø +
REPT (A STORE) A=1,10↙ Ø -
```

3.3.4 FILE CREATION

Since every record is created in the active buffer, a finished record must be removed from the active buffer and stored before the next record can be created. A finished record is always stored in the active file. There are three operators for storing records in the active file: STORE, SUB, and UP. STORE stores the record in the active buffer at the end of the active file. SUB replaces the specified record in the active file with the contents of the active buffer. The record to be replaced can be

specified explicitly or implicitly. If SUB is followed by a level 0 operand, then that value is taken to be the record to be replaced. If SUB is followed by RETURN or an upper keyboard key, then the record to be replaced is assumed to be the value of the active file marker. UP works exactly like SUB except that the active buffer's contents are placed before the specified record. Also, all the operators for storing a record copy the active buffer to the save buffer, set the value of the active buffer pointer and save buffer pointer to 1, return the display carriage, and set the active buffer to blanks.

3.3.5 LOADING AND DISPLAYING RECORDS

Normally record modification is a three step procedure. First, the record must be loaded into the active buffer; second, the active buffer must be modified; and finally, the record must be stored. The second and third steps have been discussed in the preceding sections. This section completes the procedure. If the record is newly created, then it is already in the active buffer and ready for modification. Otherwise the record must be found, loaded into the save buffer, and transferred to the active buffer. A record may be found manually by displaying the active file, or automatically by searching the active file for some identifying field in the record. The latter is accomplished by the level III EVAL or MOD operators (section 3.4.3). A record in the active file is displayed by pressing DISPLAY followed by an integer or level 0 operand, followed by RETURN. The specified record is displayed, the record is copied into the save buffer, and the active file

marker is set to the value of the record displayed. Successive RETURN's display successive records, copy them into the save buffer, and set the active file marker to the number of the record displayed. BACK displays the preceding record, copies it into the save buffer, and sets the active file marker to the number of the record displayed.

When one has found the record to be modified, the active file marker will be set to its value and a copy of the record will be in the save buffer. Thus to complete the first step the user needs only to transfer the contents of the save buffer to the active buffer. REFL copies of the save buffer to the active buffer. INV switches the contents of the active and save buffers.

3.4 LEVEL III - A FILE MANIPULATION LEVEL

The operators on level III enable the user to delete blocks of records from the active file, to purge the entire active file, to insert blocks of records from the inactive file into the active file, and display blocks of records from the active file.

3.4.1 MOVING RECORDS FROM THE INACTIVE FILE TO THE ACTIVE FILE

⊕ concatenates the inactive file onto the active file. The inactive file is purged.

UP inserts all of the inactive file into the active file. UP followed by an integer or level 0 operand followed by an upper keyboard key or RETURN specifies that the inactive file is to be inserted before the specified record. If no record is specified, the inactive file is inserted before the record defined by the active file marker. The inactive file is not changed.

ARG inserts a block of records from the inactive file to the active file. ARG may be followed by three operands: ARG M,N,Ø RETURN. The first operand specifies the number of records to be copied. It is required. The second operand specifies the destination of the block of records. They will be inserted before the N^{th} record in the active file. This operand may be omitted; when it is omitted, it is assumed to be the record indicated by the active file marker. The third operand specifies the source of the block of records. They will be copied from the inactive file beginning with the $Ø^{th}$ record. This

operand may be omitted; when it is omitted, it is assumed to be the value of the inactive file marker. The inactive file is not changed by this operator.

3.4.2 DISPLAYING BLOCKS OF RECORDS IN THE ACTIVE FILE

The entire active file may be displayed by pressing DISPLAY RETURN. A subset of the active file may be displayed by pressing DISPLAY M,N RETURN. M specifies the number of records to be displayed. N specifies the first record to be displayed. Both M and N are optional. If N is omitted its value is assumed to be the value of the active file marker. If M is omitted, the remainder of the active file is displayed beginning with the record indicated by the second operand.

EXAMPLE: The key sequence:

DISPLAY 5,99 RETURN

will display five records from the active file beginning at the ninety-ninth record.

EXAMPLE: Given that the active file marker has a value of 12, the key sequence:

DISPLAY 7 RETURN

will display seven records from the active file beginning with the twelfth record.

EXAMPLE: Given that the active file marker has a value of twelve, the key sequence:

DISPLAY , RETURN

will display the last part of the active file beginning with the twelfth record.

3.4.3 SEARCHING THE ACTIVE FILE

The EVAL operator enables one to search the active file for a designated character string. The format of the EVAL operator is: EVAL N RETURN operand RETURN. N specifies the column number of the first character of the character string one is searching for. The second operand, which is the character string one is searching for, is any valid level I operand (see section 3.2.1). If the specified character string is found, the record number of the record is displayed and the active file marker is set to that number. If the specified character string is not found, the active file marker is set to zero and "0" is displayed. The search for the second operand begins with the record after the record indicated by the value of the active file marker. Thus it may be necessary to set the active file marker before the search. The active file marker is set by the ⊙ operator.

EXAMPLE: To set the active file marker to 23 one would press:

⊙ 23 RETURN

EXAMPLE: One wishes to search the active file for the character string SMITH which should start in column 17. Assume one is not sure of the value of the active file marker and wants to start the search with the first record. One possible sequence is:

⊙ 0 EVAL 17 RETURN 'SMITH RETURN

The EVAL operator requires a column number to search on. In some cases one will not know the column of the character string, thus it would be desirable to search the entire file starting at the first column. This capability is provided by the operator MOD. MOD followed by a level I operand followed by RETURN starts at the record after the one indicated by the active file marker and searches the active file for the designated character string.

EXAMPLE: Start with the seventh record and search the active file for the character string JONES,A.C. One would press:

⊙ 6 MOD 'JONES,A.C. RETURN

If the character string is found the record number where it first occurs is displayed and the active file marker is set to the record number. If the character string is not found the active file marker is set to zero and "0" is displayed.

3.4.4 DELETING BLOCKS OF RECORDS

One can delete the entire active file with the operator DOWN. This purges the entire active file from the system.

To delete a block of records from the active file one uses the operator DEL. The format of DEL is: DEL M,N RETURN. The first operand is required; it is the number of records to be deleted. N is the number of the first record of the block to be deleted. It may be omitted. If it is, it is assumed to be the value of the active file marker.

EXAMPLE: To delete the tenth through twentieth records from the active file, one would press:

DEL 11,10 RETURN

3.5 LEVEL IV - OPERATING SYSTEM INTERFACE LEVEL

Level IV operators provide an interface between COL and the 360 operating system. They allow a user to submit a batch job for execution, access its output, access any operating system data set, and display the status of any unit in the operating system.

3.5.1 ACCESSING OPERATING SYSTEM DATA SETS

Operating system data sets can be loaded into the active file by the operator LOAD. The data set requested is loaded into the active file behind the present contents of the active file. That is, the operating system data set is concatenated onto the existing records in the active file. If the record length of the operating system data set is greater than the declared record length on level II the system keeps the entire record (up to a maximum of 254 characters); however, the entire record is not displayed unless the record length is adjusted.

To load a data set the user must specify the unit type, volume-serial number, data set name (dsname), and member name if the data set is a partitioned data set. When the volume-serial number is entered COL looks to see if the requested device is mounted on the appropriate drive; if it is then COL requests the data set name. If the volume is not mounted then COL issues a request to the operating system (and hence the Computer Center operator) to mount the requested volume and displays the message "VOLUME IS BEING MOUNTED". Then follows a

delay of perhaps several minutes while the operator physically mounts the requested volume. During this time the console is disabled against RESET, so if a typing error has been made the user must wait while the operator verifies that there is no such volume. Thus it is a good idea to check the volume name on the screen before pushing RETURN. When the volume is available, "DSNAME=" is displayed. If there is no space immediately available or no such volume COL will return the diagonistic "VOLUME CANNOT BE MOUNTED". When this happens and the volume exists the user is asked to wait five minutes and try to load his data set again. If this second try is unsuccessful, then the user should call the operator on duty, and make arrangements to have the volume mounted. When a volume is mounted for a user and he expects to access it several times, it is courteous to the operator to call and tell him, as each access causes his console alarm to ring unless he changes the status of the volume.

EXAMPLE: Loading a 2314 disk data set called EXDRE. The data set resides on volume number D999.

User presses	System responds
LIV LOAD	UNIT =
2314 RETURN	VOLUME =
D999 RETURN	DSNAME =
EXDRE RETURN	(pause) NOW LOADING (pause)
	FILE LOADED

EXAMPLE: Loading a member of a partitioned data set. The data set is called RJEOUT. The name of the member we wish to load is called POLY. The data set resides on a 2314 disk, volume number MVT180.

User presses	System responds
LIV LOAD	UNIT =
2314 RETURN	VOLUME =
MVT180 RETURN	DSNAME =
RJEOUT RETURN	MEMBER =
POLY RETURN	(pause) NOW LOADING (pause)
	FILE LOADED

If a requested data set or member cannot be found, the diagnostic "NOT FOUND ON VOLUME" is displayed. If a number of users are loading data sets simultaneously, the message "WAITING" may be displayed indicating the necessity of waiting a few seconds to use a resource.

Index sequential or direct data sets cannot be loaded, nor can a data set with variable length spanned record format.

A user wishing to load security protected data sets must make arrangements with the Computer Center when he acquires his account number.

3.5.2 REMOTE JOB ENTRY (RJE)

Any job which can be submitted for batch-mode execution at the Computer Center can be executed through RJE. For job

submission, the users active file is considered to be his input stream to the batch regions which execute jobs. That is, the active file must contain the job to be executed. The active file can contain more than one job. Each job must be structured as if it were being submitted on the card reader. For all practical purposes, the active file can be visualized as a deck of cards and the SUB operator as a command to start the user's card reader into the system. Thus when the active file is submitted to the operating system it must be a complete job. This means the job must have a valid JOB card and a valid EXEC card; as well as any source decks, object decks, data cards, and other job control language cards necessary to complete the definition of the job being submitted.

The JOB card supplied by the user is modified by COL before it is submitted to the operating system. This modification applies only to the current job submission, it does not change the record in the active file. For convenience to the operators in returning output to the proper output location the six characters 'RJE---' are inserted into the comment field of the JOB card. This means the maximum number of characters in a comment field is limited to fourteen. RJE does not truncate a comment field that is too long; therefore if the user's comment field exceeds fourteen characters, he will get a JCL error from the operating system.

When the user pushes the SUB operator to submit the job he will see the message "VOLUMES NEEDED". If the user's job requires a private storage device (i.e., a disk or tape) to be

mounted before the job can begin execution, it must be requested here. If no private volume is required, RETURN will complete the job submission. If more than one private volume is required, they can be requested by listing each volume followed by a comma.

Once a user has pushed RETURN, the job is submitted. RESET will not stop the submission of the job. As long as a user has not pushed RETURN he can stop the submission of a job by pressing RESET or any other upper keyboard key.

EXAMPLE: Submitting a job that does not require any private volumes:

User presses	System responds
LIV SUB	VOLUMES NEEDED
RETURN	JOB SUBMITTED

EXAMPLE: Submitting a job that requires a private disk. The volume number of the disk is D999.

User presses	System responds
LIV SUB	VOLUMES NEEDED
D999 RETURN	JOB SUBMITTED

COL checks for a valid job card before submitting the active file to the operating system. If there are no records in the file, the diagnostic "NO RECORDS IN FILE" is displayed. If the first card in the active file is not a JOB card or the job card is invalid, then the diagnostic "INVALID JOB CARD" is displayed

on the display scope and the job is not submitted. If the active file contains only a JOB card and no other records, the diagnostic "JOB HAS NO STEPS" will be displayed on the display scope and the job is not submitted. If all internal readers are in use the message "WAITING--RJE BUSY" is displayed. As soon as one of the readers is available (a matter of seconds) the submission will continue.

3.5.3 DIRECTING RJE AND BATCH OUTPUT TO THE REMOTE DATA SET

If a user desires, he can direct the output of any batch job to a remote data set called RJEOUT. This data set is on a system direct access disk which is always available when the on-line system is available, thus it can be loaded as a COL file. What this means to the OLS user is that he can submit a JOB to the operating system via level IV SUB, then he can load the job's output via level IV LOAD and examine it for meaningful output or errors. This facility is applicable only to system output devices, private data sets (as defined by DD cards) are not affected.

A user directs output to the remote data set by inserting a "T" in the eighth field of a HASP JOB card accounting field. (for information on the accounting field in a JOB card see the COMPUTER CENTER GUIDE.)

EXAMPLE: Accessing the remote data set (RJEOUT). Assume the following JOB card which directs the system output to go to the printer:

```
//JOBNAME JOB (ACCT,USERNAME),'GEOLOGY',MSGLEVEL=1
```

The following JOB card directs the output of the above job to RJEOUT.

```
//JOBNAME JOB (ACCT,USERNAME,,,,,,T),'GEOLOGY',MSGLEVEL=1
```

EXAMPLE: Loading the output of the above job as a COL file.

User presses	System responds
LIV LOAD	UNIT =
2314 RETURN	VOLUME =
MVT180 RETURN	DSNAME =
RJEOUT RETURN	MEMBER NAME =
JOBNAME RETURN	(pause) NOW LOADING (pause)
	FILE LOADED

If the data set or member cannot be found, the system returns the diagnostic "NOT FOUND ON VOLUME".

PRINTING A MEMBER OF THE RJEOUT DATA SET

Any member of the RJEOUT data set may be printed by simply executing a procedure called PRJEOUT. This allows a user to get a hardcopy of his output without having to pay for executing his program again. The procedure is invoked with an EXEC card of the following form:

```
//STEP EXEC PRJEOUT,NAME=jobname
```

In place of the word jobname, the user inserts the name of his

job. The member will be printed and the printed output put in the output box specified on the users JOB card.

3.5.4 DISPLAYING THE STATUS OF SYSTEM DEVICES

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

3.6 DEFINITION OF LI OPERATORS

3.6.1 LI OPERAND FORMS

(S = Storage location, C, N, and L = L0 operands)

FORM	MEANING
'string	The string "string". NOTE: No closing apostrophe.
S	The string stored in storage location S.
S,C	Substring of S starting at the C^{th} character and continuing to the end of the string.
S,C,L	Substring of S starting at the C^{th} character continuing for L characters.
LII N	LII record N.
LII N,C	Substring of record N starting at the C^{th} character and continuing to the end of the record.
LII N,C,L	Substring of record N starting at the C^{th} character and continuing for L characters.

3.6.2 DEFINITION OF LI OPERATORS

(Ø = LI operand, D = Data location, N and M = L0 operands)

⊕ Ø	The operand is concatenated to the end of the active string. The length of the active string is incremented by the length of the operand.
⊖ Ø	The operand is inserted 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 the N^{th} character 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 the N^{th} character and continuing to the end of the string buffer replaces the previous string buffer. The length of the string buffer is decremented by N-1.
SUB ,M or ⊙ ,M	The character string starting at the LI 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 LI 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.
∅	The divided operation deletes a substring from the active string and leaves the remaining parts in the buffer. It has the same operands as SUB.
DEL	strips all trailing blanks from the active string and recalculates the length of the active string.
LS ∅ RETURN RS ∅ RETURN	The active string is searched for the indicated LI operand. LS searches to the left, RS to the right. If it is found, and is unique, the LI search pointer is set to the position where the string was found. If the operand was not found, the LI search pointer is set to zero. When the string occurs in more than one place, a second operand may be specified after pushing a RETURN. The second operand specifies which occurrence of the operand the search pointer is to be set. If there is no such occurrence the search pointer is set to zero.

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

EVAL Ø

The active string is compared to the operand. The comparison proceeds from left to right. When the active string length is not equal to the operand length, the shorter is used to determine how much of the string should be used for the comparison. If the first N characters (where N is the least of the two lengths) are not equal, a plus or minus one is returned when the L0 operation EVAL + is performed. A minus one indicates that the character in the active string occurred earlier in the collating sequence. A plus one indicates the inverse. When the first N characters are equal the longer string is considered to be farther in the collating sequence. Only when the length and characters in a string are equivalent is zero returned by the L0 EVAL + operation.

MOD STRING

The string buffer is searched for "STRING". If it is found, the LI search pointer is set to that value. If it is not found, the LI search pointer is set to zero. If the string is not unique the sequence MOD "STRING" may be followed by the buttons RETURN N RETURN where N, a L0 term, specifies to which occurrence of string the LI search pointer is to be set. If there is no such occurrence the LI search pointer is set to zero and the message "NO SUCH OCCURRENCE" is displayed.

NOTE: LI MOD is identical in operation to LII MOD except LI MOD changes the LI search pointer.

SIN Ø RETURN
Ø RETURN

All occurrences of the first operand are replaced by the second operand. The operation may change the length of the active string. The null string is a valid second operand.

In that case all occurrences of the first operand would be deleted.

EXAMPLE for SIN:

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

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

produces: "XYZCCDEXYZC"

COS Ø RETURN
Ø RETURN

The characters specified by the first operand are translated to 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. After operation active string = "TYXS XS Z MESSZGE".

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

ID

ID causes all presently supported characters to be loaded into the string buffer.

LOAD Ø

The operand Ø is loaded into the string buffer.

LOAD RETURN

The null string is loaded into the string buffer.

DISPLAY Ø

The operand 0 is loaded into the string buffer and displayed.

DISPLAY SPACE Ø

The operand 0 is loaded into the string buffer and displayed without a carriage return preceding the display.

STORE D

The string buffer replaces the previous value of data location D. The string buffer is not altered.

STORE LII N

The string buffer replaces LII record N. LII record N must have been previously defined. The string buffer is not altered.

3.7 DEFINITION OF LII OPERATORS

In all of the following definitions N may be any LO operand.

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 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 push 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 C	(where C is a character string) searches for a unique character string. If no match is found a diagnostic message 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 column.

MOD C RETURN N RETURN — for multiple occurrences of a character string, N specifies that one is searching for the N^{th} occurrence of the specified string.

MOVING CONTENTS BETWEEN ACTIVE AND SAVE BUFFERS

INV — switches the active and save buffer.

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 first character of the destination. 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 N^{th} 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	will load the N^{th} 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...C RETURN	inserts the character string CCC...C 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.
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 N^{th} 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 N^{th} 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, by default N is assumed to be the value of the active file marker.
DOWN N RETURN	deletes the N^{th} record from the active file. If N is omitted, N is assumed to be the value of the active file marker.

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

Ø +	enables automatic skip-duplicate-left-zero option.
Ø -	disables automatic skip-duplicate-left-zero option.
SQ M,L,A RETURN	defines a field L characters long, starting at character M, A defines the operation code for this field: S skip the entire field D duplicate the entire field N right adjusts a numeric field (left zero)
SQRT	clears all tabs and field definitions.
SET TAB or SUM M,N,...RETURN	sets a tab at column M, column N,...
SET TAB or SUM RETURN	sets a tab at the value of the active buffer pointer.
CLEAR TAB or DIFF	clears the tab at the value of the active buffer pointer.
TAB or NEG	skips to the next tab setting.

3.8 DEFINITION OF LIII OPERATORS

⊕	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 N^{th} 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 operand RETURN	EVAL 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. The other operand is any valid LI operand.
MOD operand	searches each record (starting with the active file marker plus one) for the specified string. The operand is any valid LI operand. If the string is found, the file marker is set to the record number of the record containing the string, the LI 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 marker and pointer are set to zero.
SORT N,M RETURN or SQRT N,M RETURN	takes the inactive file, sorts it 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 it defaults to the record length minus N plus 1.

ARG M,N,Ø RETURN	copies M records from the inactive file and inserts them into the active file. The block of records will be inserted before the N^{th} record in the active file. Ø is the record number of the first record in the inactive file to be copied. N and Ø may be omitted, in either case their value defaults to the respective file marker. 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.
DISPLAY M,N RETURN	displays M records, starting with record number N.
DISPLAY M RETURN	displays M records, the first record displayed is indicated by the active file marker.
DISPLAY , N RETURN	displays the remainder of the active file starting with the N^{th} record.
DISPLAY , RETURN	displays the remainder of the active file starting with the record indicated by the active file marker.
DISPLAY RETURN	displays all of the active file.

3.9 DEFINITION OF LIV OPERATORS

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 all the 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.

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 L0 EVAL OPERATORS FOR COL

EVAL RETURN	EVAL followed by a return places the length of the LI string buffer in the L0 quotient register.
EVAL D	places the length of LI storage location D in the L0 quotient register.
EVAL +	The result of the last LI EVAL operation is loaded into the L0 quotient register.
EVAL ,	The current value of the LI search pointer is loaded into the L0 quotient register.
EVAL (	The current value of the active buffer pointer is loaded into the L0 quotient register.
EVAL)	The current value of the save buffer pointer is loaded into the L0 quotient register.
EVAL .	The number of records in the active file is loaded into the L0 quotient register.
EVAL ?	Loads the value of the active file marker into the L0 quotient register.
EVAL LI	Loads the decimal number that the first nine numbers of the string buffer represent. If the length of the string buffer is less than nine, then convert what is there.
EVAL -	Loads the decimal equivalent of the first character in the string buffer. This assumes that the first character is a hexadecimal number.

MATHEMATICALLY ORIENTATED 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

follow:

a) If no sign precedes a number, it is assumed to be positive.

b) If no decimal point is typed, it is assumed to be at the end of the number.

c) If no exponent or power of ten is indicated, the power is assumed to be zero.

d) If any key other than +; -; ·; ,; 0; 1; 2; 3; 4; 5; 6; 7; 8; 9; BACK; or SPACE is pressed, the end of the number is taken to be the last button just preceding that key. BACK deletes the preceding key. SPACE deletes the existing scalar and enables the user to enter a new scalar.

e) All numbers are entered as decimal expressions. Numbers like π or 1/3 must be converted to decimal form. For example, 3.14159 is a suitable approximation of π.

Complex numbers a + ib are entered by typing the real number a, then a comma "," followed by the real number b. Thus $3.14 \times 10^{2} - 1.6 \times 10^{3}i$ would by typed as 3.14+2, -1.6+3. The complex number $-2.65 \times 10^{-3} + 8.31 \times 10^{-2}i$ would be represented by -2.65-3, 8.31-2 or -.265-2, .831-1 or -.00265, .0831. The complex number 871i would be entered by 0,871 or ,871.

MOLSF provides single-precision numbers with up to seven significant decimal digits, ranging from 10^{-75} to 10^{75}, approximately. MOLSF gives sufficiently accurate results for most computations.

4.2 DATA STRUCTURES AND THE WORKING REGISTERS

Whether the user is working with scalars (LI), vectors (LII), or arrays (LIII), setting up problems and carrying them through to solution is greatly simplified if he comprehends the system software for MOLSF operators. There are two work areas for each level, called the α and β working registers, respectively. The working registers are where all computations are executed. A user has to load a number into the appropriate working register or registers before he can begin calculations on it.

Table 3.1 Currently Available MOLSF Levels and Their Data Structures

Level		Data Structure	Name of Working Register	Length
LI	REAL	Single real numbers	β_I	1
LI	CMPLX	Single complex numbers	(α_I, β_I)	1
LII	REAL	Real vectors	β_{II}	$1 \leq n \leq 873$
LII	CMPLX	Complex vectors	$(\alpha_{II}, \beta_{II})$	$1 \leq n \leq 873$
LIII	REAL	Real arrays	β_{III}	$1 \leq n \leq 873$ $1 \leq m \leq 873$
LIII	CMPLX	Complex arrays	$(\alpha_{III}, \beta_{III})$	$1 \leq n \leq 873$ $1 \leq m \leq 873$

The data structure employed in MOLSF is selected by the user. If a user wants to perform calculations on real or complex scalars, then he uses level I. Once a user has signed on MOLSF explicitly or implicitly (as described in sections 2.1.1 and 2.1.6), he accesses level I by pressing the LI key. Once he has done this all operator keys execute operations on the working registers dedicated to scalars.

On level I each register is composed of a single scalar, called α_I and β_I respectively. On LI REAL the β_I working register contains the real scalar being manipulated. On LI CMPLX the α_I working register contains the real part of the complex scalar and the β_I working register contains the imaginary part of the complex scalar.

A user accesses the vector operators by pressing the LII key. Ordered lists of scalars are called vectors in accordance with mathematical terminology. A real vector with N components is a list of N scalars. A complex vector with N components is a list of N complex scalars, or equivalently two lists of N real scalars. The number of components of a vector is called its context. A user specifically declares the length of the vector he wishes to work with by means of the CTX key.

EXAMPLE: Setting the length of a vector to 25 components.

CTX 25 RETURN

CTX followed by an integer or LO operand sets the context of the working register to that value. The number of components of any vector may vary from one to 873. When a user sets up a user

number with the Computer Center he requests a maximum context. This user requested limit cannot be exceeded unless a systems programmer changes the user requested limitations. (This limit is imposed for billing purposes, as cost increases with increased context.) If a user tries to exceed his maximum context, the diagonistic "CONTEXT ERROR" is displayed. A user can display the current working register context by pressing:

DISPLAY CTX RETURN

A user can display the context of a stored vector by pressing:

DISPLAY CTX F RETURN

where F is the storage location for a vector. When a user first signs on the air the context is automatically set to 51. If he wishes to use any other context, he must specifically change the context.

The working registers on level II, called α_{II} and β_{II}, are two ordered lists of scalars. The number of scalars is determined by the context key as described above. The basic MOLSF operators are written so that the α_{II} and β_{II} working registers serve a natural purpose. When functions of a real scalar are constructed on LII REAL the β_{II} working register contains the ordinates or Y values which are typically used in mathematical calculations. In particular, most of the LII REAL operators transform the data in the β_{II} working register. The components of the α_{II} working register are not involved in the computations, but are used as abscissas, or X coordinates, for graphical display.

As a further illustration, consider the function $y = f(x)$. The user first must choose a domain for the independent variable, x, say $u \leq x \leq v$. Once the domain is selected, the number of points for which f can be evaluated in any one computation is determined by the length of the α and β registers. This means that $f(x)$ can be computed at up to 873 points simultaneously. The selection of points between u and v for which f is to be calculated constitutes the discrete domain of x. The user constructs this domain in the β_{II} working register on level II REAL. He may put these values (or other values) into the α_{II} working register. He then computes f. On level II REAL, the α_{II} working register is not changed by any operator except SUB or SORT. All other computations are performed on the β_{II} working register only, and the results of the calculation replace the previous contents of the β_{II} working register.

Operations with vectors are performed component by component. That is, the operation is performed on each component, and the vector whose components are the results of these calculations is the resultant vector. For example, if the α_{II} and β_{II} registers contain

$$\underbrace{(a_1, a_2, \ldots, a_n)}_{\alpha_{II}} \qquad \underbrace{(a_1, a_2, \ldots, a_n)}_{\beta_{II}}$$

and the user pushes the SIN key on level II REAL, the resulting contents of the working registers are

$$\underbrace{(a_1, a_2, \ldots, a_n)}_{\alpha_{II}} \qquad \underbrace{(\sin a_1, \sin a_2, \ldots, \sin a_n)}_{\beta_{II}}$$

The user may now store the entire contents of the β_{II} working register under one of the white alphabetic keys (A-Z, α-ω) on the lower keyboard. For example, STORE Y would put the contents of the β_{II} working register in the storage location Y of level II REAL.

Now suppose that α_{II} and β_{II} were initially as shown above, and one pressed SIN X; then the resulting contents of the working register would be

$$\underbrace{(a_1, a_2, \ldots, a_n)}_{\alpha_{II}} \qquad \underbrace{(\sin x_1, \sin x_2, \ldots, \sin x_n)}_{\beta_{II}}$$

where $X = (x_1, x_2, \ldots, x_n)$ had been previously stored under X.

In this way, the discrete domain of the independent variable is preserved in the α register, and the current stage of f's calculations is in the β register. At any time, the curvilinear display which plots the ordered pairs (a_i, b_i) (a_i the i^{th} component in the α_{II} register and b_i the i^{th} component in the β_{II} working register where i = 1, ..., present CTX) may be obtained on the output device by pressing the keys DISPLAY RETURN.

When complex functions are constructed using LII CMPLX, the mathematical operators operate on the α_{II} and β_{II} working

registers simultaneously, as required by the definition of the complex operations. The α_{II} working register contains the real part of the function; the β_{II} working register contains the imaginary part. The curvilinear display on LII CMPLX is still a plot of the ordered pairs (a_i, b_i) as described above, but the significance and meaning of the graph have changed. The display is now in the complex plane with the imaginary part plotted against the real part.

The array manipulation level is accessed by pressing the LIII key. A series of M vectors each having N components is called an (N,M) array, in accordance with mathematical terminology, A real (N,M) array is an array of N x M real scalars. A complex (N,M) array is an array of N x M complex scalars, or equivalently, two arrays of N x M real scalars each. The size of an array is called its dimension. The dimension of an array is defined by an ordered pair of integers. The first integer defines the number of rows in the array; the second, the number of columns. Thus a five by seven array would have five rows and seven columns.

The CTX key is used to declare the dimension of an array. The maximum square dimension (i.e. N=M) is fixed when a user number is set up by the Computer Center. This maximum may not be exceeded, unless a systems programmer changes the accounting data associated with the user number. A user defines the array dimension he wishes to work with by pressing the CTX key followed by the number of rows, followed by a comma, followed by the number of columns, followed by the RETURN key.

EXAMPLE: Setting the dimension of an array to 5 by 7.

CTX 5,7 RETURN

The maximum dimension possible is a function of (1) the Maximum Square Dimension (MSD) mentioned above and (2) the Maximum Context (MC) mentioned in the previous section. When a user requests a dimension of (N,M) the MSD and MC limits are utilized as follows

1. If N=M
   then $M \leq MSD$

2. If $N \neq M$
   then a. $(M)(N) \leq (MSD)^2$
   and b. $MAX(N,M) \leq MC$

A user can display the current dimension by pressing:

DISPLAY CTX RETURN

A user can display the dimension of a stored array by pressing:

DISPLAY CTX F RETURN

where F is the storage location of an array. When a user first signs on the air the array dimension is set to 11 by 11. If a user wishes to work with any other array size he must specifically change the dimension.

The working registers on level III are called the α_{III} and β_{III} working registers. The level III operators manipulate the working registers in the same manner as the level II operators, except for one difference. Because of the large amount of storage required for an array there is no α_{III} working register when one is working on level III REAL. Otherwise the description of the level II working registers applies here.

When a user wants to perform computations on a level other than the current one, he simply presses the desired level key and computes. He will stay on this level until he explicitly changes to another level. Several points about this operation should be emphasized. First both real and complex modes share the same working registers on each level; thus if the user is working on a real vector and then goes to the complex vector mode, the real vector may be changed by any complex operator. If one wishes to save the real vector it must be stored for later use. STORE is discussed in the following pages. Secondly, if the user is on one level and switches to another level his working registers on the old level are not changed, unless he does so explicitly. There are operator keys for interlevel transfer of data (LOAD, STORE, SUB, and EVAL). These are discussed in the section on operator definitions.

4.3 MATHEMATICAL OPERANDS

4.3.1 OPERAND FORMS

The flexibility of MOLSF allows for many operand forms. Roughly they can be divided into two groups, numerical operands and alphabetic operands, which can be used in three ways. The numerical operands are entered by pressing the lower keyboard integers. The means for entering real numbers, complex numbers, and exponents are detailed in the section on number representation.

Alphabetic operands are of the form:

[level] [alphabetic key] [(component)]

This form allows a user to access data stored on other levels without having to change levels and transfer data. If a user wants to access data stored on the level he is working on, then he need only press the alphabetic key which defines the storage location where the data is stored. The level and component entries (which are optional) enable a user to access data on a different level. Examples of the use of the standard operand form are given in section 4.4 (loading of data).

NOTE: In many uses one may omit the component entry when accessing data stored on a higher level. If this is done, MOLSF always assumes the missing indicies are one.

4.3.2 JUXTAPOSITION OPERANDS

Often in mathematical computations the same binary operator is used repetitively, such as in the expression:

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 ⊕ E

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 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 parenthesis:

LI REAL SIN A SQ ⊕ (COS A SQ)

4.4 LOADING OF DATA

The primary function of LOAD is to explicitly enter numbers or copy data from storage locations (A through Z, α through ω, levels I, II, or III) into the working registers for that level.

4.4.1 LOAD FOLLOWED BY A NUMBER (a numeric 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 $(\alpha_{II}, \beta_{II})$ registers.

3) LIII REAL LOAD 2 RETURN enters 2 into every component of the β_{III} register.

The LOAD key is a provision for explicit data transfer. 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 is implicit.

4.4.2 LOAD FOLLOWED BY AN ALPHABETIC OPERAND

When the LOAD key is followed by a letter of the alphabet, the data in that storage location for the current level is copied into the working register.

EXAMPLES:

1) LI CMPLX LOAD Z - The contents of Z on level I CMPLX are loaded into the (α_I, β_I) registers by the LOAD instruction.

2) LII REAL LOAD G - The contents of LII REAL G are placed into β_{II} by the LOAD instruction.

LOADING OF HIGHER LEVEL DATA INTO LOWER LEVEL REGISTER

The LOAD button also permits the transfer of a component from higher level data to the working register of a lower level.

EXAMPLE:

1) LI REAL LOAD LII A(3) - takes the 3rd component of the LII REAL vector A and loads it into β_I.

LOADING WHEN COMPONENT AND ENTRY SPECIFICATIONS ARE INTEGER VARIABLES

The component specification in the above example is an integer but it also may be a L0 operand.

EXAMPLE:

1) LI REAL LOAD LII A(P) - uses the integer stored in Level 0 P to find the P^{th} component of the LII REAL vector A and loads the component into β_I.

MODIFICATION OF VARIABLE SPECIFICATION BY INCREMENTING LOAD INSTRUCTION

The variable specification can be modified by providing an increment each time the LOAD instruction is executed.

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.

NOTE: The preceding two examples have no instruction sequence which would lead the reader to believe that the examples will be executed more than once. Their explanations, however, are based on the assumption that they are embedded in a program which is repeated a number of times in the course of a problem solution.

GENERAL LOAD FORMAT

The general format for the keys which follow LOAD is:

LOAD [level] [location] [(component)]

The "level" designation is L0, LI, LII, LIII, or omitted; "location" ranges over the Latin and Greek alphabets, A through Z and α through ω, or may be a number; "(component)" may be "(i)", "(i+j)" or omitted, i and j being integers or level 0 operands.

EXAMPLE:

LII REAL LOAD LI A - places the scalar stored in LI REAL A into every component of β_{II}.

In order to load level III data into level I and II registers it is necessary to specify which component(s) to load.

In the case of loading into level I both a row and a column must be specified. The form is:

LI REAL LOAD LIII A(3,2) RETURN

To load into level II either a row or a column may be indicated.

LII REAL LOAD LIII A(3,) RETURN

or

LII REAL LOAD LIII A(3) RETURN

for loading a row

and

LII REAL LOAD LIII A(,2) RETURN

for loading a column.

4.5 STORING OF DATA

The instruction STORE is the counterpart 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 location are replaced by the quantity which is stored. The level specification most recently preceding the data designation key will be the one used to determine which storage location is desired. On level 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:

STORE [level] [location] [(component)]

1) "level" is LI, LII, LIII, or omitted.
2) "location" is one of the letters A through Z, α through ω.
3) "(component)" is (i), or (i+j), or omitted, with i and j integers or Level 0 variables.

EXAMPLES:

1) LI REAL LOAD 3.2 STORE A - stores the single real number 3.2 in LI REAL A.
2) LII REAL ID STORE X - stores the uniformly-spaced discrete domain of the interval $-1 \leq x \leq 1$ into X.
3) LI REAL LOAD B $\odot$ 5 $\oplus$ 3 STORE D - stores the single real number 5B + 3 into LI REAL D.
4) LI REAL LOAD I STORE II Q(J) - stores the contents of LI REAL I in the J^{th} component of the REAL vector A. J is an L0 operand.

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

6) LII REAL LOAD 1 STORE S - stores 1 into every component of the vector S.

7) LII LOAD A STORE LI B - stores the first component of level II A into Level I B.

8) LIII LOAD A STORE LI B - stores the first (1,1) element of level III A into Level I B.

9) LIII LOAD A STORE LII C - stores the first column of Level III A into Level II C.

NOTE: When executing LOAD and STORE operations, the absence of a "location" specification implies loading from/storing into the working register specified by "level" "(component)". For example, LI REAL STORE LII (K) stores the contents of β_I into the K^{th} component of β_{II}.

(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.)

The execution of STORE does not affect the contents of the working register. For example, the contents of β_I after the end of the sequence of instructions in Example 1 above will still be 3.2, even though STORE A is executed.

4.6 DISPLAY FACILITIES

4.6.1 DISPLAY KEY

The DISPLAY instruction generates numerical or graphical display on the output device(s). There are five different forms of the display instruction, as follows:

1) DISPLAY RETURN - displays the contents of the α and β registers as appropriate to level I or II. If the user is on LI REAL the decimal numerical value in β_I is written on the scope. On LI CMPLX both α_I and β_I are displayed numerically. On level II REAL and CMPLX the (α_{II}, β_{II}) registers are cross-plotted and displayed graphically. On level III REAL, β_{III} is plotted as a surface over a fixed x-y grid. Scaling is discussed in Section 4.6.3.

   On level II the graphical display consists of the points whose x and y coordinates are the components of the vectors in the α and β registers, with each successive pair of points a_i, a_{i+1} connected by a straight line segment. On level III the surface display consists of plotting each column of β_{III} as a level II vector (with hidden lines removed).

2) DISPLAY . RETURN - point display same as above on level II, and III without connecting lines.

3) DISPLAY .. RETURN - same as (2), except that large dots or special characters (see section 4.6.4) are used in the display.

4) DISPLAY k RETURN (where k is an integer). On level II the k^{th} component of the vector in β_{II} or (α_{II}, β_{II}) is displayed.

5) DISPLAY i,j RETURN (where i and j are integers). On level III the (i,j) element of β_{III} or (α_{III}, β_{III}) is displayed.

6) DISPLAY A, DISPLAY . A, DISPLAY .. A - automatically loads A into the appropriate register for the current level and then generates a display as specified above.

In those cases above which generate numerical displays, SPACE may be inserted immediately following DISPLAY, and will

cause the usual carriage return preceding the display to be suppressed. This provision facilitates labeling of displays on the TYPE level. For example, the display "X=3" can be generated by the instructions:

TYPE RETURN X =

LI LOAD 3 DISPLAY SPACE RETURN

A curvilinear display for complex functions assumes the real axis (α register) to be horizontal, and the imaginary axis (β register) to be vertical. In numerical display of complex values, the left most number is the real part, the right most number is the imaginary part.

4.6.2 NUMERICAL DISPLAY

On the REAL and COMPLEX levels a number is displayed, by default, with one digit before the decimal point, one to six digits after the decimal point, and the appropriate decimal scale (power of 10). Insignificant zeros are suppressed.

EXAMPLES:

```
     .00301 is displayed as 3.01  -03
    3.01    is displayed as 3.01  +00
30100       is displayed as 3.01  +04
   30       is displayed as 3.    +01
```

This display format can be changed, as will be discussed in Section 4.6.4.

On LI, DISPLAY A causes the present value in the appropriate storage location A (REAL, or COMPLEX) to be loaded into the

appropriate working register [β_I or (α_I,β_I)] and printed on the output device on the line immediately below the last printed information. The sequence DISPLAY RETURN causes the value already in the register for the current level to be printed.

On level II one must load the vector into the working register and specify the component to obtain a numerical display. Thus

LOAD A DISPLAY 6 RETURN

results in a numerical display of the sixth component of A, REAL or COMPLEX, just as for level I.

To display the "j"th component of a level II vector, where "j" is the value of a level 0 variable J, one must go to level I:

LI REAL LOAD LII A(J) DISPLAY RETURN

It frequently happens on level II that one wishes to display a sequence of components of a vector. This can be accomplished in a straightforward manner. On level II, after the first numerical display has occurred, each subsequent depression of the RETURN key displays the next value. Each depression of BACK displays the previous value. Pressing the "?" key on the lower keyboard displays the number of the component just displayed.

Similarly for level III the user must load the array into the working register and specify the component to obtain a numerical display. Thus

LOAD A DISPLAY 3,5 RETURN

results in a numerical display of the 5^{th} component in the 3^{rd} row of A, REAL or COMPLEX.

To display the "i,j"th component of a level III array, where "i" and "j" are the values of level 0 variables I and J, one must go to level I:

LI REAL LOAD LIII A(I,J) DISPLAY RETURN

Often on level III one wishes to display a series of components of an array. This is accomplished easily by pressing:

RETURN - displays the (i+1,j) component

BACK - displays the (i,j-1) component

SPACE - displays the (i,j+1) component

| (on the lower keyboard, the "or" symbol) - displays (i-1,j) component.

Pressing "?" on the lower keyboard will identify the last element displayed.

4.6.3 GRAPHICAL DISPLAY

One of the most important aspects of MOLSF is its curvilinear display capability which allows the user to obtain a graphical representation of the results of his computations. One problem associated with plotting of level II vectors on an output device(s) is proper scaling. Unless otherwise instructed, the basic display program computes the best scale for viewing a single functional display. But if two or more level II vectors are being displayed for comparative purposes and their scale factors

do not match, then conclusions about intersections, distance between points, etc., may be erroneous. To aid in these comparisons a set of vectors can be displayed on a common scale. Means for establishing this scale are discussed below.

REPRESENTATION OF SCALE FACTORS FOR LEVEL II

For display purposes, MOLSF uses a floating vector organization in which a vector with n components is represented by n numbers (the mantissas), with magnitudes less than or equal to 1, together with a single binary scale. Thus the vector A, having mantissas a and the binary scale S, is represented in the form

$$A = a \cdot 2^S$$

When a curve is displayed on the scope only the mantissas are used to form the display points. The binary (i.e., base 2) exponent S is called the display scale of the curve being displayed. If a curve has scale S, then the limits of the display area on the scope are $\pm 2^S$. The value S for the function currently in the β_{II} register (denoted S_β) can be determined, on level II REAL, by

DISPLAY 0 RETURN

On level II CMPLX, this sequence of keypushes displays the binary scales (S_α, S_β) of the α_{II} and β_{II} registers. In effect, the display scale may be thought of as the 0^{th} component of the vector.

If no scaling operations are performed, vectors are displayed in normal form:

1) On level II REAL the α_{II} and β_{II} registers are normalized separately, with the display scale S chosen so that at least one mantissa is greater than 1/2 in absolute value; i.e., the display is as large as possible.

2) On level II CMPLX the α_{II} and β_{II} registers are scaled to a common display scale S, chosen so that the absolute value of at least one ordinate or one abscissa is greater than 1/2.

DETAILED SCALING ALGORITHMS

A property of each register (α_{II} and β_{II}) is its display scale (denoted S_α and S_β, respectively). S_α and S_β are defined by the constraint that the distance between left and right, and upper and lower edges of the scope face, be $2^{S\alpha+1}$ and $2^{S\beta+1}$, respectively.

The display scale is selected by OLS software on the basis of two additional register properties:

1. Data Scale (denoted D_α and D_β)
   D is defined to be the smallest integer greater than or equal to the log (base 2) of the register component largest in absolute value.

2. Relative Scale (denoted R_α and R_β)
   R is independent of the contents of the register and is set or modified by various operators.

Given these properties the display routines compute S_α and S_β as follows:

1. LII REAL - S is the algebraic sum of D and R:

$$S_\alpha = D_\alpha + R_\alpha$$

$$S_\beta = D_\beta + R_\beta$$

2. LII COMPLEX - S is the sum of R and the maximum of D_α and D_β:

$$S_\alpha = R_\alpha + \max \{D_\alpha, D_\beta\}$$

$$S_\beta = R_\beta + \max \{D_\alpha, D_\beta\}$$

The sequences LII REAL DISPLAY 0 RETURN and LO EVAL 0 DISPLAY RETURN display S_β as computed in (1) above. The sequence LII CMPLX DISPLAY 0 RETURN displays S_α and S_β as computed in (2) above.

S_α and S_β can be explicitly changed by the user as follows:

1. LO REAL LOAD N SUB 0 RETURN sets R_β such that S_β will be equal to the current contents of Level 0 N when computed according to (1) above.

2. LO CMPLX LOAD N SUB 0 RETURN sets R_α and R_β such that both S_α and S_β will be equal to the current contents of Level 0 N when computed according to (2) above.

3. LII REAL ENL (or CON) N RETURN will decrement (or increment) R_β by the contents of Level 0 N.

4. LII CMPLX ENL (or CON) M,N RETURN will decrement (or increment) R_α and R_β by the contents of Level 0 M and N, respectively.

In addition, LII REAL and COMPLEX mathematical operators (LII CMPLX REFL, LS, RS, LII REAL, LS, RS, MOD, REFL, NEG are exceptions) set R_α, and R_α and R_β to zero, respectively.

A value of R_β is associated with each level II REAL storage location. This value of R_β is that of β_{II} when the vector was last stored. When the vector is reloaded, this value becomes β_{II}'s R_β. Similarly, a value for R_α and a value for R_β are associated with each level II COMPLEX storage location. These values are those of α_{II} and β_{II} when the vector was last stored.

When the vector is reloaded, these values become α_{II}'s R_β and β_{II}'s R_β, respectively.

TECHNIQUES

Because the (α_{II}, β_{II}) registers are used in complex arithmetic, some caution must be observed when forming level II REAL displays after performing complex arithmetic. The α_{II} coordinates will often need to be restored by proper use of ID X or SUB as explained in section 4.8.3.

At times it may be desired to place coordinate axes on the display scope to use as reference axes. Rectangular axes can be plotted on level II REAL by using the operation ID, which places the vector consisting of n equally spaced values in the range [-1,1] in the α_{II} and β_{II} registers. To obtain an X axis the key sequence LII REAL ID LOAD 0 DISPLAY RETURN is pressed. The Y axis is generated by the instruction set LII REAL ID SUB 0 DISPLAY RETURN or, immediately after displaying the X axis, by rotating it 90 degrees by CMPLX REFL DISPLAY RETURN. Note that these axes can be generated and then stored as complex vectors, for general plotting purposes, by LII REAL ID LOAD 0 CMPLX STORE X REFL STORE Y. Then, whenever axes are desired, the user simply pushes LII CMPLX DISPLAY XY or, for dotted axes, LII CMPLX DISPLAY . XY.

LEVEL II REAL SCALING OPERATORS

When two or more level II vectors are displayed on the scope for comparison, their display scales must be taken into consideration.

For example, the functions e^x and sin (10x), for $-1 \leq x \leq 1$, are computed and displayed (Figure 6.3.1) by

LII REAL ID EXP DISPLAY RETURN
ID ⊙ 10 SIN DISPLAY RETURN

With no scaling information, it is impossible to determine relative magnitudes, intersections, etc. In Figure 6.3.2 the display scales of the two curves are displayed by

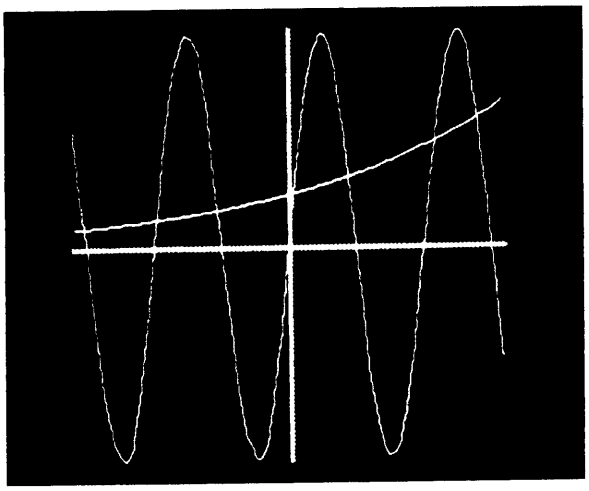
LII REAL ID EXP DISPLAY RETURN DISPLAY 0 RETURN ID
⊙ 10 SIN DISPLAY RETURN DISPLAY 0 RETURN

Now it is apparent that the exponential curve has a display scale of 2 and the sine curve has a display scale of 0.

With the display scales known, in manual operation, the contract operator may be used twice to display the sine curve with a display scale of 2, for valid comparison with the exponential curve (Figure 6.3.3):

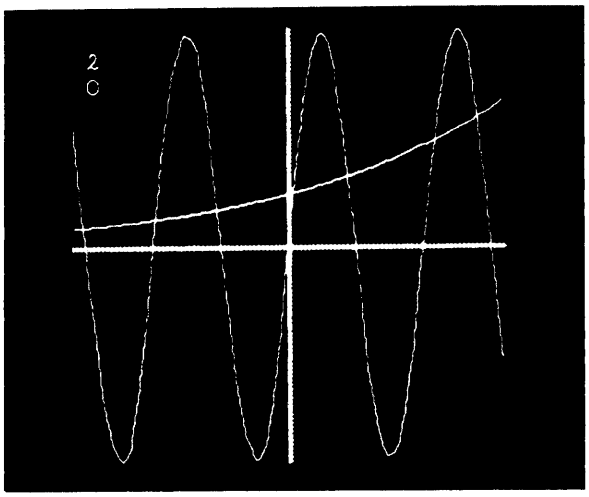
LII REAL ID EXP DISPLAY RETURN ID
⊙ 10 SIN CON CON DISPLAY RETURN

Previously stored vectors may be displayed with a common scale automatically. Common scale display is initiated with the sequence DISPLAY , followed by a list of the vectors to be displayed and is closed with the RETURN key. When RETURN is pushed, the maximum scale is found and all curves are displayed on that scale. On complex, the maximum scale for both α and β



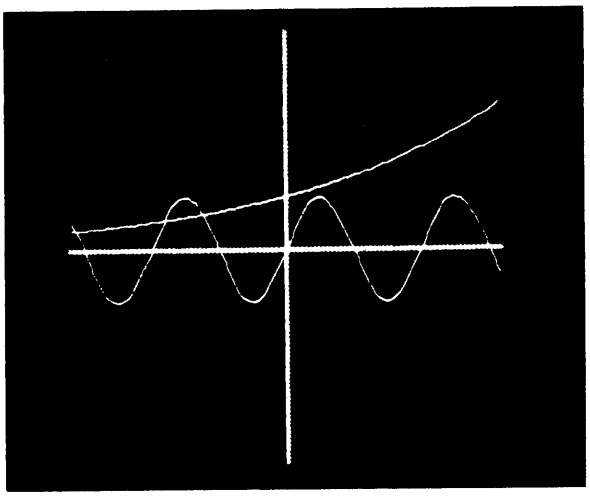
6.3.1

No scaling



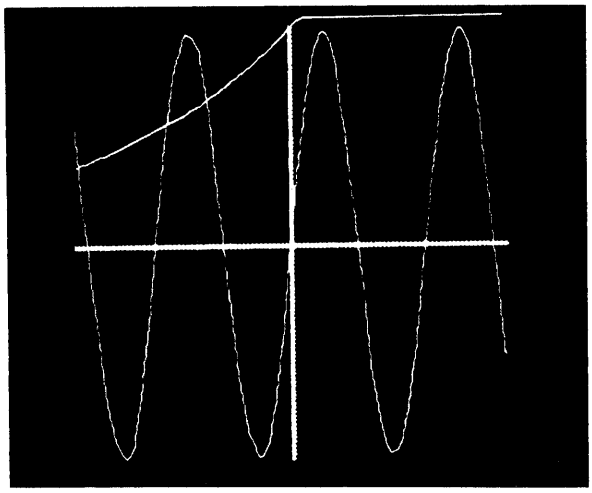
6.3.2

View display scale



6.3.3

Proper scaling



6.3.4

Improper scaling

Scaling Examples: $y = e^x$

$y = \sin 10x$

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 would be displayed normally, B and C in dot mode, and D in dot-dot mode. The scale used for the display would be 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 series of curves.

EXAMPLE:

LII DISPLAY ,2A RETURN

The vector A is displayed with a scale of two.

LEVEL III DISPLAY

Level III Display is similar to Level II Display. With very few conceptual changes the same button 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 FORMATING

Numerical and curvilinear dot-dot displays may be formated. There are three types of formats: Integer display format, floating-point display format, and the character used when a dot-dot display is made.

INTEGER DISPLAY FORMAT ITEM (L0 data, LII and LIII contexts, display scales) - ($n \leq 25$)

In	Left justified - leading zeros suppressed
Ln	Right justified - leading zeros not suppressed
Sn	Right justified - leading zeros suppressed

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 - ($n + m \leq 25$)

Dn.m	Float - suppress trailing zeros
En.m	Float - leave trailing zeros
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 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,*)

4.7 MATHEMATICAL OPERATORS FOR LEVEL I

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

1) "A" represents an alphabetic operand.

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

3) "j", "j_1", and "j_2" are non-negative integers or L0 operands.

4.7.1 OPERATOR DEFINITIONS FOR LEVEL I REAL

The level I REAL operators are as follows:

⊕ , ⊖ , ⊙ , ∅	followed by "A" or "r" computes the indicated combination with the number in the β_I register and leave the result in the β_I register. If these operators are followed by any <u>other</u> operator, they have no effect.
<u>SUB</u>"j"	puts the contents of the β_I register into the j^{th} component of the level II working register, β_{II}, where "j" is a positive integer ($1 \leq$ "j" ≤ 873) or a level 0 operand.
<u>EVAL</u> "j"	puts the j^{th} component of the level II working register β_{II} into the β_I register, where "j" is a positive integer ($1 \leq$ "j" ≤ 873) or a level 0 operand.

(In the following, if the operator key is followed immediately by "A" or "r", the operand is the value in the level I REAL storage location A, 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	perform the indicated operation on the operand. LOG acts on the absolute value of the operand. LOG of zero gives -183.8464.
ARG	0 if operand $\geq$ 0; π if operand < 0.
DEL	0 if operand $\neq$ 0; 1 if operand = 0.

4.7.2 OPERATOR DEFINITIONS FOR LEVEL I COMPLEX

On level I CMPLX MOLSF operates on complex scalars. The operators available for this purpose are listed below:

$\oplus$, $\ominus$, $\odot$, $\oslash$	followed by "A" or "r_1, r_2" computes the indicated complex combination with the complex number in the (α_I, β_I) register.
SUB "j"	puts the contents of the (α_I, β_I) register into the j^{th} component of the level II working register (α_{II}, β_{II}), where "j" is a positive integer ($1 \leq$ "j" ≤ 873) or a level 0 operand.
EVAL "j"	puts the j^{th} component of the level II working register (α_{II}, β_{II}) into the (α_I, β_I) register, where "j" is a positive integer ($1 \leq$ "j" ≤ 873) or a level 0 operand.

(In the following, if the operator key is followed immediately by "A" or "r_1, r_2", the operand is the value in the level I COMPLEX storage location A, or the complex number $r_1 + ir_2$, respectively. If the next key pushed after the operator key is not "A" 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	perform the indicated operation on the operand; LOG takes the branch provided by ARG.
ARG or ARG -	computes the argument in the interval $[-\pi, \pi]$ of the operand. The argument of 0 + 0i is defined to be 0.
ARG +	computes the argument in the interval $[0, 2\pi]$ of the complex number in the (α_I, β_I) register. The argument of 0 + 0i is defined to be zero.
DEL	0 if operand $\neq$ 0 + 0i; 1 if operand = 0 + 0i.

4.7.3 ADDITIONAL COMMENTS ON LEVEL I

Data in the working register can also be transferred between level I REAL and level I CMPLX by simply changing levels. 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. Calculates the principal value of log (3 + 2i) and prints it.

4) LI CMPLX MOD Z DISPLAY RETURN. Computes the modulus of the complex number stored in Z and displays it 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 I REAL and I 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

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 "A", which represents an alphabetic key, now implies a vector $A = (a_1, a_2, \ldots, a_n)$ and "r", which represents a real number, now defines a constant vector of n components.

$\oplus$, $\ominus$, $\odot$, $\oslash$ followed by "A" or "r" perform the indicated operations componentwise using the vector in the β_{II} register and the real vector in "A" 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:

$\oplus$ A: $\beta_{II} = (\beta_1 + a_1, \beta_2 + a_2, \ldots, \beta_n + a_n)$

$\ominus$ A: $\beta_{II} = (\beta_1 - a_1, \beta_2 - a_2, \ldots, \beta_n - a_n)$

$\odot$ A: $\beta_{II} = (\beta_1 a_1, \beta_2 a_2, \ldots, \beta_n a_n)$

$\oslash$ A: $\beta_{II} = (\beta_1/a_1, \beta_2/a_2, \ldots, \beta_n/a_n)$

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

LS shifts each component β_k of the β_{II} register into the (k-1)st position 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 "j" is equivalent to repeating LS j times.

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

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

RS "j" is equivalent to repeating RS j 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 "j" repeats ENL j 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 "j" repeats CON j 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 "A" replaces f(X) with f("A"). The process is as follows: for each component a_i of "A" 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 $f(a_i)$ as

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

If a_i is greater than (or less than) all of the α_{II} components, the value of $f(a_i)$ is set equal to the β_{II} correspondent of the maximum (minimum) β_{II} component. If the dimension of "A" is not equal to the dimension of the β_{II} register, the α and β registers

containing the result, "X" and f("A"), will have the dimension of "A". If followed by "r", EVAL creates a constant vector proceeding as above.

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

EVAL - — Similar to EVAL except that $f(a_i)$ 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.

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

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.

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

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.

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

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 "A", SUB loads contents of "A" into α_{II} working register. When followed by "r", SUB loads "r" into the α_{II} working register.

[In the following, if the operator key is followed immediately by "A" or "r", the operand is the vector in the level II REAL storage location A, or the constant vector r, respectively. If the next key is not one of these, the operand is the real vector in β_{II}. We will call the operand $(a_1, a_2, \ldots, a_n)$ A. The result is always put into the β_{II} register.]

SQ — squares each component of the operand.

$$\beta_{II} = (a_1^2, a_2^2, \ldots, a_n^2)$$

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

$$\beta_{II} = (\sqrt{a_1}, \sqrt{a_2}, \ldots, \sqrt{a_n})$$

NEG — negates each component of the operand.

$$\beta_{II} = (-a_1, -a_2, \ldots, -a_n)$$

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

$$\beta_{II} = (1/a_1, 1/a_2, \ldots, 1/a_n)$$

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

$$\beta_{II} = (a_2-a_1, a_3-a_2, \ldots, a_n-a_{n-1}, 2a_n-3a_{n-1}+a_{n-2})$$

SUM — forms the running summation of the components of the operand.

$$\beta_{II} = (a_1, a_1+a_2, a_1+a_2+a_3 \ldots, \sum_{k=1}^{n} a_k)$$

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

$$\beta_{II} = (a_1,\ a_1a_2,\ a_1a_2a_3,\ \ldots,\ \prod_{k=1}^{n} a_k)$$

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

$$\beta_{II} = (a_n,\ a_{n-1},\ \ldots,\ a_1)$$

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

$$\beta_{II} = (|a_1|,\ |a_2|,\ \ldots,\ |a_n|)$$

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

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

SIN, COS, LOG, EXP, ATAN — perform the indicated operation componentwise on the operand vector.

SIN: $\beta_{II} = (\sin a_1,\ \sin a_2,\ \ldots,\ \sin a_n)$

COS: $\beta_{II} = (\cos a_1,\ \cos a_2,\ \ldots,\ \cos a_n)$

} (a's in radians)

LOG: $\beta_{II} = (\ln a_1,\ \ln a_2,\ \ldots\ \ln a_n)$

EXP: $\beta_{II} = (e^{a_1},\ e^{a_2},\ \ldots,\ e^{a_n})$

ATAN: $\beta_{II} = (\tan^{-1}a_1,\ \tan^{-1}a_2,\ \ldots,\ \tan^{-1}a_n)$

} (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 the 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. So that

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

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

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 $a_k = 0$, $\beta_k = 1$

If $(a_k)(a_{k+1}) < 0$, $\beta_k = 1$ if $|a_k| < |a_k + 1|$

$\beta_{k+1} = 1$ if $|a_{k+1}| < |a_k|$

All other $\beta_k = 0$

CONV

provide a means for obtaining a discrete approximation to the integrals.

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

$\int_{o}^{t} K(t - \tau)\ F(\tau)\ d\tau$

and

$\int_{o}^{2\pi} K(t - \tau)\ F(\tau)\ d\tau$ K period = 2

by pressing: LII REAL LOAD F CONV K,j.

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 integral 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 $K \cdot \Delta\tau$ 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 a positive integer 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

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

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

To use the convolution operator one must first select the kernal as a vector with a context less than or equal to F and then reflect it by LII REAL REFL. The convolution operator does not reflect K. Given the reordered kernal, geometrically, one may picture a fixed graph for the function F and a graph for K being translated to the right one position. K is then multiplied with a dot product with the graph of F. This operator is then repeated a number of times equal to the context of F.

4.8.2 OPERATOR DEFINITIONS FOR LEVEL II COMPLEX

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

⊕ , ⊖ , ⊙ , ⊘ followed by "a" or "r_1, r_2" perform the indicated complex combination of the operand vector with the complex vector in the $(\alpha_{II}, \beta_{II})$ 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.

LS shifts each component (α_k, β_k) of the $(\alpha_{II}, \beta_{II})$ registers into the (k-1)st position, placing the first component into the last position.

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

LS "j" left shifts the α_{II} register j times. LS "j_1, j_2" shifts the α_{II} and β_{II} registers separately, the α_{II} register j_1 times and the β_{II} register j_2 times. LS , j left shifts the β_{II} register j times.

RS

shifts each component (α_k, β_k) of the $(\alpha_{II}, \beta_{II})$ registers into the (k+1)st 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 "j" right shifts the α_{II} register j times. RS "j_1, j_2" shifts the α_{II} and β_{II} registers separately, the α_{II} register j_1 times and the β_{II} register j_2 times. RS , j right shifts the β_{II} register j times.

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 "j" enlarges the α_{II} register j times. ENL "j_1, j_2" enlarges the α_{II} and β_{II} registers separately, the α_{II} register j_1 times and the β_{II} register j_2 times. ENL , j enlarges the β_{II} register j times.

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 "j" contracts the β_{II} register j times. CON "j_1, j_2" contracts the α_{II} and β_{II} registers separately, the α_{II} register j_1 times and the β_{II} register j_2 times. CON , j contracts the β_{II} register j times.

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 $(\alpha_{II}, \beta_{II})$ registers.

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

[In the following, if the operator key is followed immediately by "A" or "r_1, r_2", the operand is the vector in the level II CMPLX storage location A, or the constant complex vector $r_1 + ir_2$, 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 $(\alpha_{II}, \beta_{II})$ registers.]

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 subtotals in the corresponding components of the result. The operator is performed in accordance with the rule for complex multiplication.
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	perform 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 +.
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 in the $(\alpha_{II}, \beta_{II})$ register, assuming that the argument of the first point lies in the interval $(-\pi, \pi)$. 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

identically to LOAD except that the vector is loaded into the α_{II} register instead of the β_{II} register. Therefore, any set of X coordinates desired may be computed and substituted into α_{II} by use of the SUB key. On LII REAL for example, the button sequence

ID ⊙ 6.28 SIN STORE S . . . . . SUB S

would generate sin X, where X is a set of values equally spaced in $(-2\pi, 2\pi)$, and substitute it into the α_{II} register. (If there is no need to save the vector SIN x for other computations, or if the α_{II} register will not be disturbed before the level II REAL display occurs then another method of accomplishing the above is:

LII REAL . . . ID ⊙ 6.28 SIN COMPLX REFL REAL . . .).

If one wishes to insert the ID vector into the α_{II} register for display abscissas with ordinates already computed and currently in the β_{II} register, ID X on II REAL will do so without altering the β_{II} register.

The general definition of curvilinear display is that corresponding components of α_{II} and β_{II} are cross-plotted (i.e., β_{II} versus α_{II}) and displayed with a scale computed by the display program. On LII REAL this scaling does not affect α_{II} but results in the binary scale presenting the largest possible display of the β_{II} coordinates. On LII CMPLX, unless the user specified otherwise, the binary scales of the $(\alpha_{II}, \beta_{II})$ register are taken to be common (i.e., the vertical and horizontal scales are the same) such that the resulting display is of maximum size.

VARYING CONTEXT

Varying the context is helpful when the user wants to replace the first k components of a vector of dimension n ($k \leq n$). The procedure is as follows: Suppose a vector of dimension n, $(a_1, a_2, \ldots, a_n)$, is contained in the level II working register and the user wishes to change the first k components. He changes the working register length to k by pressing CTX K, constructs the k-length vector $(f(a_1), f(a_2), \ldots, f(a_k))$, and restores the context to its original value by pressing CTX N. The working register now contains $(f(a_1), \ldots, f(a_k), a_{k+1}, \ldots, a_n)$. It is particularly important to note that reducing the context does not alter the physical length of the working register, but rather redefines the number of components to be involved in computations. Reducing the context causes the residual data to be unaltered in all subsequent operations until the context is again increased. This philosophy has also been extended to level III, thereby facilitating the manipulation of sub-arrays contained in larger level III arrays.

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 CTX 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

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 the "A", which represents an alphabetic key, now implies an array

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

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

$\oplus$, $\ominus$, $\odot$, $\oslash$ followed by "A" or "r" perform the indicated operations component by component using the array in the β_{III} register and the array "A" or "r". Let

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

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

$\oplus$ A :

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

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

ATAN, LOG, EXP, SIN, COS perform the indicated operation component by component on the operand array.

SQ squares each component of the specified array.

SQRT takes the square root of each component of the specified array.

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

REFL C reverses the order of the components in each of the columns of the β_{III} accumulator.

REFL R — reverses the order of the components in each of the rows of the β_{III} accumulator.

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

NEG — negates each component of the operand.

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

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

The following operators work strictly on columns just as if they were level II variables.

LS — shifts each column β_k of the β_{III} register into the (k-1)st column of the β_{III} register, placing the 1st column in the last position.

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

LS "j" repeats LS j times.

RS — shifts each column $\beta_{(k,)}$ of the β_{III} register into the (k+1)th column, placing the last column into the initial column.

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

RS "j" repeats RS j times.

UP

shifts each element of each column $\beta_{(i,j)}$ into the $[\beta_{(i-1,j)}]$th position, placing the 1st component of each column in the last position.

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

UP "j" repeats UP j times.

DOWN

shifts each element of each column $\beta_{(i,j)}$ into the $[\beta_{(i+1,j)}]$th position, placing the last component of each column in the initial position.

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

DOWN "j" repeats DOWN j times.

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

MAX

sets each column equal to the maximum component in each column.

DEL

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

if $a_{(i,k)} = 0$ then $\beta_{(i,k)} = 1$

if $[a_{(i,k)}][a_{(i,k+1)}] < 0$ then

$\beta_{(i,k)} = 1$ if $|a_{(i,k)}| < |a_{(i,k+1)}|$

$\beta_{(i,k+1)} = 1$ if $|a_{(1,k+1)}| < |a_{(i,k)}|$

all other $\beta_{(i,k)} = 0$

$i = 1, \ldots, n$

$k = 1, \ldots, m$

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.
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 ID R places the vectors in rows rather than columns.

4.9.2 OPERATOR DEFINITIONS FOR LEVEL III COMPLEX

The operands for the level III CMPLX operators are previously defined complex arrays "A", complex constant arrays "R" [representing $(r_i + ir_2)_{j,k}$ where j is the row number and k is the column number], or the contents of the $(\alpha_{III}, \beta_{III})$ registers. The results of the operations are always complex arrays put into the $(\alpha_{III}, \beta_{III})$ registers.

$\oplus$, $\ominus$, $\odot$, $\oslash$	followed by "a" or "R" perform the indicated complex computation component by component on the operand array with the complex array in the $(\alpha_{III}, \beta_{III})$ registers. 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.

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 +.

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.

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 in the $(\alpha_{III}, \beta_{III})$ register, assuming that the argument of the 1st point lies in the interval $(-\pi, \pi)$. The following values are true arguments based on that branch cut.

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

The following operators are column operations as opposed to the previously listed component operations.

LS — shifts each column $[\alpha_{(1,)}, \beta_{(1,)}]$ of the $(\alpha_{III}, \beta_{III})$ registers into the (i-1)th position, placing the 1st column into the last position.

LS "j" repeats LS j times. LS "j_1, j_2" shifts the α_{III} and β_{III} registers separately, the α_{III} register j_1 times and the β_{III} register j_2 times. LS , "j" shifts the β_{III} register j times.

RS

shifts each column $(\alpha_{(1,)}, \beta_{(1,)})$ of the $(\alpha_{III}, \beta_{III})$ registers into the (i+1)th position, placing the last column into the 1st position.

RS "j" repeats RS j times. RS "j_1, j_2" shifts the α_{III} and β_{III} registers separately, the α_{III} register j_1 times and the β_{III} register j_2 times. RS , "j" shifts the β_{III} register j times.

UP

shifts each component of each column $(\alpha_{(i,j)}, \beta_{(i,j)})$ of the $(\alpha_{III}, \beta_{III})$ registers into the (i-1)th position, placing the 1st component into the last position of the column.

UP "j" repeats UP j times. UP "j_1, j_2" shifts the α_{III} and β_{III} registers separately, the α_{III} register j_1 times and the β_{III} register j_2 times. UP , "j" shifts the β_{III} register j times.

DOWN

shifts each component of each column $(\alpha_{(i,j)}, \beta_{(i,j)})$ of the $(\alpha_{III}, \beta_{III})$ registers into the (i+1)th position, placing the last component into the 1st position in the column.

DOWN "j" repeats DOWN j times. DOWN "j_1, j_2" shifts the α_{III} and β_{III} registers separately, the α_{III} register j_1 times and the β_{III} register j_2 times. DOWN , "j" shifts the β_{III} register j times.

NOTE: For RS, LS, UP, and DOWN a negative operand implies the inverse operator. For example, DOWN , -11 is equivalent to UP , 11.

MAX

sets each column equal to the maximum value in each column separately for both the real and complex components.

REFL

interchanges the α_{III} and β_{III} working registers.

DEL — executes LII REAL DEL on the real and imaginary parts of each column of the operand 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, extrapolating to get the final component of each result.

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

ID — If the (α_{III}, β_{III}) registers are dimensioned (n,m), ID places m complex vectors which each form a unit square at the origin with vertices at (1,1) (-1,1), (-1,-1), (1,-1) in the (α_{III}, β_{III}) registers.

ID · — If the (α_{III}, β_{III}) registers are dimensioned (n,m) ID · places m complex vectors which each form a unit circle centered at the origin in the (α_{III}, β_{III}) registers.

4.10 DEFINITIONS OF L0 SUB AND EVAL

The level 0 EVAL key is useful for extracting integer data from data structures on levels I, II, and III. The SUB key is used to insert integer data in data structures on levels I, II, and III. (Note: these definitions of SUB and EVAL are exclusive to MOLSF.)

EVAL "j"	takes the nearest integer to the j^{th} component of the β_{II} working register and places it in the L0 quotient register.
EVAL 0	loads the value of the β_{II} working register display scale into the L0 quotient register.
CMPLX EVAL 0	loads the common scale of α_{II} and β_{II} into the L0 quotient register.
EVAL "A"	takes the nearest integer to the value in LI REAL "A" and places it in the L0 quotient register.
EVAL	takes the nearest integer to the value in the β_I working register and places it in the L0 quotient register.
EVAL +	takes the least integer greater than or equal to the value in the β_I working register and places it in the L0 quotient register.
EVAL -	takes the greatest integer less than or equal to the value in the β_I working register and places it in the L0 quotient register.
EVAL CTX	loads the current LII context into the L0 quotient register.
EVAL CTX "A"	loads the context of the LII vector "A" into the L0 quotient register.
SUB "A"	takes the integer in the L0 quotient register and stores it in the LI storage location "A".

SUB "j"	takes the integer in the L0 quotient register and stores it in the "j"th component of the β_{II} working register.
REAL SUB 0	modifies the LII REAL relative display scale so that the actual display scale will be equal to the integer in the quotient register.
CMPLX SUB 0	modifies the LII CMPLX relative display scale so that the actual display scale will be equal to the integer in the quotient register.
SUB (anything else)	takes the integer in the L0 quotient register and stores it in the β_I working register.

4.11 DEFINITION OF LEVEL V OPERATORS

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 the 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 either 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 either 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.

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

LII REAL ID ⊙ 2 ⊕ 1 ∅ (ID ⊙ 3 ⊕ 1)

DISPLAY RETURN

A comparison of the parenthetical and nonparenthetical programs for evaluating (2X + 1) / (3X + 1) indicates two distinct advantages for the parenthetical format:

1) Temporary storage (e.g., STORE A) is allocated by the system when parentheses are used.

2) Utilization of parentheses allows arithmetic expressions to be entered in a format which is familiar to the casual user. This is of particular value to the new user who has a FORTRAN IV or similar programming language background.

Nesting of parentheses in the usual manner is perfectly acceptable. It is helpful to remember that parentheses are needed to make an operand of an operator key, as in

SIN A ⊕ (COS A) ⊙ 3

to compute 3(sin A + cos A). Here the expression (cos A) is the operand for the operator ⊕ . If the parentheses around COS A were omitted, in this example, the addition operator would be lost, for want of an operand, the value cos A would replace the value sin A in the β_{II} working register, and the result would be simply 3 cos A.

EXAMPLES:

1) LII REAL SIN (SQ Y ⊕ (COS Y ⊙ (COS (Y ⊕ LI T))))

leaves the function sin (Y^2 ⊕ cos Y cos (Y ⊕ T) in the β_{II} register. (Y is a real vector and T is a level I constant.)

2) LII REAL EXP (NEG (SQ (ID ⊙ 5))) DISPLAY RETURN

generates the function e^{-X^2}, for $-5 \leq X \leq 5$, and displays it on the scope. Note that, since this computation includes mostly operators that can operate directly on the β_{II} register, it is more straightforward to program it.

LII REAL ID ⊙ 5 SQ NEG EXP DISPLAY RETURN

Care must be exercised in constructing parenthetical expressions to insure that there are the same number of right parentheses as left. When computations are performed manually, if there are too few right parentheses, the console will wait until the ")" key (or RESET) is pushed. If a console program has open parenthetical expressions, its execution will stop until the ")" is pushed manually.

DISPLAY PARENTHESIZED EXPRESSION

When the user constructs a parenthetical expression, a false TEST key may be inserted anywhere after the outermost left parenthesis and before any right parenthesis. After the system has been given all of the expression (the same number of left and right parentheses), it will cause the coding, generated by the system to evaluate the expression, to be displayed, as illustrated below:

KEYBOARD ENTRY	CRT RESPONSE
LI REAL LOAD 1 ⊕	
(TEST 2 SIN + (	
1 ⊕ (2 COS)))	STORE TEMP1 LOAD 2
	SIN STORE TEMP2 LOAD
	2 COS ⊕ 1 ⊕ TEMP2 STORE
	TEMP2 LOAD TEMP1
DISPLAY RETURN	2.49315+00

TEMP1, etc. are temporary storage locations on MOLSF which are inaccessible to the user. The first thing MOLSF does is to store the working register under TEMP1. This is done so that the first operator, ⊕ , which has the parenthesized operand will operate on the correct contents of the working register. In the evaluation of the expression the system used the storage location TEMP2 and put the final result of the parenthesized expression in TEMP2 for the use of that first operator.

LOAD TEMP1

In most instances, if a user wants to use the contents of the working register later, he must explicitly store it for later recall. In a parenthesized expression, a matched pair of parentheses, "()", with no intervening keys instructs the on-line system to use TEMP1 as the specified operand. As its name implies, TEMP1 is a temporary storage location which holds the contents of the working register before the parenthesized expression is executed.

If an operand is omitted in a parenthesized expression, the on-line system assumes that TEMP1 is the proper operand. Thus SIN A ⊕ (SQ ( ) ⊙) is a valid expression. It would be executed as SIN A + $\sin^3$ A.

HIERARCHY OF OPERATORS

In addition to the normal MOLSF computation a user may specify that a parenthesized expression be executed using

FORTRAN's hierarchy of operators. This is accomplished by using the lower keyboard keys *, /, and ** as if they were the FORTRAN operators. When these keys appear in a parenthesized expression computation is performed left to right according to the following hierarchy:

1) Evaluation of functions
2) Exponentiation "**"
3) Multiplication "*" and division "/"
4) Addition "⊕", subtraction "⊖", multiplication "⊙", and division "⊘".

KEYS NOT ALLOWED IN A PARENTHETICAL EXPRESSION

The keys RESET, LIST, TYPE, USER, SYST, REPT, LO, SEL, ENL, CON, and CASE are not allowed in a parenthetical expression. Attempts to use the above keys will produce the error message "PAREN ERROR_ _INVALID EXPR".

Appendix A

PROCEDURE TO OPEN AN ON-LINE ACCOUNT

A considerable amount of information is necessary for the Computer Center to set up a user number. To provide the staff with this information a new user must fill out the form: "REQUEST FOR OLS USER NUMBER (OR CHANGE)", (see figure A.1). This form may be picked up at the Computer Center office. Note: to open an on-line account, one must have a valid Computer Center account number.

Complete the application form and return it to the Computer Center office. You will be contacted when an on-line user number has been assigned to you.

REQUEST FOR OLS USER NUMBER (OR CHANGE)

IF A CHANGE IS REQUESTED, ENTER USER NO._____

INDICATE REQUESTED CHANGES ONLY, AND SIGN.

ACCOUNT CONTROLLER_____ PHONE_____

PRINCIPAL USER_____ PHONE_____

COMPUTER CENTER ACCOUNT NUMBER_____

FUNDS ALLOCATED TO THIS USER NO._____

TERMINATION DATE FOR THIS USER NO._____

ID NUMBER (UP TO 8 HEX DIGITS)_____

AUTHORIZED NAMES ON THIS USER NUMBER (THESE WILL BE CHECKED AT SIGN ON. YOU MAY ALSO SPECIFY INDIVIDUAL LIMITS ON FUNDS AND LIBRARY STORAGE).

NAME (16 CHARACTER MAX.)	FUNDS (SPECIFY AMT.)	LIBRARY STORAGE

DO YOU WISH TO ENTER A PROBLEM NAME? YES _____ NO _____

MAXIMUM LIBRARY STORAGE IN BYTES (DEFAULT=150K)_____

MAXIMUM VECTOR LENGTH (DEFAULT=101)_____

MAXIMUM ARRAY DIMENSION (DEFAULT=20)_____

DO YOU PLAN TO USE PRIVATE DISK PACKS? YES _____ NO _____

SIGNATURE_____

FOR COMPUTER CENTER USE ONLY	
DATE RECEIVED	
APPROVED	
DATE COMPLETED	
REPLY	

Figure A.1

Appendix B

The implementation of new features may occasionally introduce errors into the existing system. Consequently, if you encounter problems which in your judgement can be attributed to software or hardware failures, please notify the Computer Center at (805) 961-2274. If on-line personnel are available, your call will be routed to the appropriate person. If no one is available, the dispatcher will take down a message to be forwarded to the appropriate person.

Figure B.1, "OLS USER COMPLAINT," supplies the information that Computer Center personnel need to solve your problem. Particularly important is the key sequence which gave you the error. Once you have notified the Computer Center of your suspected problem, we shall attempt to solve the problem as soon as possible.

OLS USER COMPLAINT

USER'S NAME_____PHONE NUMBER_____

LOCATION_____DATE_____TIME_____

USER#_____USER SYSTEM NAME_____

DESCRIPTION OF PROBLEM (INCLUDE SEQUENCE OF BUTTONS CAUSING TROUBLE):

REFERRED TO_____

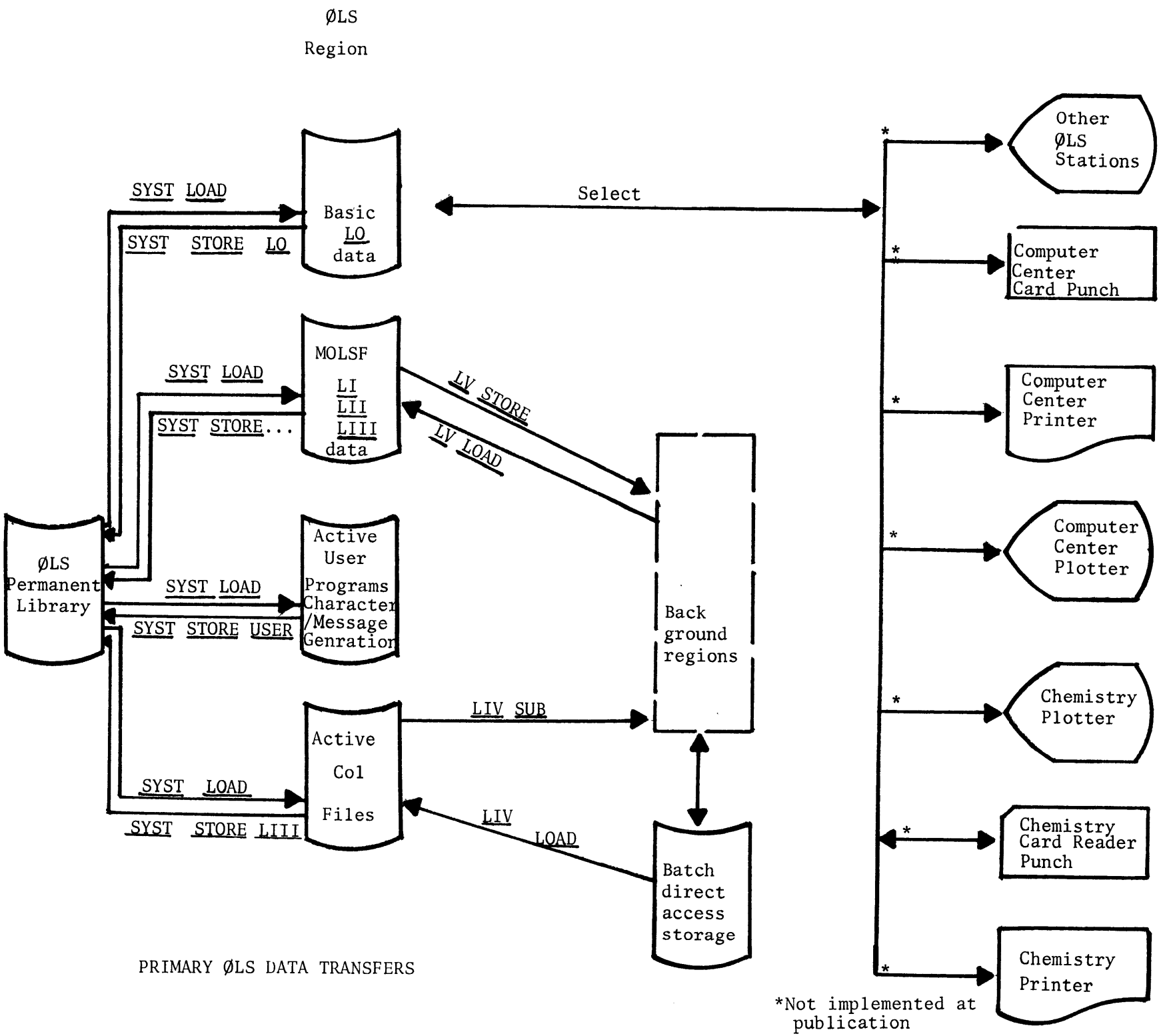
DISPOSITION:

USER'S CALL RETURNED BY_____DATE_____

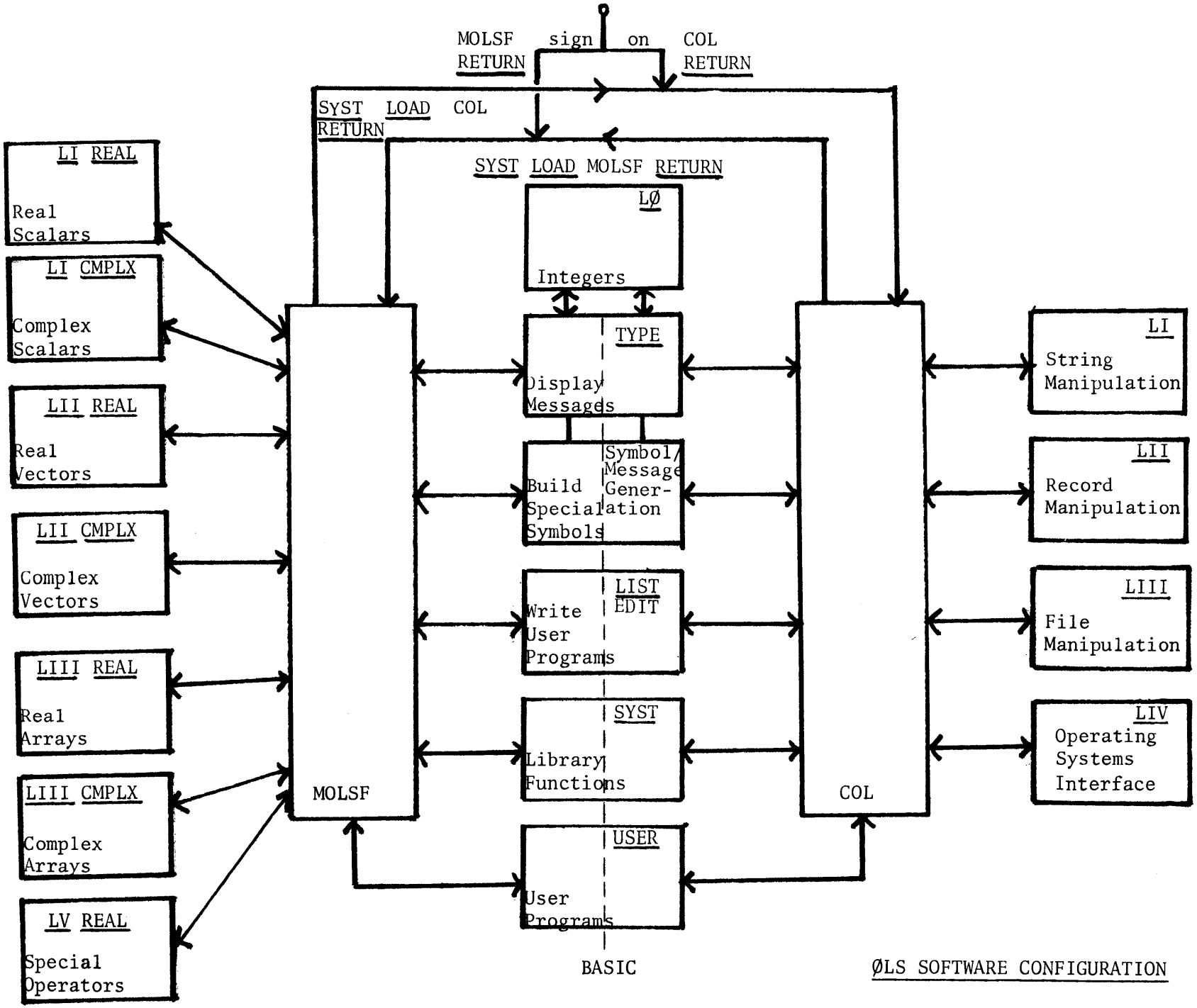
Figure B.1

Appendix C

ØLS SOFTWARE STRUCTURE & KEYBOARD DIAGRAMS

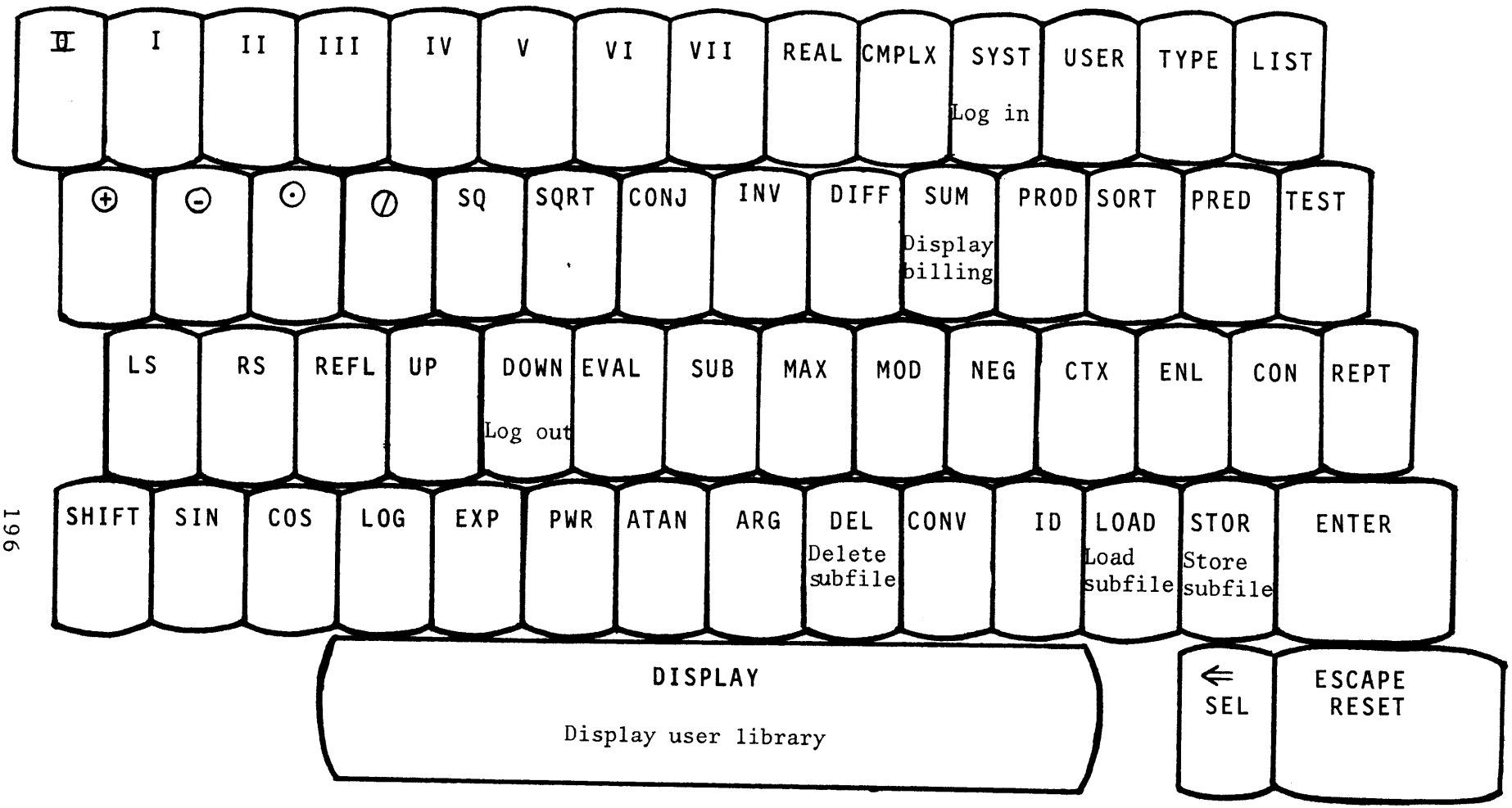


PRIMARY ØLS DATA TRANSFERS



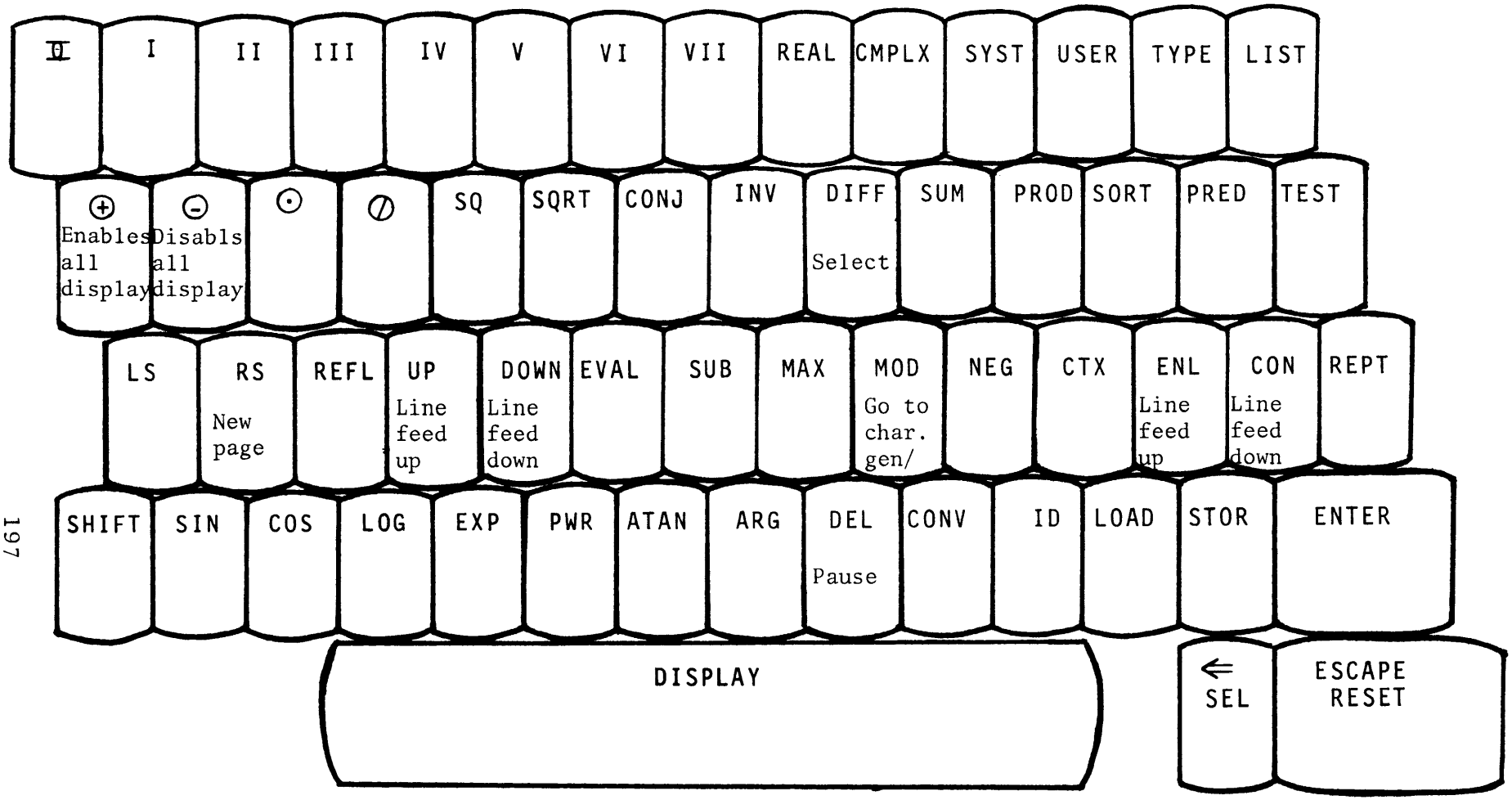
ØLS SOFTWARE CONFIGURATION

BASIC SYSTEM: SYST KEYBOARD



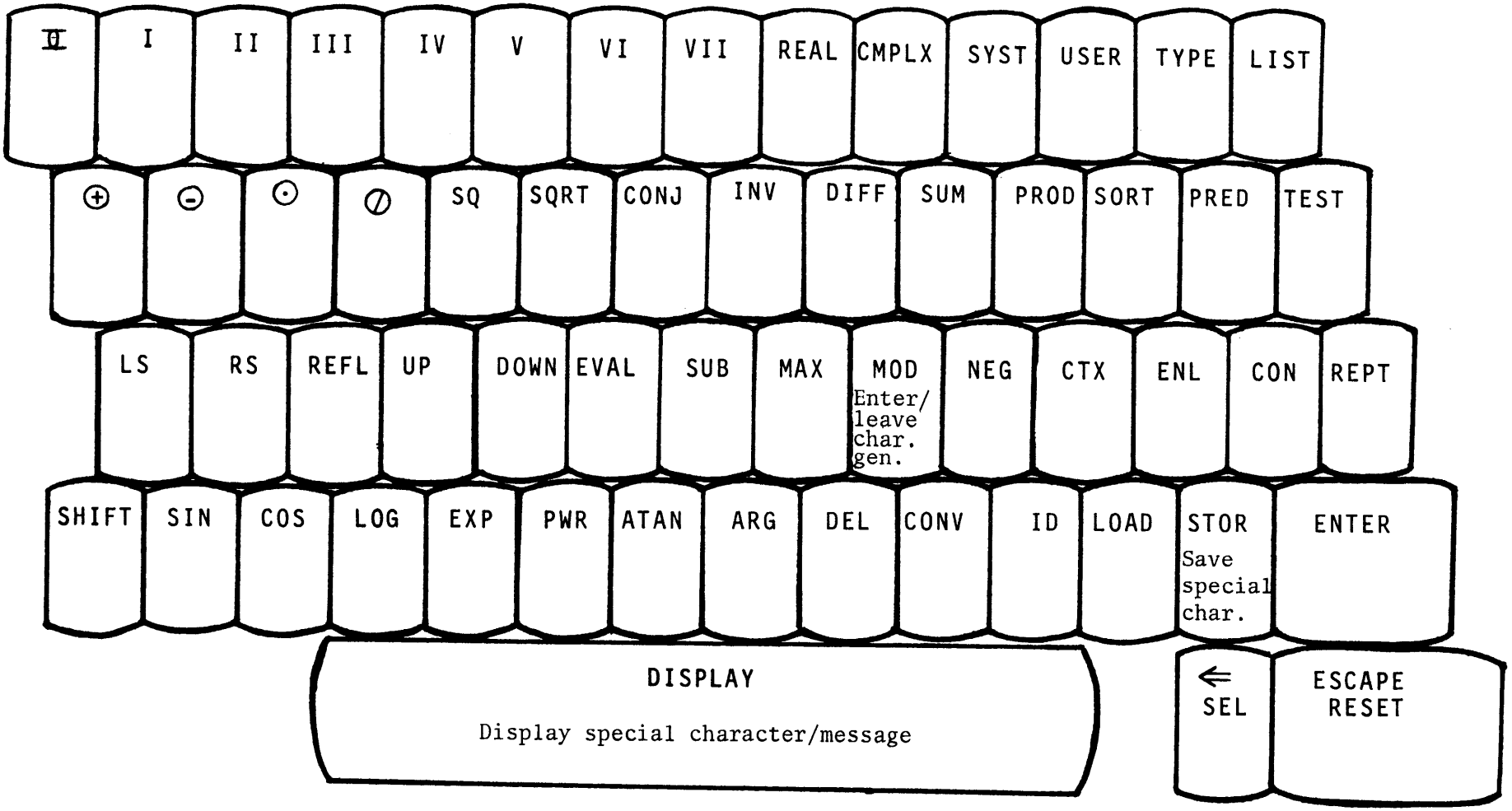
,(after display): Displays all subfiles with common names

BASIC SYSTEM: TYPE KEYBOARD



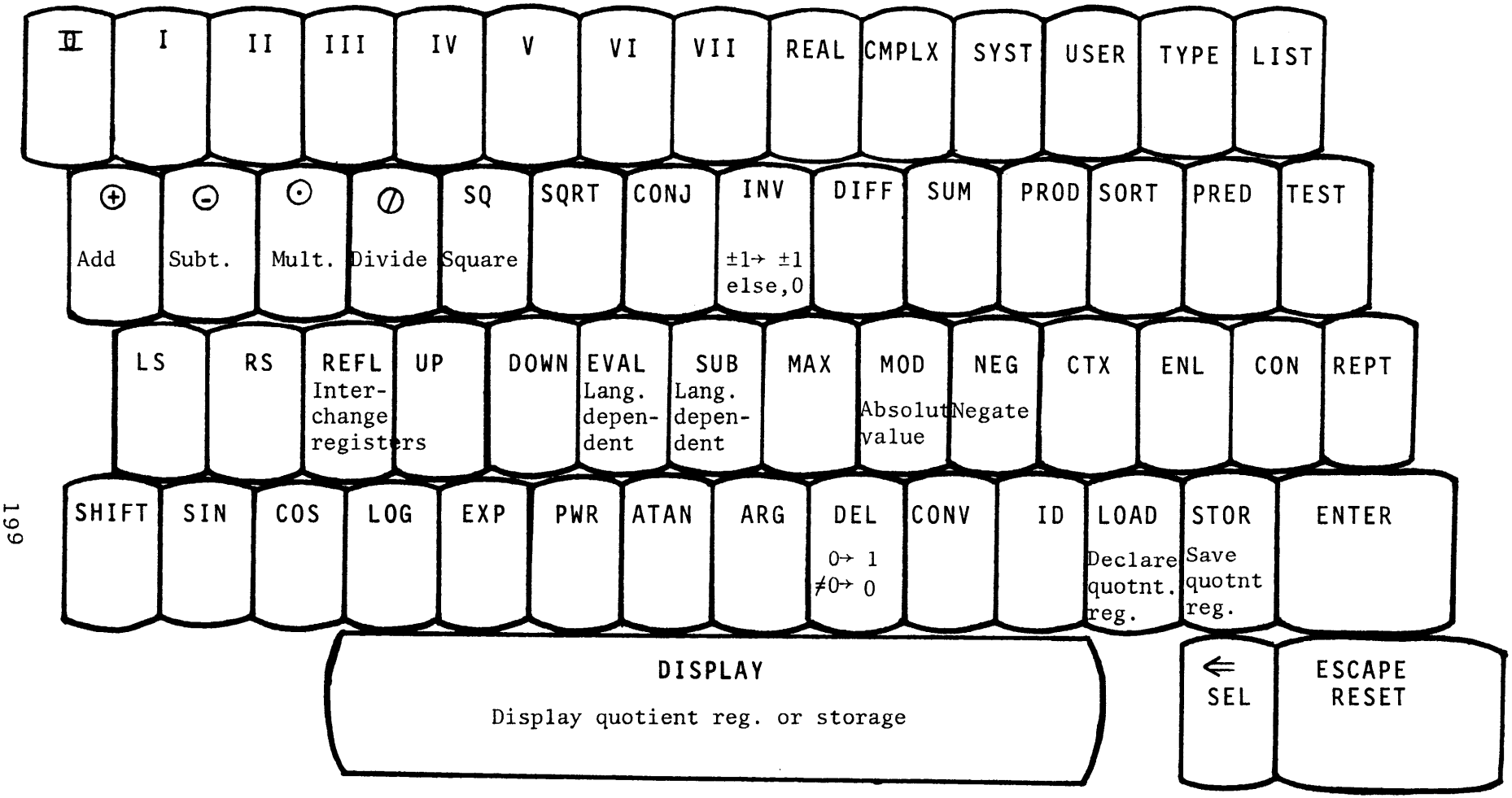
Return: Carriage return.
Space : Advance to the right.
Back : Backspace to the left.

BASIC SYSTEM: MESSAGE/CHARACTER GENERATION KEYBOARD

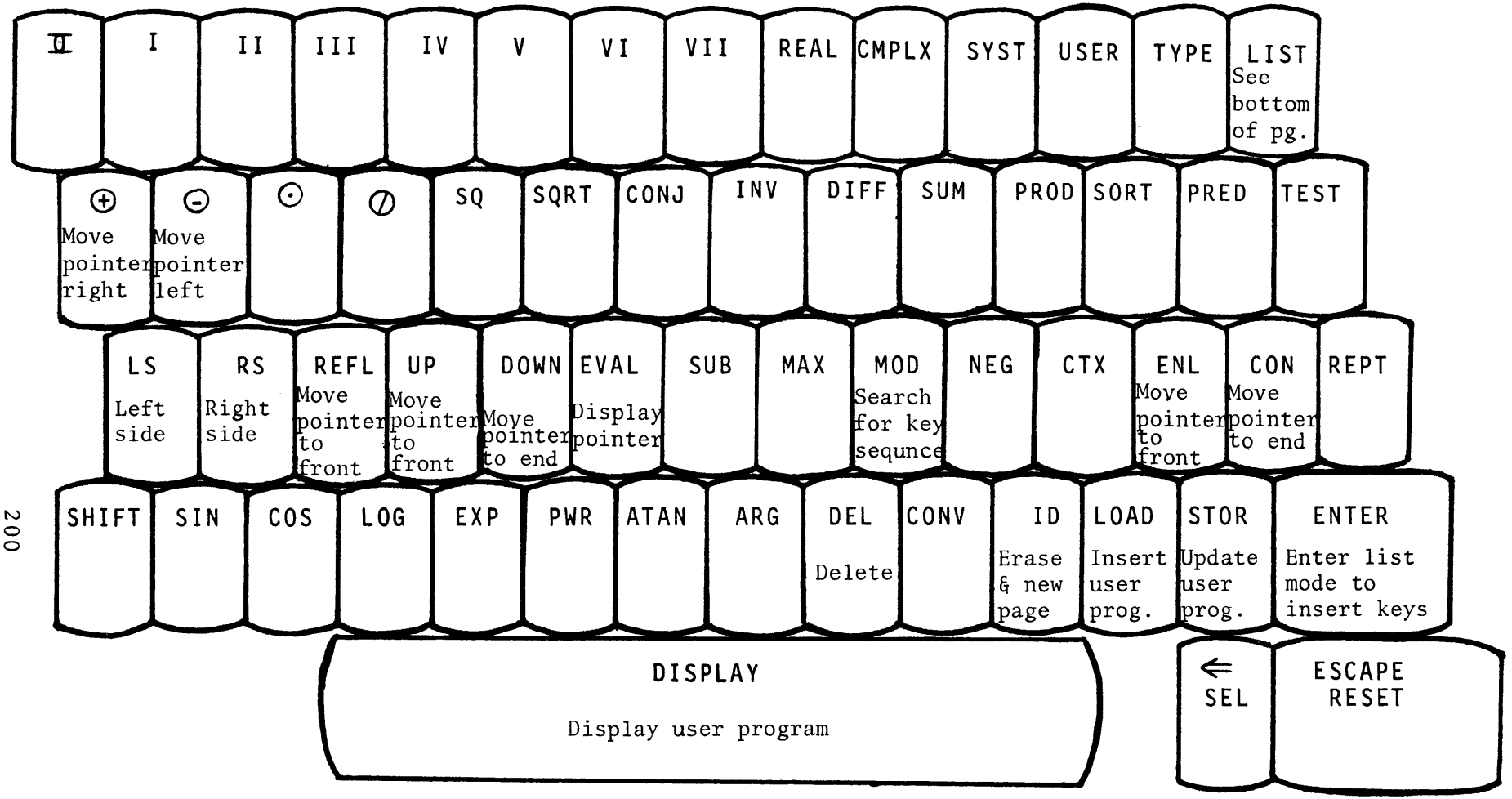


Back : Delete last direction keypush.
Decimal point: Reposition the dot.

BASIC SYSTEM: LO KEYBOARD

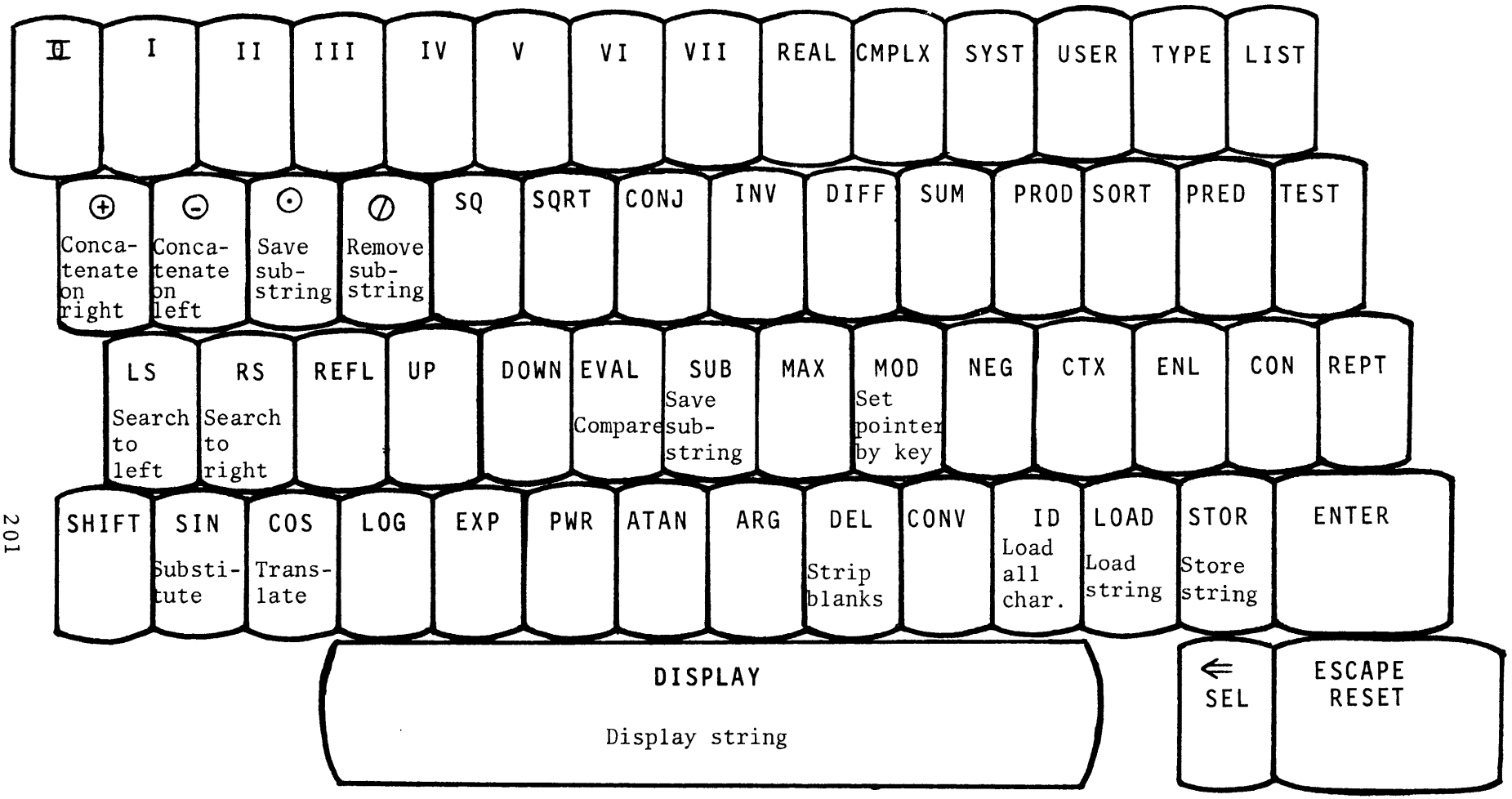


BASIC SYSTEM: EDIT LEVEL KEYBOARD

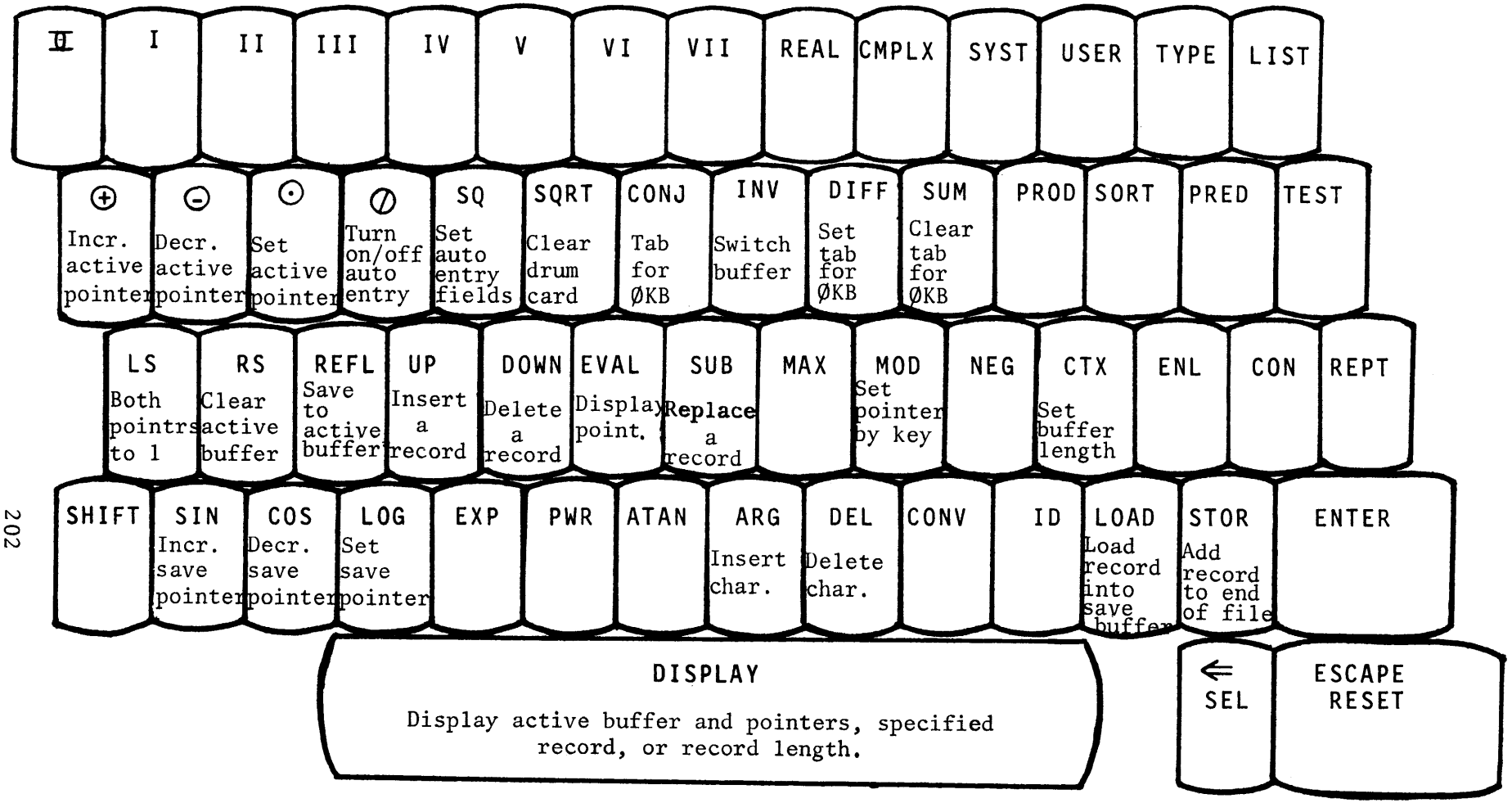


Back : Delete one key to left of pointer.
Space: Delete one key to right of pointer.
List : Leave EDIT level, enter LIST mode.

COL: LI KEYBOARD

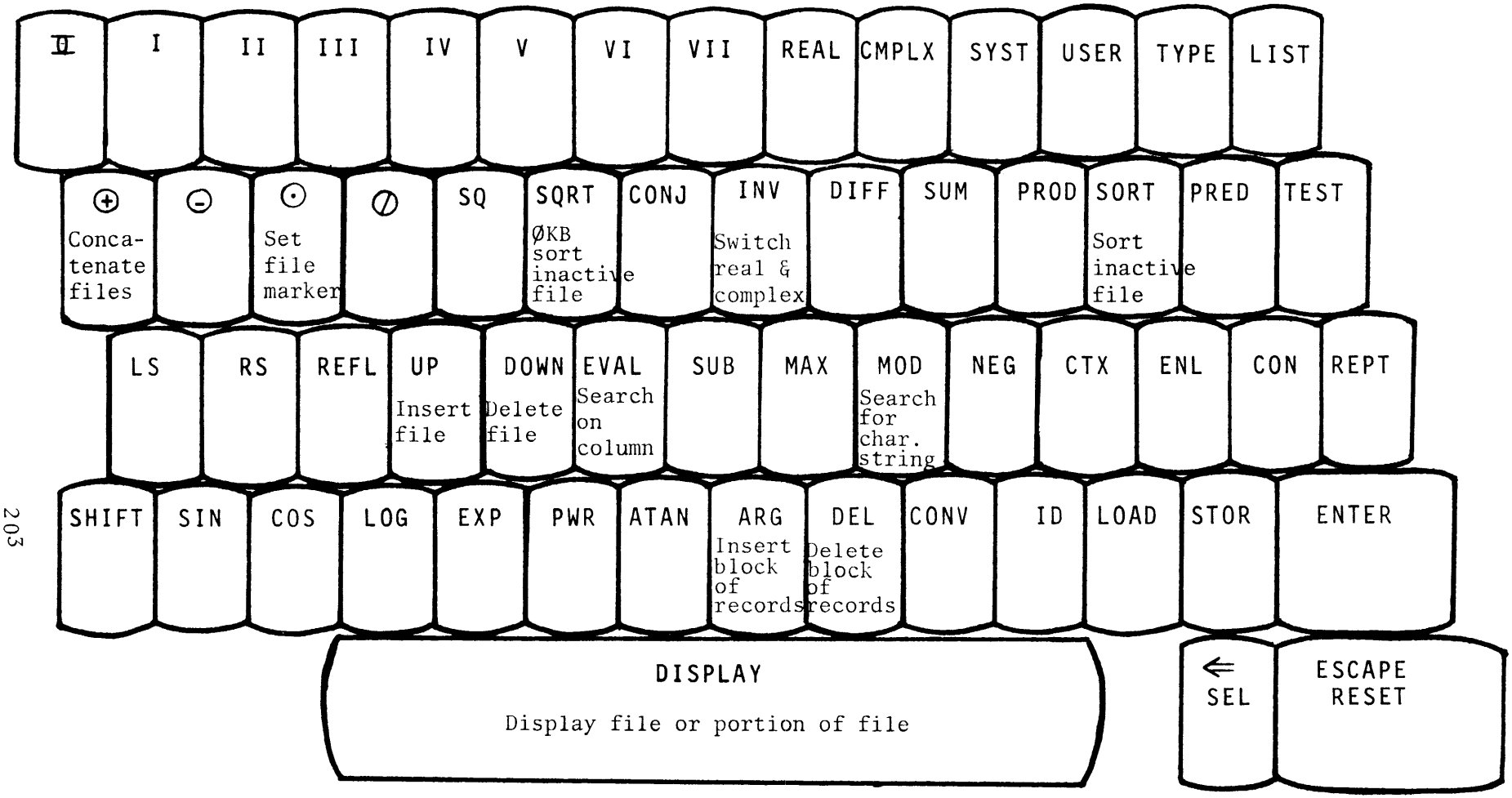


COL: LII KEYBOARD (KEYPUNCH)

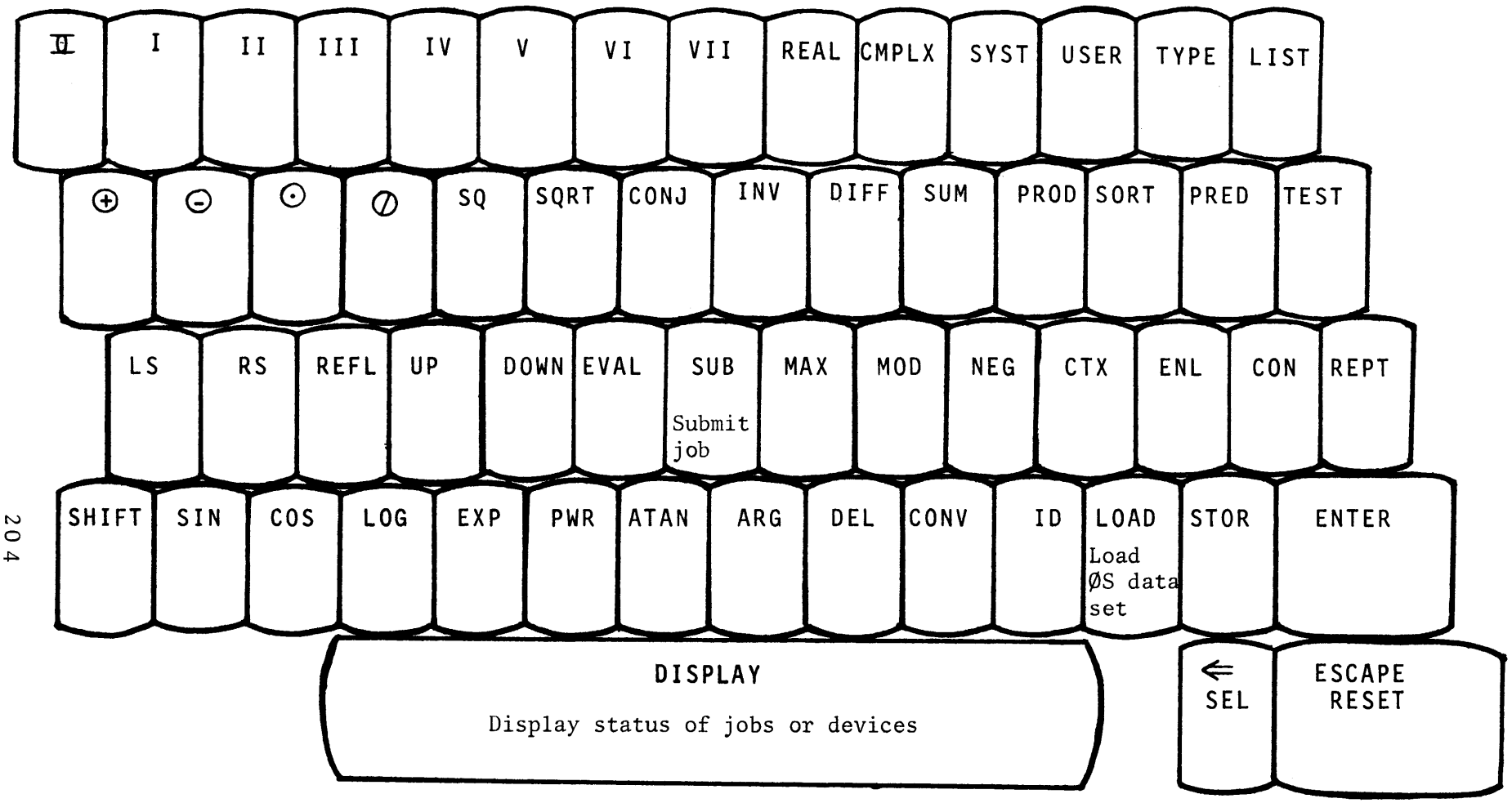


Back: Display preceeding record or move active buffer pointer left.
Tab : Tab.
Set:: Set tabs.
Clr : Clear tabs.

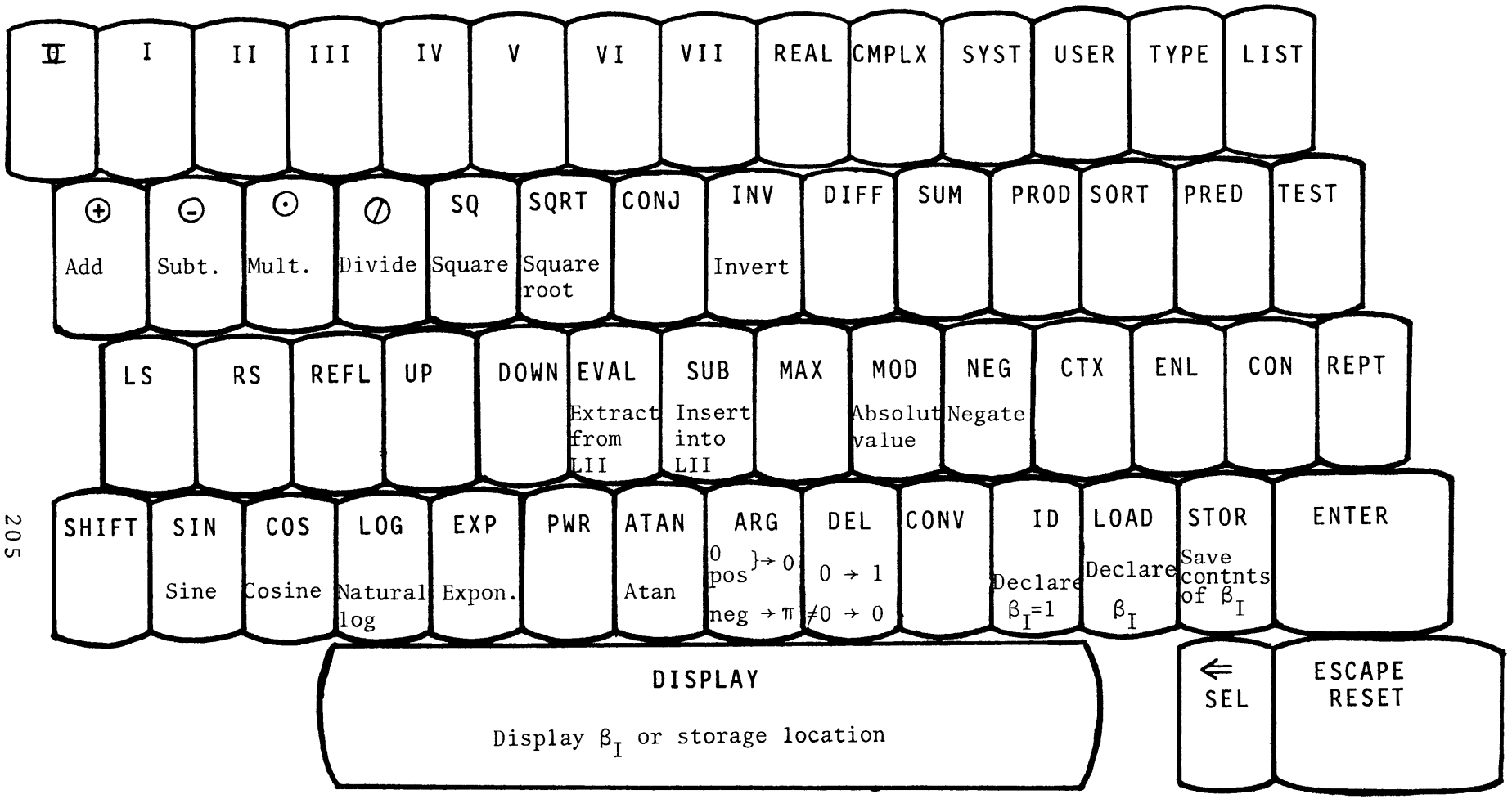
COL: LIII KEYBOARD



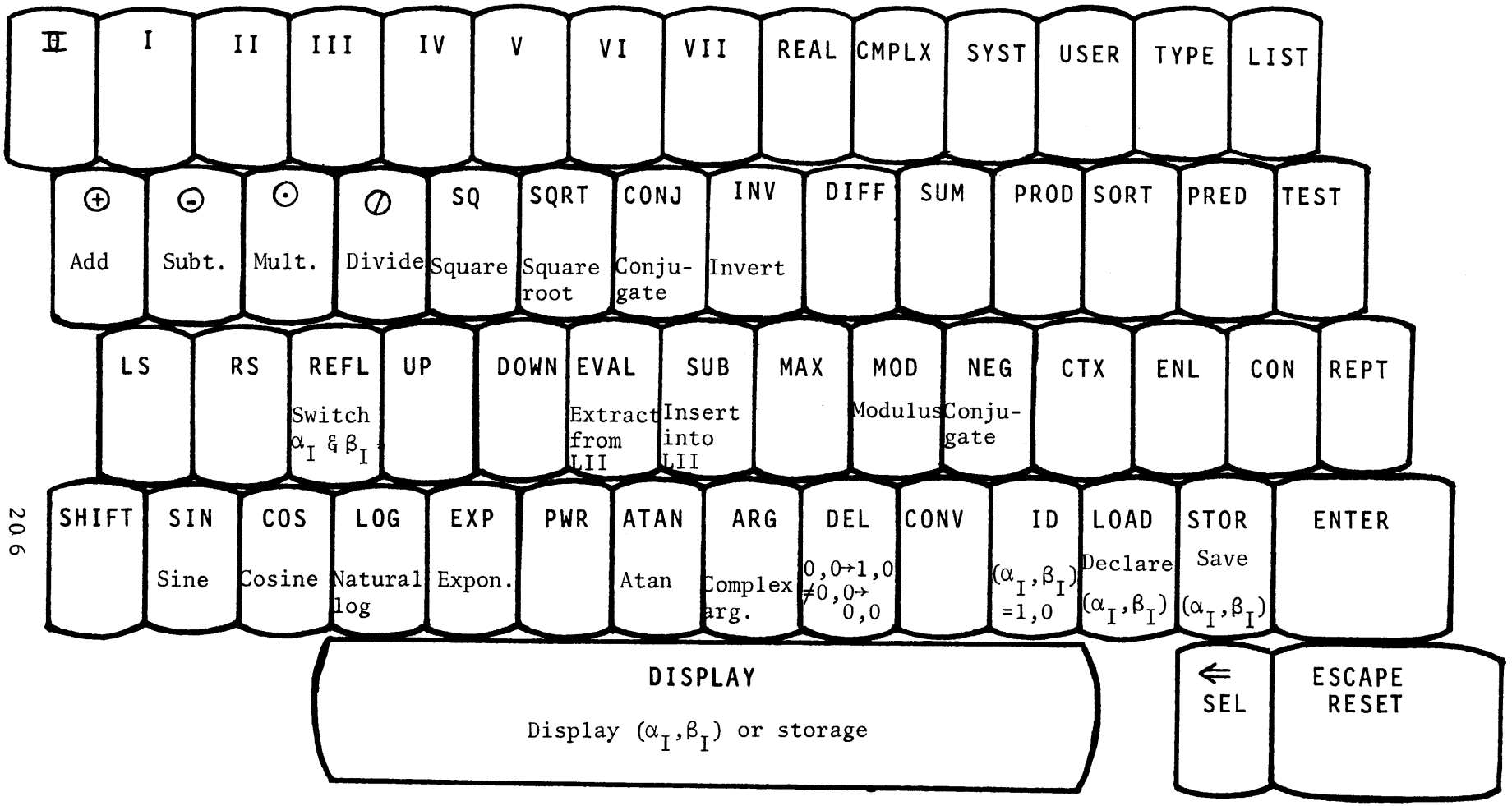
COL: LIV KEYBOARD



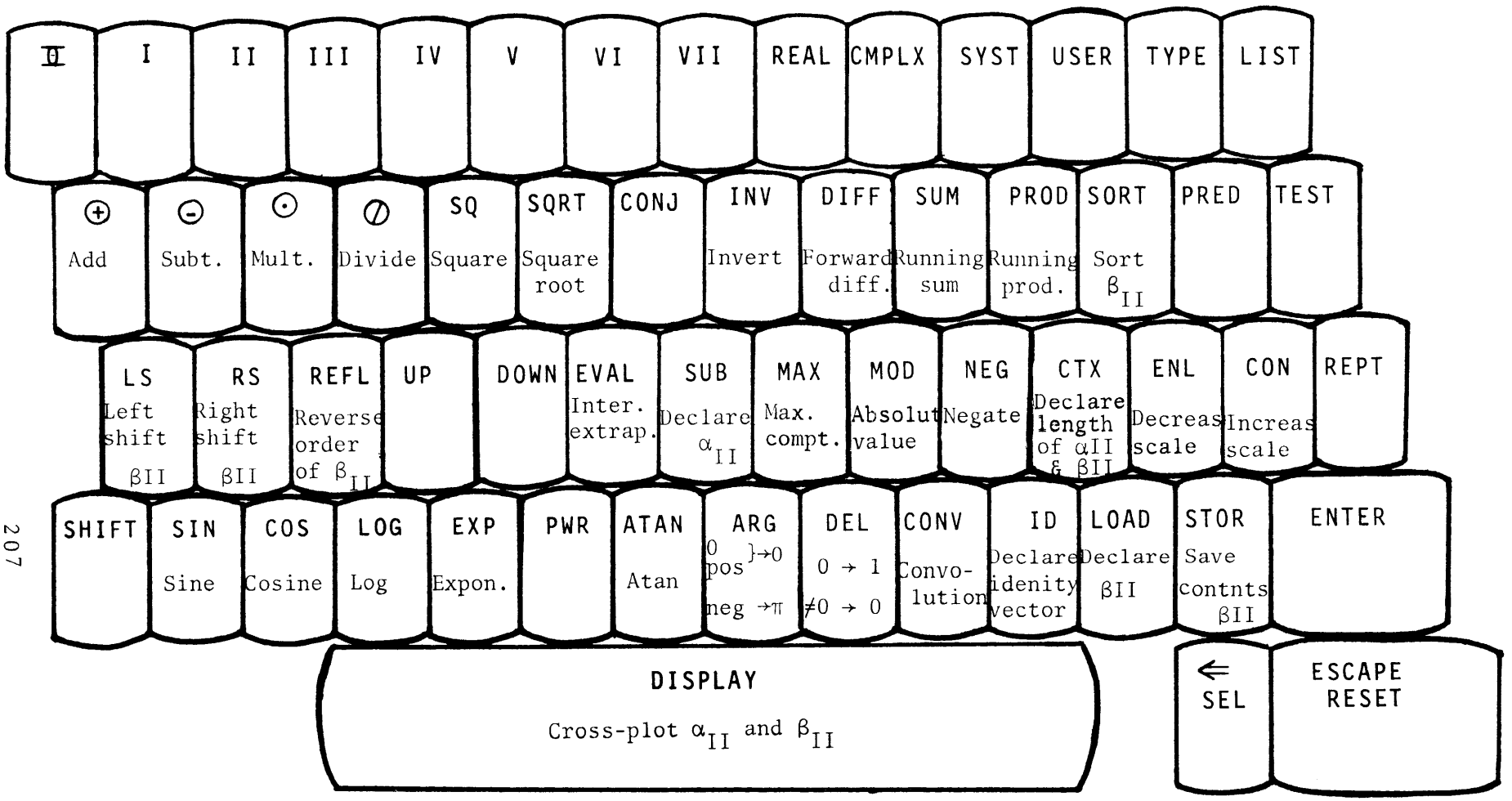
MOLSF: L1 REAL KEYBOARD



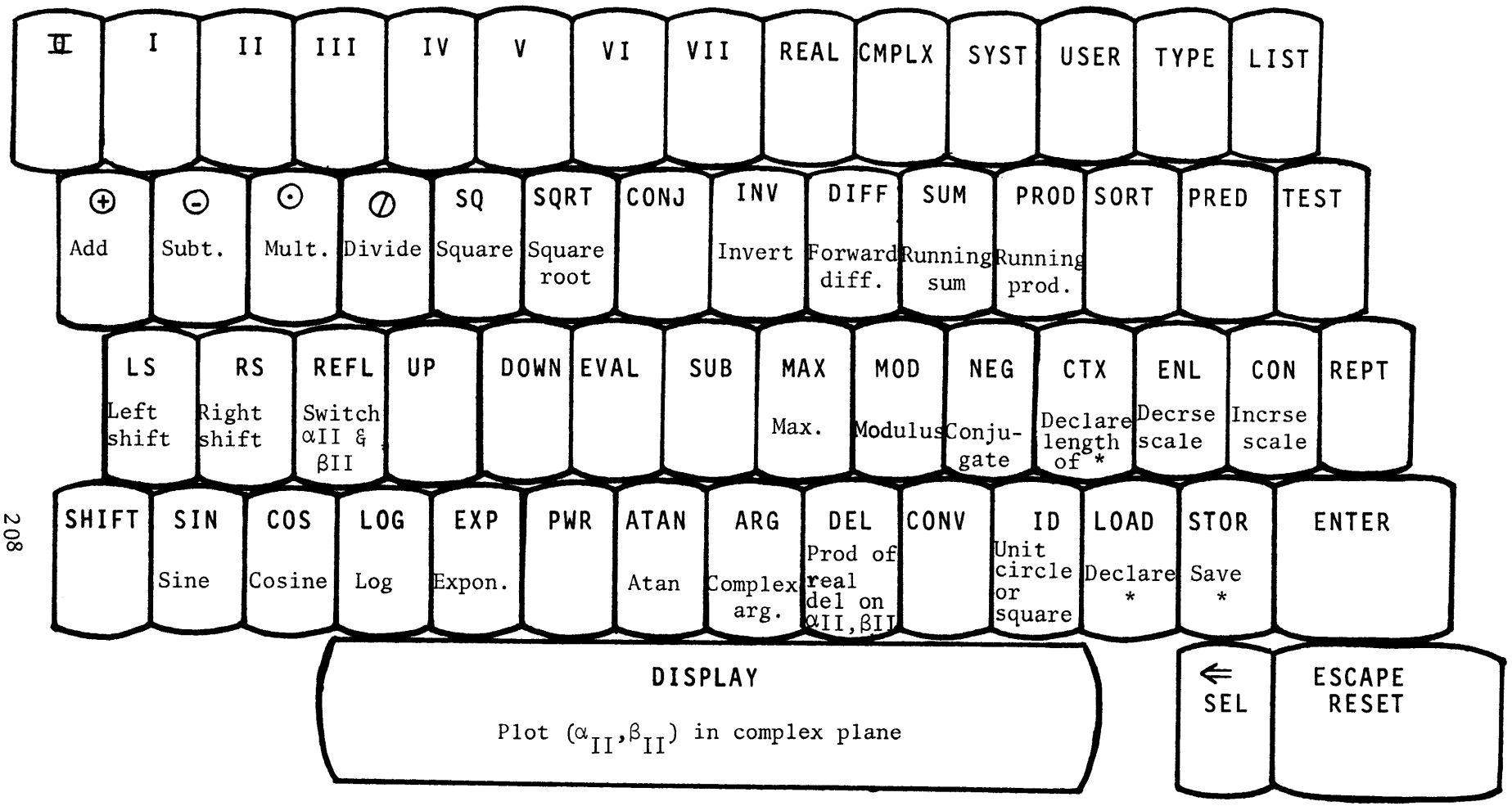
MOLSF: L1 COMPLEX KEYBOARD



MOLSF: LII REAL KEYBOARD

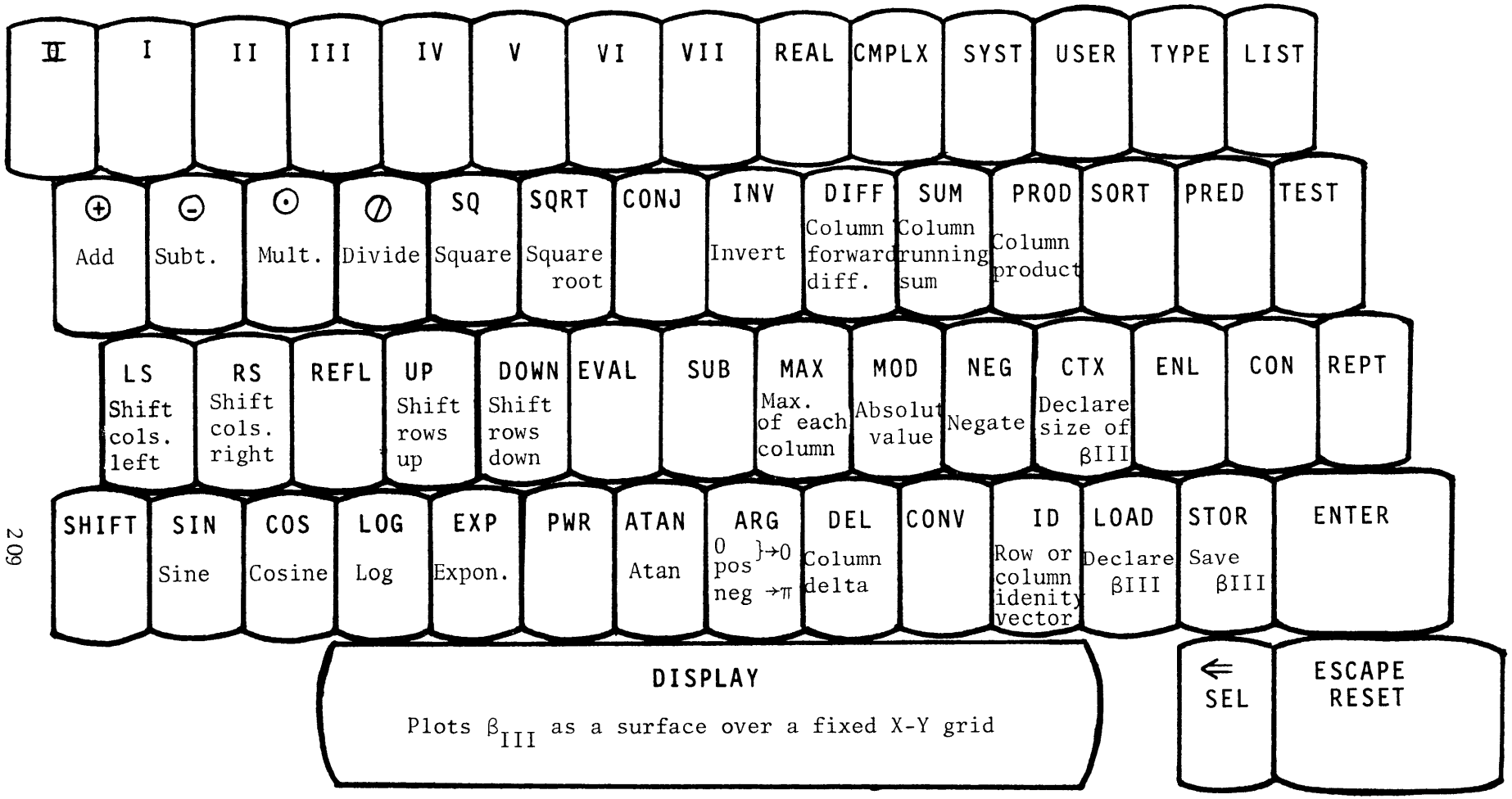


MOLSF: LII COMPLEX KEYBOARD

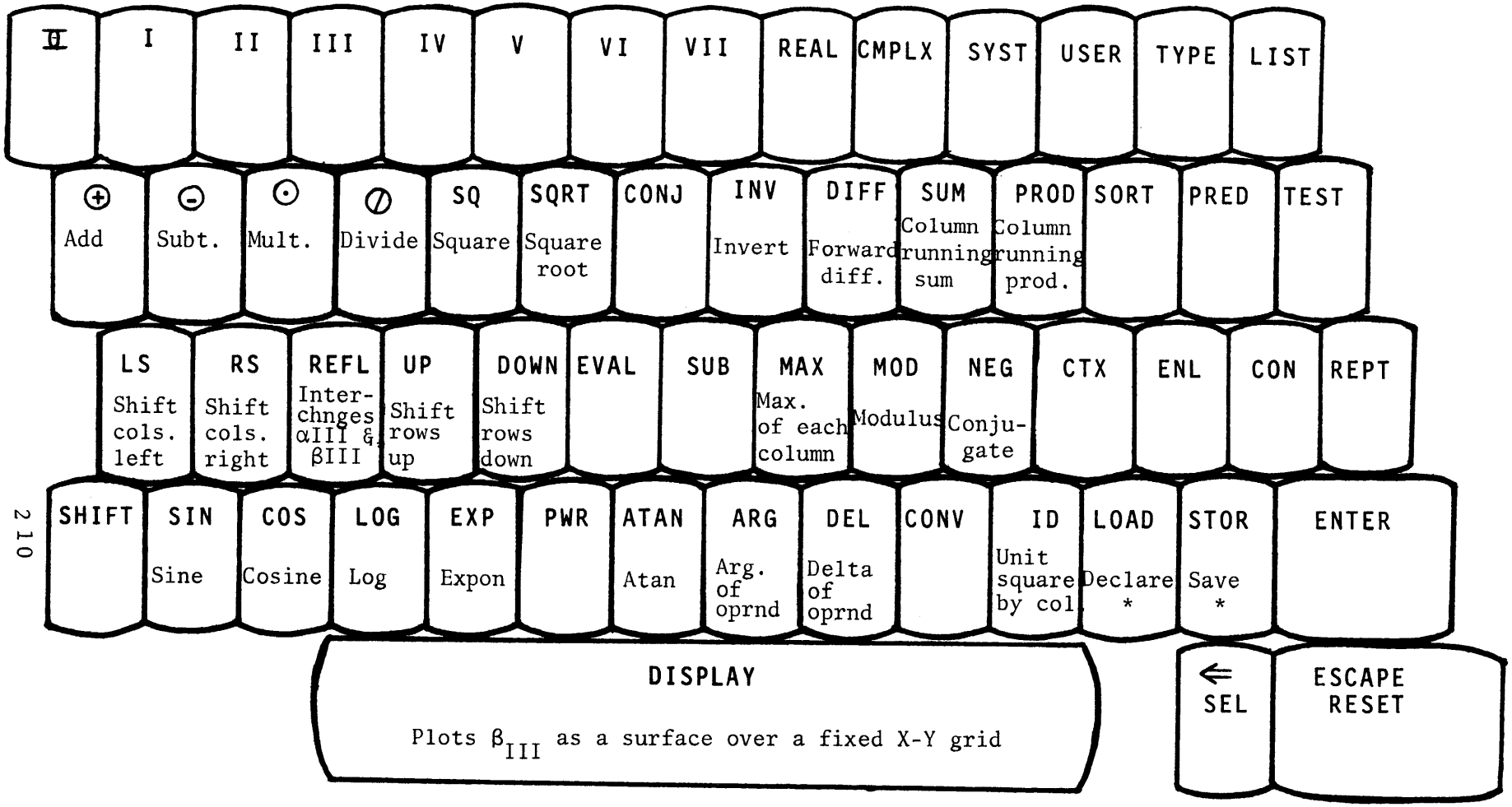


* (α_{II},β_{II})

MOLSF: LIII REAL KEYBOARD

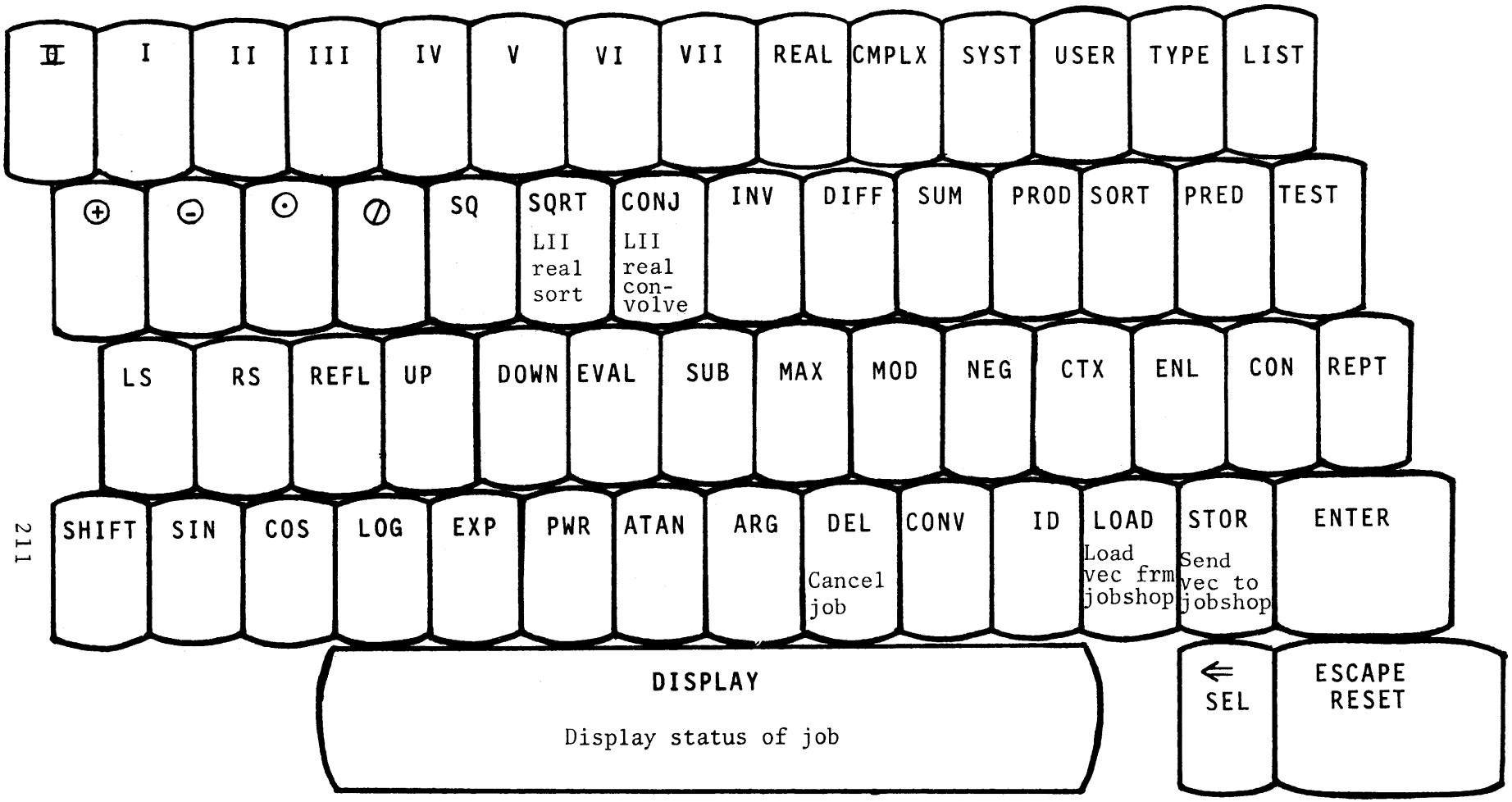


MOLSF: LIII COMPLEX KEYBOARD



* $(\alpha_{III}, \beta_{III})$

MOLSF: LV REAL KEYBOARD



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.

System messages are followed by a page number in the text which points to the explanation of the message.

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.

Check your data and the order of your operators (for example, the division of a very large number by a very small value) and try again.

FILE LOADED See page 23

FILE NAME = See page 21

FIXED POINT DIVIDE CHECK

⊘ or INV with zero operand.

Division by zero on level 0 has been attempted.

Check your operand.

FORMAT IS Format See page 153

ID NUMBER = See page 15

INSUFFICIENT STORAGE

STORE USER level operator.

There is no more room for storing USER programs on the specified USER level.

Store your USER program on another level or delete unwanted programs and try to store again.

INVALID FILE TYPE

SYST STORE level mode

You have specified an invalid subfile type.

Try again, specifying what type of subfile you wish to store.

INVALID ID NUMBER

RETURN (after entering an invalid ID number)

Your entered values are not in accordance with the accounting codes stored in the computer.

Inadvertantly, you probably pressed the wrong key, try again.

INVALID JOB CARD

LIV SUB (on COL)

You have attempted to submit through remote job entry a COL file which contains a bad JOB card.

Go to level II and correct the JOB card, then resubmit your file; or, make sure you submitted the proper file.

INVALID PARAMETER

Various key sequences.

A parameter list has been specified that is clearly invalid.

Try again.

INVALID USER NO.

RETURN (after entering an invalid user number)

Your entered values are not in accordance with the accounting codes stored in the computer.

Try again.

JOB HAS NO STEPS

LIV SUB . . . RETURN (on COL)

You have attempted to submit through remote job entry a COL file which contains only a JOB card.

JOB NAME = See page 15

level mode operator UNDEFINED

level mode operator

The operator is not defined.

Use a different operator.

level operator UPDATED See page 46

LOAD See page 22

MEMBER LIST FULL

LII STORE or UP (on COL)

You have attempted to store more records in the active file than the on-line system can handle.

Divide your file into smaller files and store them in the library.

MODULE SECURITY CHECK

SYST LOAD . . . RETURN

You have attempted to load a language which is reserved or restricted to Computer Center personnel.

Load another file name.

NO MATCH

MOD . . . (on COL)

The character string you were searching for does not exist in the active buffer or active string.

Display the active buffer or active string to make sure the character string exists, if it does, repeat the search.

NONEXISTENT STRING

MOD on the EDIT level specified a nonexistent string of keys.

The EDIT pointer remains unchanged. The user program is not affected.

Respecify if desired.

NO RECORDS IN FILE

LIV SUB

You have attempted to submit through remote job entry a COL file which has no records.

Check to make sure you have submitted the proper file.

NO SUCH OCCURRENCE

MOD . . . RETURN integer RETURN (on COL)

The multiple occurrence of the character string you were searching for does not exist.

Check the second operand of MOD to make sure the occurrence of the first operand you requested does exist.

NOT FOUND ON VOLUME

LIV LOAD unit RETURN volume RETURN dsname RETURN (on COL)

The data set name or member name requested can not be found on the specified volume.

Check the data set name or member name and volume name. If they are correct then the data set does not exist.

NO TEXT

STORE USER level operator

You have tried to store a program which has no keys.

Build a proper list and then store it. Note: to delete a program use: DEL on the EDIT level.

NO TEXT IN PARAMETER LIST

USER level operator ( )

A null parameter list was passed to a program containing PRED.

Re-execute the program, entering a parameter list.

Note: A null argument may be specified by inserting a comma between the parentheses.

OPERATION ABORTED

An invalid operand

The operand contained a key that was not valid for the requested operator.

Check youe key sequence and re-enter it.

PARAMETER LIST NOT FOUND

USER level operator

A required data list for a program containing PRED (called by the above keys) has not been passed to that program.

Re-execute the program, entering the parenthesized list required.

PAREN_ERROR INVALID EXPR

(_ _ _)

You have a key which is not allowed in a parenthesized expression, or an invalid operand.

First check the list of invalid keys on page 188. Secondly, check the operands in your expression.

PAREN_ERROR NO TEMPS

(. . .)

Your parenthesized expression has requested more temporary storage space than is available.

Check to see if you have failed to press the closing right parenthesis. Otherwise, you will have to simplify or break down your expression into several expressions.

PAREN_ERROR STACK OVRFLO

(- - -)

Your parenthesized expression is longer than the on line

system can handle.

Check to see if you have failed to press the closing right parenthesis. Otherwise you will have to break down your expression.

RECORD NOT FOUND

DISPLAY, LOAD, SUB, or DEL (on COL)

The requested record of a file cannot be located.

Check the number of the record and try again.

RESERVED NAME---RESPECIFY

RETURN (after entering a file name during a SYST STORE operation).

The name used is already defined as a language.

Use a new name.

RESET COMPLETED See page 40

RESTART COMPLETED See page 17

SEARCH ERROR

LIII DEL, ARG, or UP (on COL)

An operand referenced nonexistent records.

Manually check to see if the operation succeeded. If it failed repeat it with a smaller operand. Please call the Computer Center to report this error.

START LIST See page 44

STORE See page 21

SUBSCRIPT OVERFLOW

Interlevel operands (on MOLSF)

A component of a vector or array requested is outside the

bounds of that vector or array.

Check your subscript values.

SUBSCRIPT UNDERFLOW

Interlevel operands (on MOLSF)

A component of a vector or array requested is negative or zero.

Check your subscript values.

TAB ERROR

LII CONJ or TAB (on COL)

The "drum card" is invalid.

Recreate the drum card with SQ, try again, and please call the Computer Center to report this error.

UNDEFINED FILE

RETURN (after entering a file name).

The computer cannot find the requested file or language.

Re-enter the file name being careful to make sure that it is spelled properly.

UNLOCATABLE PARAMETERS

USER level operator (argument, argument,...)

The value of the PRED operand is negative, zero, or greater than the number of parameters supplied in a PRED parameter list.

Correct the program and/or the list of parameters.

USER level operator UNDEFINED

USER level operator

The program requested has not been defined. Note: programs

have been stored under other operators on this user level.

Create a new program and store it, or perform some other operation.

USER level UNDEFINED

USER level operator

The particular program requested has never been defined: i.e., no program has been stored under any operator key on that level.

Create a new program and store it, or perform some other operation.

USER NAME = See page 15

VOLUME IS BEING MOUNTED See page 100

VOLUME CANNOT BE MOUNTED

LIV LOAD unit RETURN volume RETURN

Either all the units of the type you specified are in use, or the volume you have requested does not exist.

Check the spelling of the volume-serial number; if it was correct wait five minutes and try again, or call the Computer Center and make arrangements for the volume to be mounted.

VOLUMES NEEDED = See page 103

WAITING See page 102

WAITING--RJE BUSY See page 105

WAITING TO PURGE

SYST DEL

Some other user is signed on your user number and is execut-

ing some operations which uses the user number library. Patience, the operation should be completed in less than a minute.

WORK AREAS PURGED See page 16

WORK AREAS UPDATED See page 16

: (Post List Marker) See page 46

APPENDIX E

SAMPLE PROBLEMS

Appendix E consists of sample programs and sample problem solutions using OLS. The problems are divided into five categories as follows:

A. Level I REAL

B. Level I CMPLX

C. Level II REAL

D. Level II CMPLX

E. Inter-level and miscellaneous examples

Most of the examples require the generation of a user program or programs to solve the given problem. This approach has been adopted since this is the normal mode of solving complex problems on OLS. It is possible that some of the user programs given in the examples can be adapted with minor changes to the solution of your own problems.

A. LEVEL I REAL EXAMPLES

A.1 Write a user program to compute the square root of a number N using the Newton-Raphson iteration procedure

$$x_{k+1} = \frac{x_k + N/x_k}{2}$$

where x_k and x_{k+1} are the k^{th} and $(k+1)^{st}$ approximations to the square root of N.

Assume that both x_k and x_{k+1} are to be stored in X, in turn, and the number N, whose square root is to be taken, is stored in N. The user program is to be stored under USER LI SQRT. The user program is constructed as follows:

LIST LI REAL LOAD N ⊘ X ⊕ X ⊘ 2 STORE X DISPLAY
RETURN USER LI LIST
STORE USER LI SQRT

Note that the LIST key is pressed to place the system in the LIST mode, the program is constructed, and LIST is pressed a second time when the program is completed. The sequence STORE USER LI SQRT stores the USER program under USER LI SQRT. To display the user program, the keys:

USER LI DISPLAY SQRT

are pressed.

N and the first guess x_1 to the square root of N are now manually loaded and stored under N and X respectively. Assuming N = 3 and x_1 = 1, the key sequence:

LI REAL LOAD 3 STORE N LOAD 1 STORE X

accomplishes the storage.

To execute the square root program press

USER LI SQRT

The computer responds with the first guess at $\sqrt{3}$ by printing 2 on the output device. Successive depressions of SQRT yield

the following displays on the output device. NOTE: Only the numerical values of the square root are displayed on the output device. However, a user program could be written to generate the column headings and the iteration number.

Iteration No.	Square Root
2	1.75 + 00
3	1.73214 + 00
4	1.73205 + 00
5	1.73205 + 00

After the fifth iteration the square root is determined within the accuracy of MOLSF. It should be noted that the program ends on level USER LI so that, once the user level has been specified, the user program can be executed by simply pressing the key under which it has been stored (in this case SQRT).

If it is desired to repeat the program ten times the sequence:

USER LI REPT SQRT 10 RETURN

accomplishes this function. Alternately one could press:

REPT (USER LI SQRT) 10 RETURN

A.2 Write a subroutine to solve the following system of simultaneous algebraic equations:

$$x + 0.20y + 0.50z = 2.00$$
$$0.20x + y + 0.30z = 1.00$$
$$0.50x + 0.30y + z = 3.00$$

A variety of methods are available for solving sets of algebraic equations. The Gauss-Seidel iteration method will be used for this example. To use this approach the above equations are rearranged by solving for x, y, and z in the first, second and third equations to yield

$$x = 2.00 - 0.20y - 0.50z \quad (1)$$

$$y = 1.00 - 0.20x - 0.30z \quad (2)$$

$$z = 3.00 - 0.50x - 0.30y \quad (3)$$

The Gauss-Seidel method involves assuming values y_0 and z_0 for y and z and solving for x in Equation (1) to yield x_1; x_1 and z_0 are then used in Equation (2) to find y_1; Equation (3) is solved for z_1 using x_1 and y_1. The values y_1 and z_1 are substituted back into Equation (1) to determine x_2 and the iteration is repeated until some convergence criterion is satisfied.

A user program to implement the Gauss-Seidel solution for equations is as follows:

```
LIST LI REAL
    LOAD 2 ⊖ (.2 ⊙ Y) ⊖ (.5 ⊙ Z) STORE X DISPLAY RETURN
    LOAD 1 ⊖ (.2 ⊙ X) ⊖ (.3 ⊙ Z) STORE Y DISPLAY RETURN
    LOAD 3 ⊖ (.5 ⊙ X) ⊖ (.3 ⊙ Y) STORE Z DISPLAY RETURN
    USER LI LIST
STORE USER LI ⊕
```

Assume that $y_0 = z_0 = 1$ are the initial approximations for y and z and that these values have been stored in Y and Z respectively. The results of the first 9 iterations of the above

program are listed below:

Iteration No.	X,Y,Z	Iteration No.	X,Y,Z
1	1.3 +00	6	6.62036 -01
	4.4 -01		7.378 -02
	2.218 +00		2.64685 +00
2	8.03 -01	7	6.6182 -01
	1.74 -01		7.3582 -02
	2.5463+00		2.64701 +00
3	6.9205-01	8	6.61776 -01
	9.77 -02		7.3541 -02
	2.62466		2.64705 +00
4	6.68127-01	9	6.61767 -01
	7.8975 -02		7.3532 -02
	2.64224+00		2.64706 +00
5	6.63083-01		
	7.4711 -02		
	2.64604+00		

As indicated by the outputs, the program has for all intents and purposes converged to a solution after 9 iterations. A convergence test and driving program could be added to this program.

A.3 Use the Newton-Raphson method to obtain one root of the polynomial equation:

$$F(x) = x^4 - x^3 - 8x^2 - 4x - 48 = 0$$

The Newton-Raphson approach employs the iterative equation

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

where

$$f'(x_n) = \frac{df(x)}{dx}\Bigg|_{x = x_n}$$

Since $f'(x_n) = 4x_n^3 - 3x_n^2 - 16x_n - 4$, the previous equation becomes

$$x_{n+1} = x_n - \frac{(x_n^4 - x_n^3 - 8x_n^2 - 4x_n - 48)}{(4x_n^3 - 3x_n^2 - 16x_n - 4)}$$

A user program which will solve the above equation is as follows:

```
LIST LI REAL
    LOAD X ⊖ 1 ⊙ X ⊖ 8 ⊙ X ⊖ 4 ⊙ X ⊖ 48 ⊘
    (X ⊙ 4 ⊖ 3 ⊙ X ⊖ 16 ⊙ X ⊖ 4) NEG ⊕ X
    STORE X DISPLAY RETURN USER LI LIST
STORE USER LI SIN
```

Assume that the initial approximation for x is 5 and store this value under LI REAL X. The program will converge to X = 4.0 after 4 iterations as indicated by the output shown below.

Iteration No.	Value of Root X
1	4.31965 +00
2	4.0454 +00
3	4.00108 +00
4	4. +00
5	4. +00

B. LEVEL I CMPLX EXAMPLES

B.1 Write a user program which will accept a complex number a + ib and convert it to polar form $A\angle\theta$ where $A = \sqrt{a^2 + b^2}$ and θ = arctan (b/a). The program should display both A and θ.

The user program for this example would be as follows:

```
LIST LI CMPLX MOD Q DISPLAY RETURN ARG Q ⊙ 57.2958
DISPLAY RETURN USER LI LIST
STORE USER LI LS
```

The program assumes the number a + ib has been stored in Q. The instruction MOD determines the magnitude A and the ARG instruction determines the angle θ in radians. If the complex number 1 + i were stored in Q the output on the display scope would be

```
1.41421   +00,   0.  +00
4.5       +01,   0.  +00
```

after execution of the program. The first number represents A and the second is the angle θ in degrees.

It is often desirable to store the magnitude and the angle of the polar form of a complex number in a single complex storage location. This can be accomplished by:

```
LIST LI CMPLX ARG Q REFL REAL STORE P CMPLX MOD Q
REAL LOAD P CMPLX STORE Z DISPLAY RETURN USER LI LIST
STORE USER LI RS
```

Here the angle is left in radian measure. For Q = 1 + i the result would be displayed as:

1.41421 +00, 7.85398 -01

Alternatively, one might use the complex natural log operation:

$$\log (a + ib) = \log\sqrt{a^2 + b^2}\ , \arctan (b/a)$$

as follows:

LIST LI CMPLX LOG Q REFL REAL EXP CMPLX REFL STORE Z
DISPLAY RETURN USER LI LIST
STORE USER LI REFL

B.2 Write a user program which will accept the polar form A θ of a complex number, stored as two complex numbers A = A + 0i and W = θ + 0i, and convert it to the form a + ib. The angle θ is to be in degrees. Note that:

$$A\angle\theta = A\, e^{i\theta} = A(\cos\theta + i \sin\theta)$$

The user program to accomplish the conversion is as follows:

LIST LI CMPLX LOAD W ⌀ 57.2958 REFL EXP ⊙ A DISPLAY
RETURN USER LI LIST
STORE USER LI MAX

The program initially converts the angle θ to radians. The instruction REFL interchanges the contents of the (α_I, β_I) register,

so that θ is now in the imaginary portion α_I and the β_I contents are zero. The key EXP forms $e^{i\theta}$ which is then multiplied by A to yield a + ib.

If A $\angle\theta$ were 1.41421 $\angle 45°$, i.e. A = 1.41421,0 and W = 45,0 then a + ib would be computed and displayed as:

9.99997-01, 9.99997-01

on the display scope.

If the polar form is stored as a single complex number Z = A, θ (θ in radians, or readily convertible to radians by REAL ⊘ 57.2958), the program is:

LIST LI CMPLX LOAD Z REFL REAL LOG CMPLX REFL EXP
STORE Q DISPLAY RETURN USER LI LIST
STORE USER LI MOD

Here the magnitude A is replaced by log A before the exponentiation, so the computation is:

$$e^{\log A + i\theta} = A\, e^{i\theta}$$

C. LEVEL II REAL EXAMPLES

C.1 Write a user program which will generate the family of Hermite orthogonal polynomials on the domain $|x| \leq 1$.

We use the recursive relation:

$$H_{n+1}(x) = 2[xH_n(x) - nH_{n-1}(x)]$$

which together with $H_0(x) = 1$ and $H_1(x) = 2x$ completely characterizes the functions.

The program for computing the Hermite polynomials consists of a preparation program stored under USER LII ⊕ , which loads and stores the initial values of n, x, $H_0(x)$ and $H_1(x)$ and then initiates USER LII ⊖ , which constructs and displays $H_{n+1}(x)$, given $H_n(x)$ and $H_{n-1}(x)$.

```
Preparation Program USER LII ⊕

LIST LI REAL ID STORE N LII REAL CTX 101 ID STORE X
⊙ 2 STORE H LOAD 1 STORE J USER LII ⊖ LIST
STORE USER LII ⊕
```

```
Principle Program USER LII ⊖
LIST LII REAL LOAD J ⊙ LI N STORE W LOAD H STORE J
⊙ X ⊖ W ⊙ 2 STORE H DISPLAY RETURN LI REAL LOAD N ⊕ 1
STORE N USER LII LIST STORE USER LII ⊖
```

Note the program USER LII ⊖ also increments n and advances H_n and H_{n-1}, so that, if repeated, it will generate a continuing sequence of the polynomials. The instruction ⊙ LI N means "multiply the entire level II vector by the number stored in level I N." The Hermite polynomials H_2, H_3, and H_4 on the domain $|x| \leq 1$ as obtained from the above program are depicted in Figure E-1. Note that no consideration has been given to scaling of these curves.

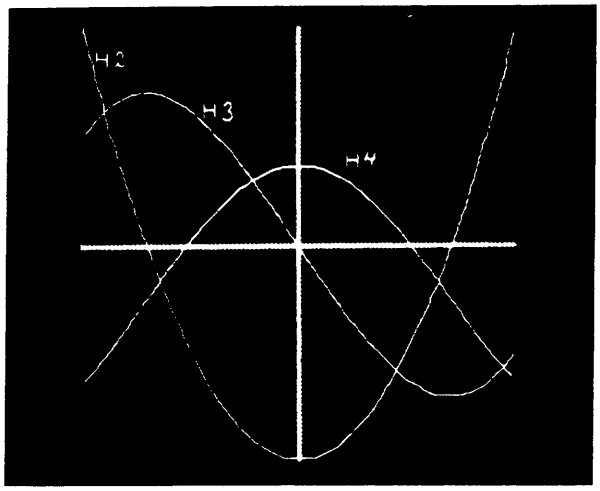


Figure E-1. Hermite polynomials H_2, H_3, and H_4 on the domain $|x| \leq 1$. (Not scaled)

C.2 Write a user program which will plot t sin ωt versus t cos ωt. The program is to crossplot these functions for some initial value of ω, say ω_0. Then ω_0 is to be incremented to a new value ω_1, the ω_0 plot erased, and a new plot using ω_1 displayed. The program is to continue in this fashion until it terminates at $\omega \geq \omega_{max}$.

A user program which accomplishes the above functions, with $\omega_{n+1} = 2\omega_n$ and $\omega_{max} = 300$, is shown below and is assumed to be stored under USER LII REFL. The parameter ω is stored under LI REAL W.

LII REAL ERASE ID STORE T ⊙ LI W STORE P COS ⊙ T STORE C SIN P ⊙ T SUB C DISPLAY RETURN LI LOAD W ⊙ 2 STORE W ⊖ 300 TEST + RS (TYPE RETURN DONE) USER LII REFL

To execute this program with $\omega_0 = 6.28$, push

LI REAL LOAD 6.28 STORE W USER REFL

Figures E-2 and E-3 show the plotted output for $\omega = 12.56$ and 50.24, respectively.

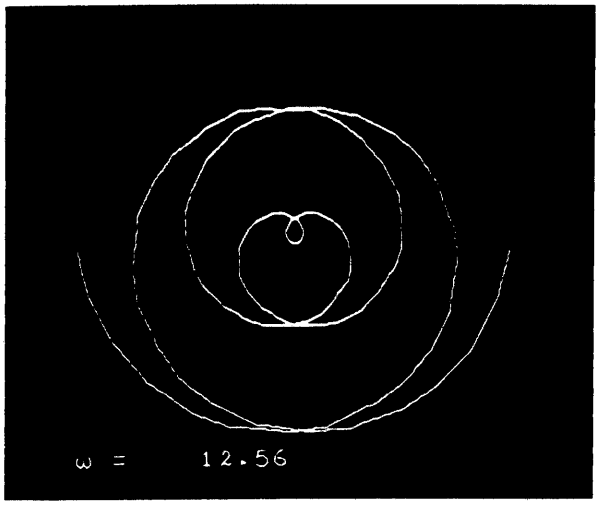


Figure E-2. Display of t sin ωt versus t cos ωt for $\omega = 12.56$

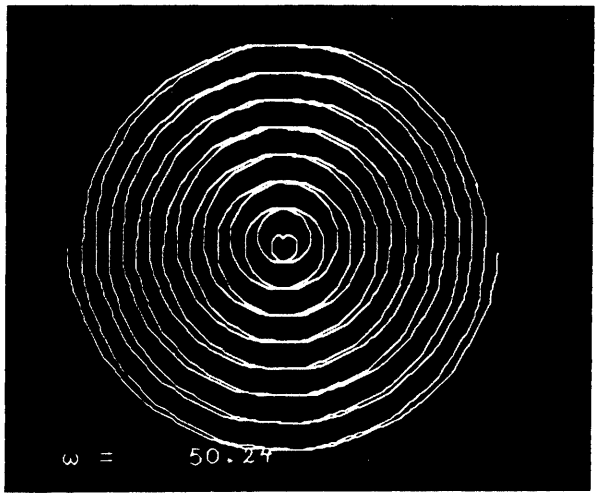


Figure E-3. Display of t sin ωt versus t cos ωt for ω = 50.24

D. LEVEL II COMPLEX EXAMPLES

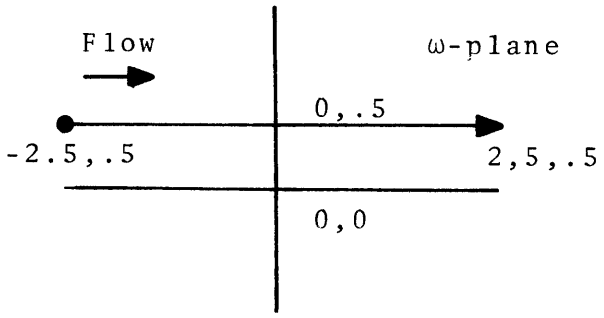
D.1 Write a user program to determine the number of roots of the polynomial $Z^3 + 3Z^2 + Z + 2$ enclosed in the circle $|Z| \leq 2$.

The technique used is to map the circle in the Z-plane to the $Z^3 + 3Z^2 + Z + 2$ plane and note the number of times the transformed figure encloses the origin.

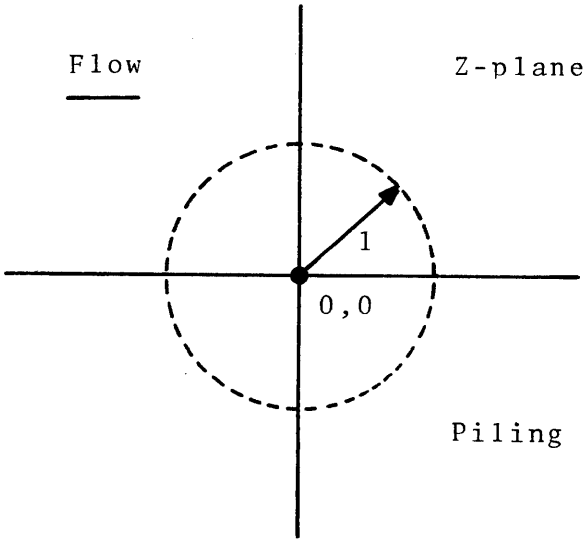
```
LIST LII CMPLX ID . ⊙ 2 STORE Z ⊕ 3 ⊙ Z ⊕ 1 ⊙ Z ⊕ 2
ARG REFL LI REAL LOAD LII (1) STORE B LOAD LII (124)
⊖ B ⊘ 6.28 DISPLAY RETURN LIST
STORE USER LII UP
```

LI REAL LOAD LII (n) will load the n^{th} component of the level II working register β_{II} into the level I working register β_I.

D.2 Given that a particle in a stream flows unobstructed for 5 units as shown below in the ω-plane,



compute the flow of the particle if a cylindrical piling is placed as shown.



The transform is $\omega = Z + \frac{1}{Z}$ or, inversely, $Z = \frac{\omega + \sqrt{\omega^2 - 4}}{2}$ or $Z = \frac{\omega - \sqrt{\omega^2 - 4}}{2}$. To find which of these is applicable, compute both and display them.

LIST LII REAL ID CMPLX ⊙ 2.5 REAL LOAD .5 CMPLX STORE

W SQ ⊖ 4 SQRT STORE R ⊕ W ⊘ 2 STORE Y DISPLAY RETURN

LOAD W ⊖ R ⊘ 2 STORE Z DISPLAY RETURN LIST

STORE USER LII DOWN

Inspection of the curves indicates that the second one is correct. To duplicate Figure E-4, where the unobstructed path (W) and the path Z around the piling are shown to scale, with the piling, constuct a unit circle to represent the piling by:

LII CMPLX ID . STORE P

Determine the scales of W, Z, and P:

LII CMPLX LOAD W DISPLAY 0 RETURN LOAD Z DISPLAY

0 RETURN LOAD P DISPLAY 0 RETURN

They are, respectively, 2, 1, and 0, so all three curves should be displayed to a scale of 2. To accomplish this, push:

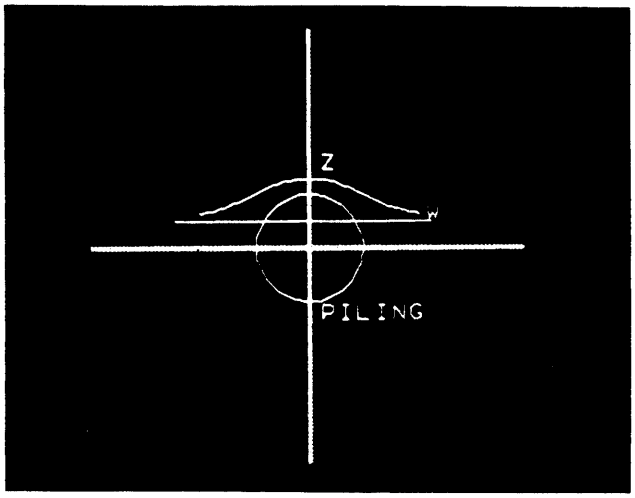
LII CMPLX DISPLAY W LOAD Z CON DISPLAY RETURN

LOAD P CON CON DISPLAY RETURN

Alternately one could press:

LII CMPLX DISPLAY , 2 W Z P RETURN

The 2 may be omitted. If it is omitted MOLSF will compute the maximum display scale possible.



E. INTER-LEVEL AND MISCELLANEOUS OPERATIONS

E.1 This sample program allows for manual creation of level II vectors.

LIST LII REAL CTX PRED 1 ID X LOAD 0
LI REPT (LOAD ENTER DISPLAY RETURN
STORE LII (N)) N=1, PRED 1 LII LIST
STORE USER LI ⊖

To create a vector with three components i.e. (1.6, 0, 29.9), one would manually press:

USER LI ⊖ (3) 1.6 ENTER 0 ENTER 29.9 ENTER

E.2 A simple example using the convolution integral program on LII REAL is the convolution of two rectangular pulses. Assume that the rectangular pulse $f(x) = f(x_1, x_2, \ldots, x_n)$ defined by

$$f(x_{47}, x_{48}, \ldots, x_{51}) = 1$$

$$f(x_1, \ldots, x_{47}, x_{52}, \ldots, x_{101}) = 0$$

is to be convolved with itself. The kernel k(x) for this case is the same as f(x) and the desired weighting in the integration is 1.

Convolution can be thought of as being obtained by translating the kernel k(x) to a particular alignment with f(x) and then determining the area under their product as the folded function is slid along the horizontal axis to the right.

The actual computations effected by the convolution integral program can be easily interpreted in terms of Figures E-5 through E-7. Initially f(x) is generated and displayed by the sequence

LII REAL ID LOAD 0 LI LOAD 1 SUB 47 NEG SUB 52
LII SUM DISPLAY RETURN

and is shown in Figure E-5.

Assume that the function f(x) and kernel k(x) are stored under F and K, respectively. The instructions STORE F K accomplish this operation. The convolution of the two functions is now accomplished by pressing

LOAD F CONV K, 49 DISPLAY RETURN

The first thing the convolution program does is to translate k(x) and align it with respect to f(x). For this example, the 49^{th} component of k(x) is aligned with the first component of f(x), as shown in Figure E-6. The area of the product (zero in this case) is determined and saved in the first component of β_{II}. The function k(x) is now slid to the right, one step at a time; the area of the product is determined at each step, and stored in successive components of β_{II}. Figure E-7 shows the case where k(x) and f(x) are coincident. The area for this case is 5.

At the end of the operation the β_{II} register contains the convolved function, which is depicted in Figure E-8, and is a triangular function with a maximum value of 5 occurring at x_{49}.

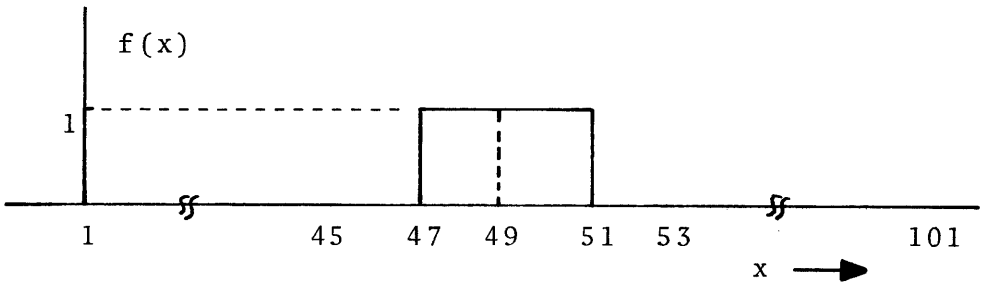


Figure E-5. f(x)

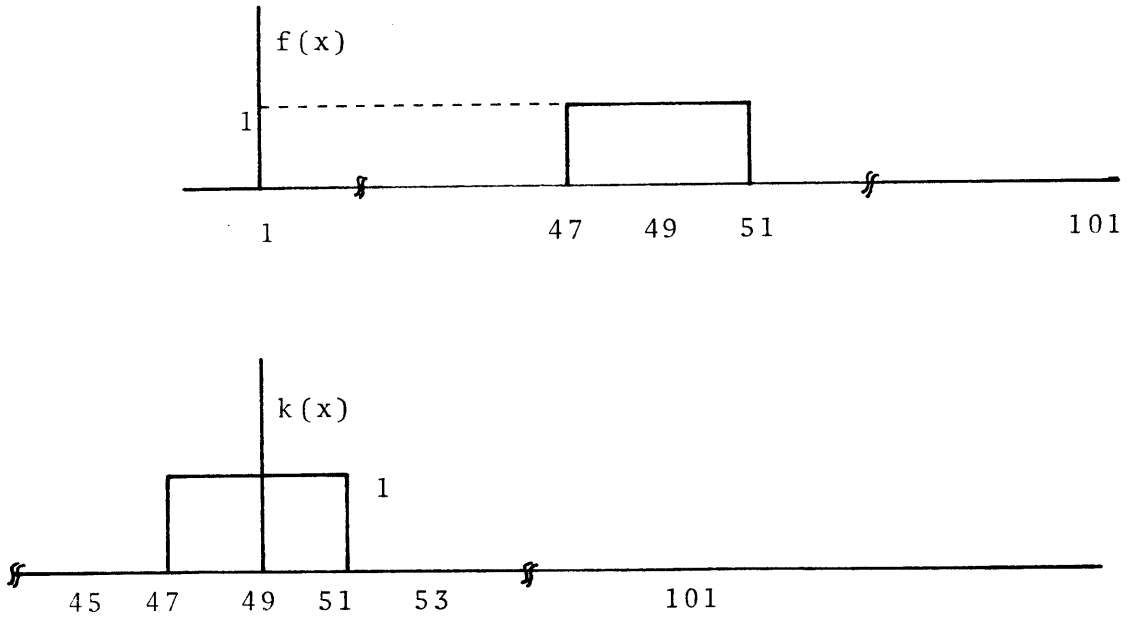


Figure E-6. k(x) translated and aligned with f(x)

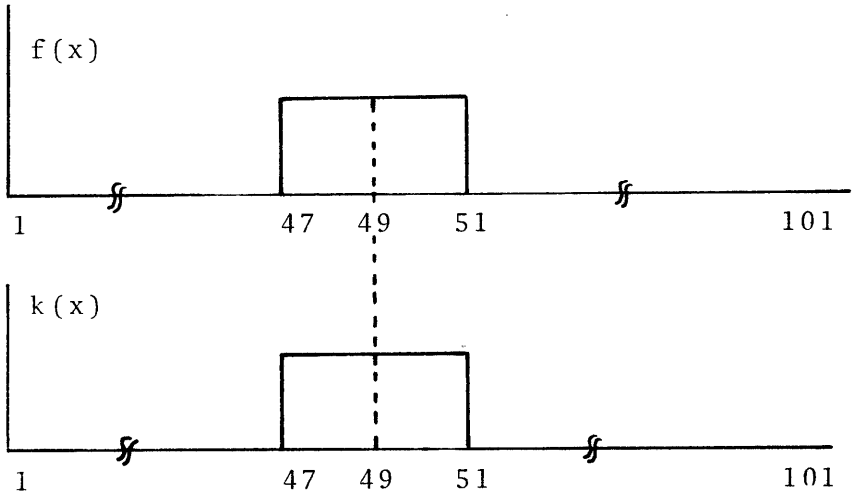


Figure E-8. Convolution integral f(x) * k(x)

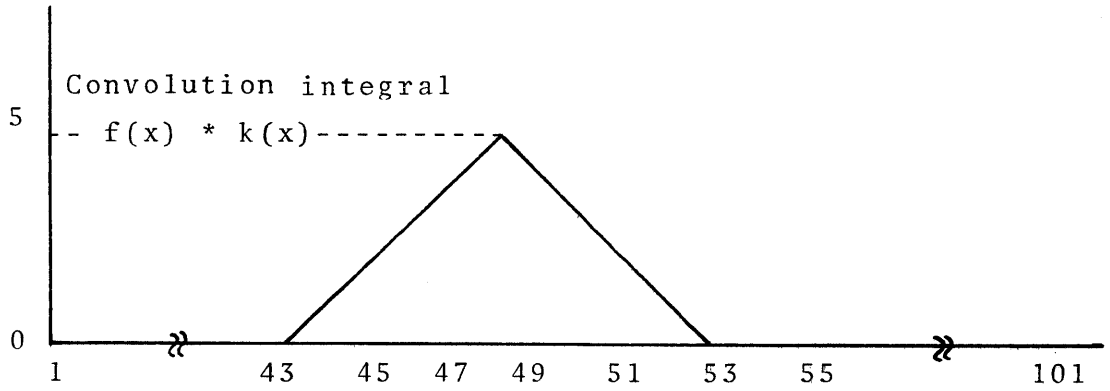


Figure E-8. Convolution integral f(x) * k(x)

E.3 From the data

P(atm)	$\bar{V}$(liters/mole)
50	0.4634
100	0.2386
200	0.1271
300	0.09004

calculate the second virial coefficient, B, for H_2 at 0°C (273°K). The virial equation is

$$\frac{P\bar{V}}{RT} = 1 + \frac{B}{\bar{V}} + \frac{C}{\bar{V}^2} + \ldots$$

so plot

$$\bar{V}\left(\frac{P\bar{V}}{RT} - 1\right) = B + \frac{C}{\bar{V}} + \ldots$$

vs $\frac{1}{\bar{V}}$ and extrapolate to $\frac{1}{\bar{V}} = 0$

On-line solution

First it is necessary to enter the data. This may be done by loading the numbers into the level I β working register and

substituting them into consecutive points of a level II vector of the appropriate context.

```
LII REAL CTX 4
ID X
```

Changing contexts often makes displays look strange - the ID in the sequence simply is there to adjust the system to displays of 4 points from what ever it might have been before.

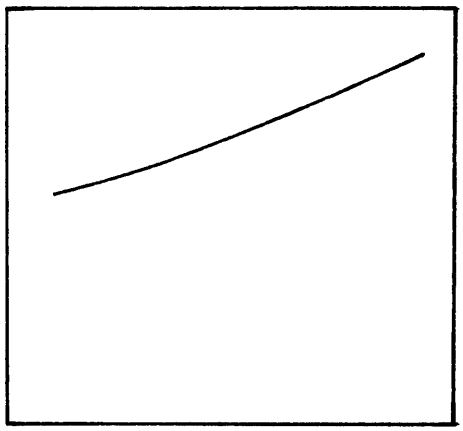
So to load the data for P:

```
LI LOAD 50 SUB 1
LOAD 100 SUB 2 LOAD 200
SUB 3 LOAD 300 SUB 4
```

This sequence takes the level I data as it is loaded into the level I β register and substitutes it into the 1st, 2nd, ... points of the level II β register.

```
LII STORE P
```

stores is away in P. A display of P would look like (roughly)



The same procedure could be followed for $\bar{V}$; however, it's simpler to use sample program E.1 which does essentially the same thing as was done for P except it is more automatic.

USER LI ⊖

calls the program;

(4)

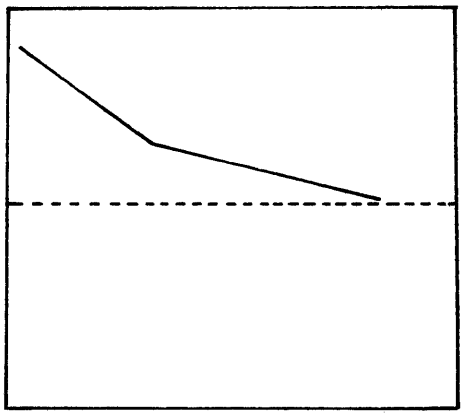
tells it that 4 data points are to be entered;

0.4634 ENTER 0.2386 ENTER

0.1271 ENTER 0.09004 ENTER

STORE V

loads the data and stores it in V. A display of V would look like (again roughly)



now make the function

$$V\left(\frac{PV}{RT} - 1\right)$$

and store it someplace - say F

LII LOAD P ⊙ V ⊘ 0.0821 ⊘ 273

⊖ 1 ⊙ V STORE F

also make $\frac{1}{V}$;

LOAD V INV STORE X

At this point, one can look at all of the functions, i.e. P, V, F and 1/V, but what you really want is to plot F vs 1/V. All displays so far on level II have been relative to an ID (-1 ... +1) vector stored in what is called the level II α register, hence to see F plotted vs 1/V it is necessary to substitute 1/V into the level II α register. To do this you push the keys:

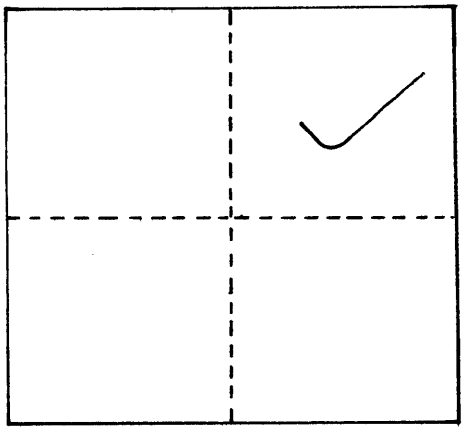
SUB X

SUB X takes what is stored in X and puts it into the level II α register; from now until ID is pushed again, all displays will be relative to the vector stored in X rather than to an ID function.

Now

DISPLAY F

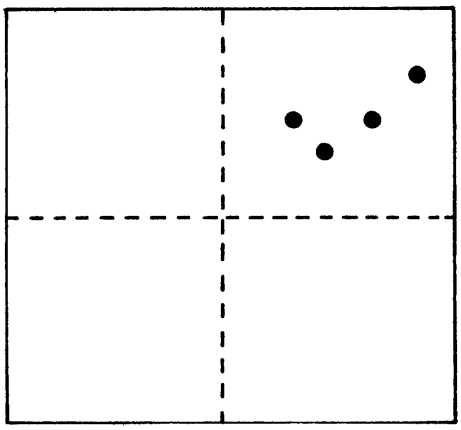
will give you the plot you want, i.e.:



Since 1/V and F are both positive the entire display is in the first quadrant of the screen. A

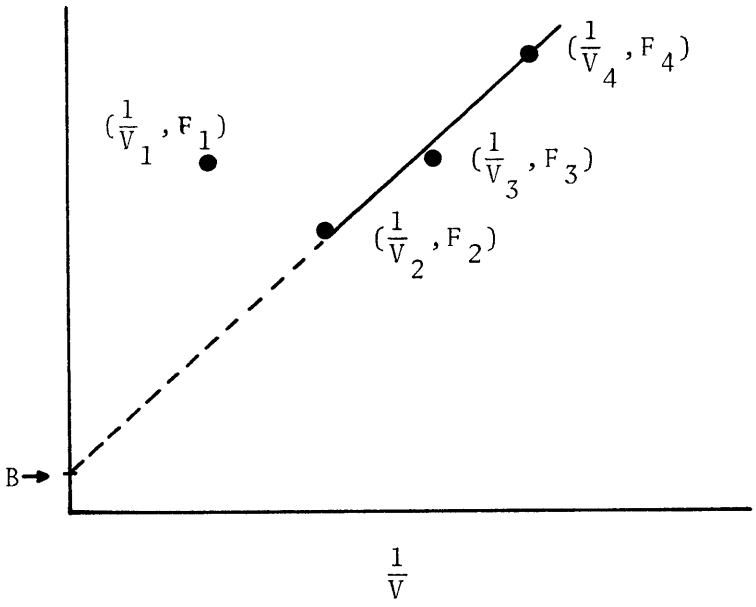
DISPLAY ... RETURN

will give you the points, i.e:



Now seeing that the last three points all lie roughly on a straight line, extrapolate to 1/V = 0 to get B (the intercept) i.e.:

$$F = \bar{V}\left(\frac{P\bar{V}}{RT} - 1\right)$$



There are lots of ways to do this; one way is to use the DIFF operator. DIFF operates on the level II β register to give a vector of first differences, i.e. suppose we say:

DIFF F

What happens is (F_1, F_2, F_3, F_4) becomes $(F_2-F_1, F_3-F_2, F_4-F_3, F_5-F_4)$. Notice that the vector of first differences has an F_5 in it - it is not really F_5 but an extrapolation which is done so that the difference vector will also have 4 points (instead of 3). Anyway store it somewhere say D, i.e:

STORE D

Now load X(= $\frac{1}{V}$) and Diff it, invert, and multiply by D to get

the slope:

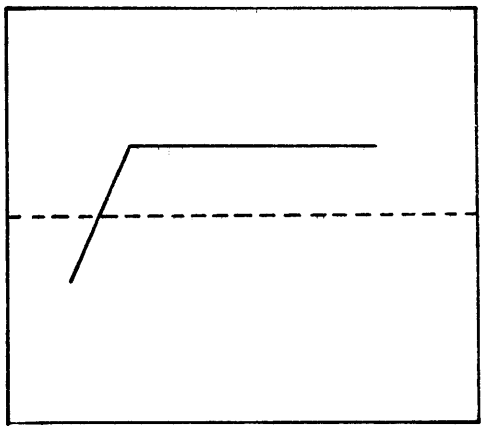
DIFF X INV ⊙ D

STORE S

To get a normal looking display of S push:

ID DISPLAY S

which should look roughly like:



with the last 3 points approximately constant. If we assume that the slope is given by:

$$\frac{(F_4 - F_3)}{(\frac{1}{V_4} - \frac{1}{V_3})}$$

then the third point of S is this number, so:

DISPLAY 3 RETURN

displays its numerical value on the screen. Now lets store it on level I; to do this push:

LI LOAD LI (3) STORE S

which takes the 3rd point out of the level II β register and stores it in the level I storage location S (note that S on level II and S on level I are completely independent of each other).

Now finally to get B calculate

$$[\bar{V}(\frac{P\bar{V}}{RT} - 1) - \frac{\text{slope}}{\bar{V}}] = B$$

where the slope = C in our original equation. So

LOAD X ⊙ LI S NEG ⊕

F DISPLAY 3 RETURN

gives the value of B which is what we were looking for in the first place.

A few comments:

The operation ... ⊙ LI S ... is

1) "multiply the entire level II vector by the number stored in level I S"

2) ... LI LOAD LII (3) ...
is the exact opposite of the level I SUB operator used to get the data in; that is, it takes a single number out of the level II β register and copies it into the level I β register.

E.4 Calculate the work involved in expanding 1 mole of SO_2 isomthermally and reversibly from a volume of 2.46 liters to 24.6 liters at 27°C (300°K) using

a) ideal gas law; PV = RT (for one mole)

b) Van der Waals' equation:

$$(P + \frac{a}{V^2})(V - b) = RT$$

where a = 6.714 and b = 0.05636 for SO_2

The work is given by

$$W = \int_{V_1}^{V_2} p\,dv$$

now since it is an isothermal process, so for part a) P = RT/V hence

$$W = RT \int_{V_1}^{V_2} \frac{dv}{v}$$

$$\boxed{W = RT \ln \frac{V_2}{V_1}}$$

where R = 1.987 calories/deg mole. First lets do it on level I

LI REAL LOAD 24.6

⊘ 2.46 LOG ⊙ 300 ⊙

1.987 DISPLAY RETURN

The answer should be 1372.56 calories. However, lets also do the problem by carrying out directly the integration. First go to level II and set up V (with a range of 24.60-2.46 = 22.14)

LII REAL CTX 124 RETURN ID ⊕ 1 ⊘ 2

⊙ 22.14 ⊕ 2.46 STORE V

We will assume an integration routine (LII REAL STORE I

DIFF ⊘ -6 ⊕ I DIFF ⊘ 2 ⊕ I ⊙ D RS LI LOAD 0 SUB 1 LII SUM)

has been stored under USER LIII SUM - part of its usage is that we are required to store in D a vector representing the differential of the independent variable; i.e. in this case ΔV. So (since V is still in the β register)

DIFF STORE D

now make P = RT/V

LOAD 1.987 ⊙ 300 ⊘ V

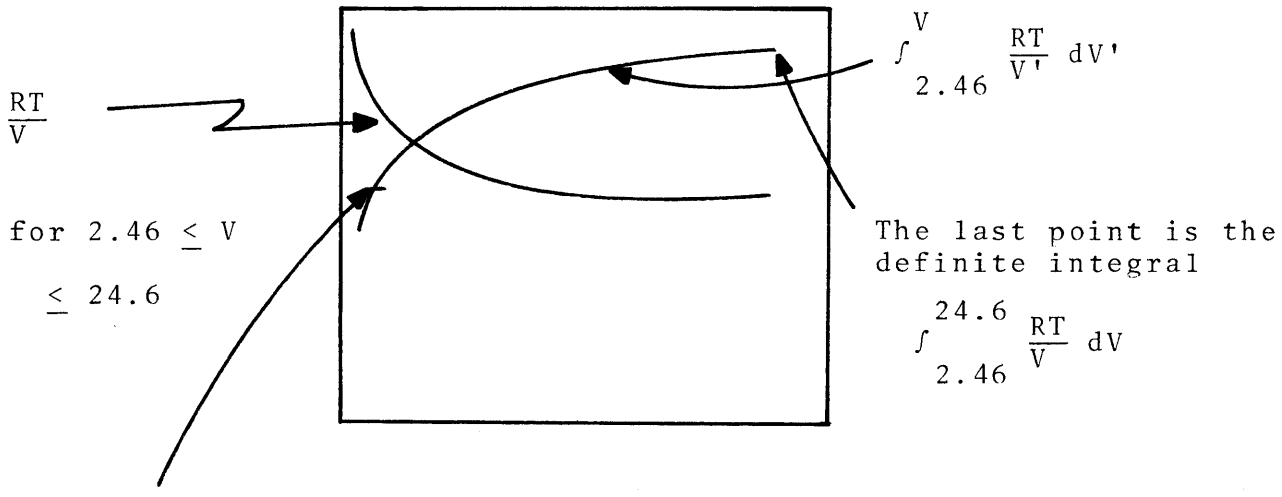
and integrate it

USER LIII SUM

and display it

DISPLAY RETURN

which will look like



This point is zero as it should be since it is

$$\int_{2.46}^{2.46} \frac{RT}{V} \equiv 0$$

DISPLAY 124 RETURN

displays the value of the integral $\int_{2.46}^{24.6} PdV$. From this display one could say a number of things about the nature of the work one gets in expanding a gas, for example.

Most of the work is obtained early in the process - if we look at the integrated equation (solved earlier)

$$W = RT \ln \frac{V_2}{V_1}$$

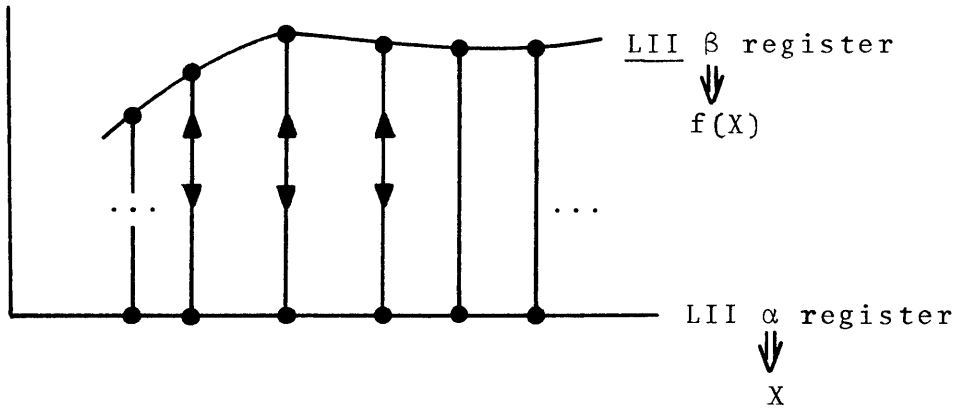
we see that one gets RT ln 2 each time the volume doubles during the expansion - so in going from:

2.46 l → 4.92 l ⇒ RT ln 2 worth of work

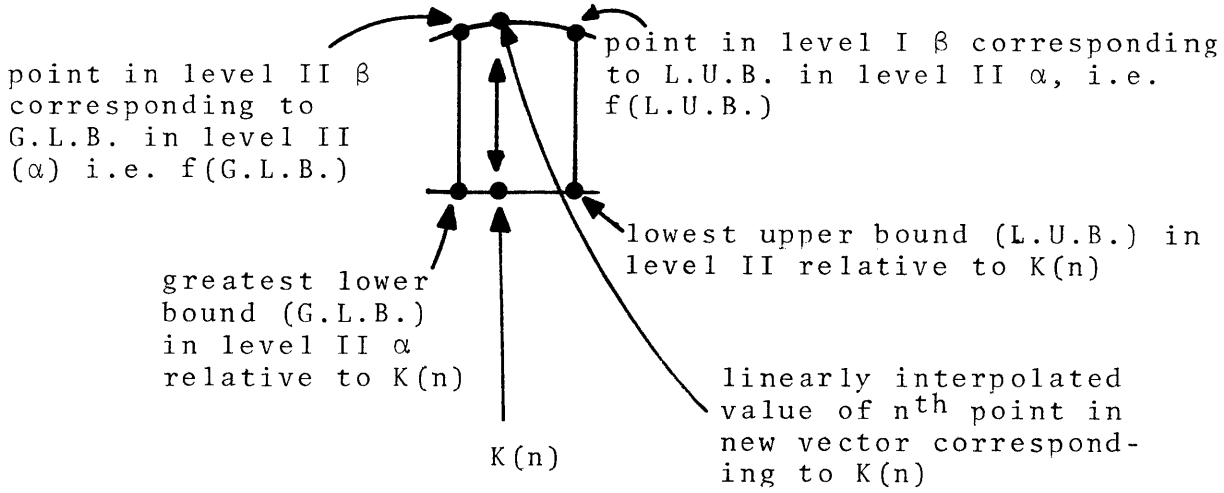
4.92 l → 9.84 l ⇒ RT ln 2 worth of work more

9.84 l → 19.68 l ⇒ RT ln 2 worth of work more

Lets store the integral representing the work and see if this is true by using the EVAL operator. First, however, a description of EVAL; suppose you have one vector stored in the level II β register and another in the level II α register. This then defines a function, i.e. a one-to one correspondence between 2 sets of numbers, schematically we have



Now suppose you have a third vector stored somewhere else - say under K. Pushing EVAL K then does the following: it takes the first point of K and searches the level II α register until it finds a number which is equal to K(1); it then takes the corresponding point in the level II β register and stores it as the first point of a new vector; it goes to the second point of K, searches the level II α for its equal, takes the corresponding point of the level II β and stores it as the 2nd point of the new vector and so on until it runs all the way through K. Generally, however, it won't find an exact equivalence between the points of K and the level II α so it interpolates, i.e.



When it is all done the level II α contains K, and the level II β register has the new vector made according to the process just described. Given X and f(**x**), we take a K and make f(K) with f(K) having the same functional relationship to K that f(x) had to X. Anyway it is a nice thing which has far more powerful uses then what we will do with it here.

Okay - so back to finding the work corresponding to the incremental doubling of the volume - we have a V and f(V) = work for the range of $2.46 \leq V \leq 24.6$ hence we can take a V = 4.96 and find f(4.96) etc. First store the integral = work = f(V) in W and proceed:

```
STORE W LOAD 4.96
STORE K LOAD W SUB
V EVAL K DISPLAY
1 RETURN
```

which will give the work corresponding to the expansion from 2.46 → 4.96 liters:

```
LOAD 9.84 STORE K
LOAD W SUB V
EVAL K DISPLAY 1 RETURN
```

which will give the work corresponding to the expansion from 2.46 → 9.84 which should be just double the work from 2.46 → 4.96 -- anyway continue for the others in the same way.

The second part of the problem was to do the calculation for the Van der Waals' equation. The integration can be done in closed form, however, we will proceed with the numerical integration.

First evaluate $P = RT/(V-b) - a/V^2$. Then integrate it and display the last point:

SQ V INV ⊙

6.714 NEG STORE A LOAD

V ⊖ 0.05636

INV ⊙ 1.987 ⊙ 300 ⊕ A

This evaluates P:

USER LIII SUM

This integrates it:

DISPLAY RETURN 124 RETURN

and this displays the whole integral and the last point on it.

Appendix F

FORTRAN SUBROUTINE CALLS FOR TRANSFER OF LII VECTORS TO AND FROM AN ONLINE TERMINAL

The following subroutines are available to a FORTRAN program (G or H level).

(1) Fetch data from an online terminal:

Calling sequence: CALL FOLS(arg_1,arg_2,arg_3)

Arguments:

arg_1:	A positive integer constant or variable indicating the number of components to be fetched from the online terminal.
arg_2:	A real scalar variable, a real array name, or an element denoting the receiving field for the transfer.
arg_3:	An optional integer variable which will contain the contents of the LO quotient register.

(2) Transfer data to an online terminal:

Calling sequence: CALL TOLS (arg_1, arg_2, arg_3)

Arguments:

arg_1:	A positive integer, constant or variable indicating the number of components to be transferred to the online terminal.
arg_2:	A real scalar variable, a real array name, or an element denoting the source field for the transfer.
arg_3:	An optional integer constant or variable specifying the new contents of the LO quotient register.

Appendix G

REFERENCES RELATED TO ON-LINE SYSTEM APPLICATIONS

This appendix contains a list of references concerning various applications and problems which have been solved using earlier forms of the system described in this manual. Simply referring to this appendix will illustrate something of the variety of capabilities available with the system.

Bruch, J. C., Jr: "Free Streamline Theory and Computer Generated Displays." Article submitted for review and publication in the ASCE Journal of Engineering Mechanics Division, August, 1970.

_____, "Hydrodynamics Through On-Line Computer Generated Displays." Handbook, College of Engineering, University of California, Santa Barbara. (Xeroxed.)

_____, "Two Dimensional Flow Visualization Using Computer Generated Displays." Article submitted for review and publication in the ASCE Journal of the Hydraulics Division, August, 1970. 16mm educational/demonstration movie, Learning Resources Motion Picture Production Section, University of California, Santa Barbara, California, 1970.

Bruch, J. C., Jr. and J. A. Howard: "Flow Visualization in Hydrodynamics Using On-Line Generated Displays." Paper presented at the Pacific Southwest ASEE Annual Meeting, University of California, Davis, December 29-30, 1969. 16mm movie, Learning Resources Motion Picture Production Section, University of California, Santa Barbara, California, 1969.

Bruch, J. C.,Jr. and R. C. Wood: "The Teaching of Hydrodynamics Using Computer Generated Displays." Bull. Mech. Engng. Educ., Vol. 9 (May, 1970). 105-115.

Economics Laboratory (Innovative Project in University Instruction), University of California, Santa Barbara: "Economics Data Bank Handbook." November, 1969.

_____, "Economic Statistics: Handbook for the On-Line System." December, 1969.

_____, "Macro-and-Micro Economic Models." Handbook, October, 1969.

_____, "Purpose and Functions." Report, September, 1969.

_____, "Sample Instructional Materials." November, 1969.

_____, "Teleputer Handbook, Revised Edition." December, 1969.

Bullock, D. L.: "Exchange Ratios in CuF_2 $2H_2O$." TRW/STL Report 9891-6001-RU-000, April 1965.

Cheng, Hung, and David Sharp: "Formulation and Numerical Solution of Sets of Dynamical Equations for Regge Pole Parameters." Phys. Rev. Letters, Vol. 132, p. 1854, 1963.

Collins, L: "The Calculation of Unpaired Electron Density on the Nucleus of Many-Electrons with a Thomas-Fermi-Dirac Potential." Ph. D. Dissertation Dept. of Physics, Univ. of Calif., Santa Barbara, Calif., October, 1966.

Cooperstein, B. D: "OGO-F Electromagnetic Compatibility Analysis (Final Report)." TRW IOC 8212.1-021, January 26, 1968.

Cooperstein, B. D. and W. R. Johnson: "1967 Plant Methods and Processes Studies in: Electrical Bonding and Computer Applications." TRW Report No. 8212-94, November 1967.

Corwin, C. W.: "Computer Aided Design of Torsion Wire Suspension for a Satellite Stabilization Boom." TRW IOC 66-3340.6-28, August 12, 1966.

Coward, D. J.: "On-Line Computer Test Cases for Computer Aided Design: Structural Design of Cylindrical Shells." TRW IOC 65-9715.9-110, Aug. 24, 1965; "Design of Helical Springs." TRW IOC 65-9715.4-104, Dec. 3, 1965; "Flexure Design Problem." TRW IOC 65-9715.6-22, Dec. 9, 1965; "Structural Design of a Beam with Varying Loads." TRW IOC 66-9713.1-14, Jan. 12, 1966.

Culler, G. J., B. D. Fried, R. W. Huff, and J. R. Schrieffer: "Solution of the Gap Equation for a Superconductor." Phys. Rev. Letters, Vol. 8, p. 399, 1962.

Culler, G. J. and R. W. Huff: "Solution of Nonlinear Integral Equations Using On-Line Computer Control." Proc. AFIPS Spring Joint Computer Conference, National Press, Palo Alto, Calif., Vol. 29, p. 126, 1962.

Deland, R. W.: "On-Line Computer Program Gas Pressure Mass Accelerator." TRW IOC 66-4722.5-2, August 9, 1966.

DeNuzzo, J.: "On-Line Solution of 2-D Trajectory Equations." TRW/STL Report 9801-6013-TU-000, May 1965.

Dixon, W. J.: "ΔV to Enter Orbit About Mars." TRW IOC VM-2, April 1965.

Emmerling, R. C.: "On-Line Computer Programs: Toroidal Diaphragm Analysis." TRW IOC 66-3520.4-26, September 27, 1966.

Ewig, C. S., J. T. Gerig, and D. O. Harris: "An Interactive On-Line Computing System and its Use in Chemistry Education." Department of Chemistry, University of California, Santa Barbara. (Xeroxed)

Field, E. C., and B. D. Fried: "Solution of Kinetic Equation for an Unstable Plasma in an Electric Field." Phys. Fluids, Vol. 7, p. 1937, 1964.

Fried, B. D.: "On-Line Root Finding in the Complex Plane." TRW Report No. 9990-7308-RU-000, July 1966.

Fried, B. D.: "Solving Mathematical Problems." Chapter VI of On-Line Computing, edited by W. J. Karplus, pp. 131-178, McGraw-Hill, 1967

Fried, B. D., and A. Y. Wong: "Stability Limits for Longitudinal Waves in Ion Beam-Plasma Interaction." Phys. Fluids, Vol. 9, p. 1084, 1966.

Fried, B. D. and C. L. Hedrick: "Two-Pole Approximation for the Plasma Dispersion Function." Phys. Fluids, Vol. 11, p. 249, 1968

Fried B. D., and L. O. Heflinger: "Scaling Law for MHD Acceleration." TRW Systems Report 9801-6014-RU-000, July 1965.

Fried, B. D., and G. J. Culler: " Plasma Oscillations in an External Electric Field." Phys. Fluids, Vol. 6, p. 1128, 1963.

Fried, B. D., and S. L. Ossakow: "The Kinetic Equation for an Unstable Plasma in Parallel Electric and Magnetic Fields." Phys. Fluids, Vol. 9, p. 2428, 1966.

Heflinger, L.: "On-Line Console Program of General Interest: Generation of Random Numbers." TRW Report No. 9863-6004-RU-000, Feb. 1, 1966.

Howard, J. A. and R. C. Wood: "Computer-Assisted Instruction in Engineering Using On-Line Computation." J. Of Engineering Education (in press).

Hrzina, J.: "Application of Mathematical Optimization Theory in Structural Design." AIAA Paper No. 68-174, New York, New York, January 22-24, 1968.

Hrzina, J.: "The TRW On-Line Computer as a Labor-Saving Device for the Expansion of Functions in Fourier Series." EM18-2, TRW/TR 99900-6544-T000, March 11, 1968.

Hrzina, Joseph: "Structural Optimization - A Utilization of the TRW On-Line Computer." EM 17016, TRW/TR 99900-6332-R000, September 15, 1967.

Hutton, R. E.: "An Investigation of Soil Erosion and Diffused Gas Blow-off Caused by the Surveyor Vernier Engine." Prepared for Jet Propulsion Laboratory, California Institute of Technology, Pa adena, Calif., by TRW, December 1, 1966.

Hutton, R. E.: "An Investigation of Soil Erosion and Its Potential Hazard to LM Lunar Landing." NASA MSC Project Technical Report 05952-6056-R000, Task ASPA 47, (TRW EM 17-11),May 1967.

Hutton, R. E.: "An Investigation of Soil Erosion During LM Lunar Landing." NASA MSC Project Technical Report 05952-H210-RO-00, Task ASPA 47, (TRW EM 17-11), May 1967.

Ishimoto, T.: "FACT-CAD Program." TRW IOC 68-3346.6-10, May 20, 1968.

Johnson, Kenneth, and Marshall Baker: "Quantum Electrodynamics." Phys. Letters, Vol. 11, p. 518, 1963.

King, J. R.: "Computer Aided Design [Diffusion Bonding and Effective Stress vs. Strain for an Orthotropic Material]." TRW IOC 4812-94, December 9, 1966.

Kinsey, P., C. S. Ewig, and D. O. Harris: "Introduction to the UCSB On-Line Computing System." Department of Chemistry University of California, Santa Barbara, 1970.

Margulies, R. S.: "Response of a Peak-Reading Instrument to a Contaminated Signal." TRW/STL Report 9990-6963-TU-000, June, 1965.

McCune, J.: "Exact Inversion of Dispersion Relations." Phys. Fluids, Vol. 9, pp. 2082-84, 1966.

Newhall, D. H.: "Monte Carlo Simulation of a Fixed Attitude Orbit Control Scheme." TRW IOC 9883.6-68, July 15, 1965.

Nishinago, R. G.: "Preliminary Design Considerations for a Gyro-damped Gravity Gradient Satellite." TRW/STL Report 8427-6005-RU-000, May 1965.

Pate, N. C., and S. N. Zivi: "An Analysis of the Efficiency of Elliots Liquid Metal MHD Energy Conversion Cycle and Its Applicability to the Power Range of 3 to 30 KWE." TRW Systems Report 9806-6002-MU-000, July 1965.

Rampton, C. C.: "On-Line Computer Program for Computing Buckling Loads on Pin Ended Columns with Variable Moment of Inertia." TRW IOC 66-3342.1-155, September 26, 1966.

Ridgway, R. I.: "Phase-Lock Loop Analysis-Interim Report." TRW IOC 7323.2-88, August 30, 1966.

Sandusky, A.: "Handbook for the Computer Laboratory: Analysis of Data in Psychology." Department of Psychology, University of California, Santa Barbara, July 1, 1970.

_____,: "Undergraduate Researcher's Manual: Analysis of Data in Psychology." Department of Psychology, University of California, Santa Barbara.

Schreiner, R. N.: "On-Line Computer Linear Grid Routine." TRW IOC 66-3340.6-21, November 4, 1966.

Schreiner, R. N.: "On-Line Computer Log-Log Routine." TRW IOC 66-3340.5-19, October 25, 1966.

Schreiner, R. N.: "On-Line Computer Program: Shock Spectrum Analysis." TRW IOC 66-3340.6-26, December 28, 1966.

Schreiner, R. N.: "On-Line Computer Routine: Bessel Functions $J_n(x)$ and $Y_n(x)$." TRW IOC 66-3340.6-24, November 29, 1966.

*Schreiner, R. N.: "TRW and Computer Aided Design: The Formative Years - 1965-1967." TRW Systems Report No. 67-3340.6-5, Feb. 6, 1967.

Schrieffer, J. R., D. J. Scalapino, and J. W. Wilkins: "Effective Tunneling Density of States in Superconductors." Phys. Rev. Letters, Vol. 10, p. 336, 1963.

Sullivan, J. J.: "Computer Based Instruction in Economics: A Report on Facilities and Applications at UCSB." Paper presented at a conference on Computers in Undergraduate Curricula, University of Iowa, Iowa City, Iowa, 16, 17, 18, 1970.

_____, : "The Ecomomics Laboratory at UCSB." Simulation and Games: an International Journal of Theory, Design, and Research, Vol. 1 No. 1 (March 1970), 81-91.

*von Waldburg, A. R.: "TRW On-Line ERS Balance Program G 5468800." TRW IOC 68-3343.2-104, May 21, 1968.

---

*Schreiner's report is concerned with TRW's computer aided design activities utilizing an on-line system similar to that described in this manual. The report contains a list of TRW reports (including abstracts) related to this computer-aided design effort.

*On-line program developed as a tool to assist in balancing satellites by addition or deletion of suitably distributed balancing weights.

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 tp Five Simultaneous Equations." TRW IOC 66-3343.3-197, December 6, 1966.