Digital Equipment Corporation
Maynard, Massachusetts

digital

PDP-8 Family
Programming Manual

FOCAL-8

FOCAL-8
PROGRAMMING MANUAL
FOR PDP-8/I, PDP-8/L, PDP-8/S, PDP-8, LINC-8, PDP-12, PDP-5

For additional copies, order No. DEC-08-AJAD-D from Program Library, Digital Equipment Corporation, Maynard, Massachusetts 01754. Price $2.00

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

1st Printing April 1968
2nd Printing (Rev) October 1968
3rd Printing December 1968
4th Printing April 1969
5th Printing (Rev) July 1969
6th Printing (Rev) January 1970

Your attention is invited to the last two pages of this manual. The Reader's Comments page, when filled in and returned, is beneficial to both you and DEC. All comments received are considered when documenting subsequent manuals, and when assistance is required, a knowledgeable DEC representative will contact you. The Software Information page offers you a means of keeping up-to-date with DEC's software.

CONTENTS

CONTENTS (Cont)

CONTENTS (Cont)

CONTENTS (Cont)

ILLUSTRATIONS

TABLES

PREFACE

This manual is intended for anyone desiring to learn and use FOCAL-8. Nothing is assumed; the manual is self-sufficient for the mastery of FOCAL-8.

FOCAL-8 (to be referred to simply as FOCAL, for FOrmula CALculator) is DEC's service program designed for solving numerical problems of any complexity using the family of 8 computers. FOCAL's simple, easy-to-learn language can be mastered with the use of this manual in a matter of a few hours.

The first three chapters are devoted to explaining how to formulate and manipulate the FOCAL language. Each of the commands that form the FOCAL language is explained in detail. After covering the first three chapters, the user can realize the power and flexibility of FOCAL by duplicating and running the demonstration programs using different variables.

Chapter 4 contains various demonstration programs which reveal many of the features and applications of FOCAL. A wide range of applications is included in this chapter, including business, scientific, educational, and algebraic uses.

Chapter 5 is devoted to expanded FOCAL capabilities. After the user has gained skill with the fundamental FOCAL system, such power as increased accuracy, longer programs, and even graphic display may be added. In addition, FOCAL may be shared by more than one user on a single computer. Each user has his own Teletype® and uses FOCAL as if he had the computer all to himself. Four and seven user systems, called QUAD and LIBRA, respectively, are available.

Complete loading procedures, a command and format summary, error messages, and programming hints for the more sophisticated user are described in the appendices.

® Teletype is the registered trademark of Teletype Corporation.

CHAPTER 1
INTRODUCTION

FOCAL (FOrmula CALculator) is a standard service language for the PDP-8 family of computers, designed to help students, engineers, and scientists solve numerical problems.

FOCAL programming is very easy to learn. The language consists of imperative English commands and mathematical expressions, which are typed primarily in standard notation. Because the language is easy to learn, it is widely used by mathematics teachers in both secondary schools and colleges as an integral part of their courses.

This manual provides several alternative approaches to learning FOCAL, depending on individual experience. If you have never programmed a computer, your teacher or an experienced operator can load the FOCAL program; then, you can reply to the initial dialogue and begin studying with Chapter 2. The best way for you to learn the FOCAL language is to sit at the Teletype console and try the commands, starting with the examples in this manual. FOCAL will locate any format errors and note them with an error code as they are encountered.

If you are an experienced FORTRAN programmer, you can begin with the command summary and error messages in the back of the book. FOCAL commands are defined in Chapter 3, and Chapter 4 contains sample programs which range from fairly easy to complex.

Chapter 5 describes advanced FOCAL systems. For example, four users can share FOCAL by adding the appropriate hardware and QUAD, the four-user system program. Only one computer is used to serve all users. Each user has all the power of FOCAL at his disposal. In addition, a program called LIBRA permits seven users to time-share a FOCAL system. FOCAL allots computer time to the users, and rarely is there any noticeable delay in execution of a command.

LIBRA features library capabilities which serve to store programs for later use, thereby avoiding the retyping of the entire routine. An initialization segment, DISKIN, serves to prepare the storage area used by LIBRA. Both the QUAD and LIBRA multi-user systems are furnished on paper tape and are easily added to the system.

For the more sophisticated user, the powers of FOCAL can be expanded by incorporating some of the additional system segments. These capabilities include:

a. 4WORD, which provides increased calculating accuracy

b. 8K, to give the user more memory space for large programs

c. CLINE, a series of graphic display functions. CLINE has two segments to be used with the CLINE functions: PLOTR for an incremental plotter and GRAPH for a KV8 display system.

These segments can be used with FOCAL in almost any combination to augment its capabilities. For example, a lengthy, complex, or repetitive mathematical routine can be executed and answers calculated to 10 digits, instead of the standard 6 digits, by adding the 4WORD and 8K overlays to the FOCAL System. The additional segments are furnished on separate punched tapes (overlays) and merely have to be read into the system, in the same simple manner as FOCAL, to add their flexibility to the regular FOCAL interpreter. (Refer to Appendix E.)

1.1 EQUIPMENT REQUIREMENTS

FOCAL operates on a 4K PDP-8/I, -8/L, -8/S, -8, -12, -5, or LINC-8 Computer with a 33 ASR Teletype. Optional equipment includes an analog-to-digital converter and an oscilloscope display. Extra core and device requirements for multi-user and additional FOCAL segments are included in Chapter 5.

Loading and operating procedures for all FOCAL configurations, including paper-tape and Disk Monitor Systems, are described in Appendix E.

1.2 THE INITIAL DIALOGUE

After FOCAL has been loaded and started, FOCAL identifies itself and the type of computer being used; FOCAL then types the options available to the user for retention of the mathematical functions. If these functions are not needed, the user answers FOCAL's questions by typing NO and the RETURN key, and FOCAL erases those functions from core; thus, the user gains additional core storage for use by his programs.

Alternate initial dialogues are shown below.

```
CONGRATULATIONS!!
YOU HAVE SUCCESSFULLY LOADED 'FOCAL,1969' ON A PDP-8 COMPUTER.

SHALL I RETAIN LOG, EXP, ATN ?:YES

PROCEED.

*
```

When the user answers YES to the above question, all mathematical functions are retained, and the user has approximately 700_{10} locations available for his programs.

```
CONGRATULATIONS!!
YOU HAVE SUCCESSFULLY LOADED 'FOCAL,1969' ON A PDP-8 COMPUTER.

SHALL I RETAIN LOG, EXP, ATN ?:NO

SHALL I RETAIN SINE, COSINE ?:YES

PROCEED.

*
```

If the user answers NO to the first question, FOCAL asks a second question. A YES answer to the second question leaves approximately 900_{10} locations available for the user's programs.

```
CONGRATULATIONS!!
YOU HAVE SUCCESSFULLY LOADED 'FOCAL,1969' ON A PDP-8 COMPUTER.

SHALL I RETAIN LOG, EXP, ATN ?:NO

SHALL I RETAIN SINE, COSINE ?:NO

PROCEED.

*
```

A NO answer to the second question erases all extended functions from core, giving the user 1100_{10} locations for use with his programs. To determine the exact number of locations available, use the LOCATIONS command (refer to Appendix C).

Note that logarithm, arctangent and exponential functions cannot be retained without the sine and cosine. Refer to DEC-08-AJBB-DL, Advanced FOCAL Technical Specifications for another way to eliminate the extended functions. After the initial dialogue has been answered, FOCAL automatically erases it from core. FOCAL concludes the initial dialogue by telling the user to PROCEED followed by an * and waits for user input.

CHAPTER 2
FOCAL LANGUAGE

After the initial dialogue has been concluded, FOCAL types an asterisk (*), indicating that the program is ready to accept commands from the user. Each time the user completes typing a Teletype line, which is terminated by depressing the RETURN key, or after FOCAL has performed a command, an asterisk is typed to tell the user that FOCAL is ready for another command.

2.1 ELEMENTARY COMMANDS

One of the most useful commands in the FOCAL language is TYPE. To FOCAL this means "type out the result of the following expression." If you type (following the asterisk which FOCAL typed),

```
*TYPE 6.4318+8.1346
```

and then press the RETURN key, FOCAL types

```
 = 14.5664*
```

Another useful command is SET, which tells FOCAL "store this symbol and its numerical value. When I use this symbol in an expression, insert the numerical value." Thus, the user may type,

```
*SET A=3.14159; SET B=428.77; SET C=2.71828
*
```

The user may now use these symbols to identify the values defined in the SET command. Symbols may consist of one or two alphanumeric characters. The first character must be a letter, but must not be the letter F.

```
*TYPE A+B+C
=   434.6300*
```

Both the TYPE and SET commands will be explained more fully in the next chapter.

FOCAL is always checking user input for invalid commands, illegal formats, and many other kinds of errors, and types an error message indicating the type of error detected. In the example,

```
*HELP
?03.28
*TYPE 2++4
?07.;6
*
```

HELP is not a valid command and two plus signs (double operators) are illegal. The complete list of error messages and their meanings is given in Appendix B.

2.2 OUTPUT FORMAT

The FOCAL program is originally set to produce results showing up to eight digits, four to the left of the decimal point (the integer part) and four to the right of the decimal point (the fractional part). Leading zeros are suppressed, and spaces are shown instead. Trailing zeros are included in the output, as shown in the examples below.

```
*SET A=77.77; SET B=1111.1111; SET C=39
*TYPE A,B,C
=    77.7700=  1111.1100=    39.0000*
```

The results are calculated to six significant digits. Even though a result may show more than six digits, only six are significant, as shown above in SET B = 1111.1111, which FOCAL typed as = 1111.1100. (See 4WORD, Chapter 5, for increased accuracy.)

The output format may be changed if the user types

```
TYPE %x.yz,
```

where x is the total number of digits to be output and yz is the number of digits to the right of the decimal point. x and yz are positive integers, and x cannot exceed 19 digits. When first loaded, FOCAL is set to produce output having eight digits, with four of these to the right of the decimal point

(%8.04). For example, if the desired output format is two digits to the right of the decimal point and two to the left of the decimal point the user may type

```
*TYPE %4.02,12.22+2.37
```

and FOCAL will type

```
= 14.59*
```

Notice above that when the format operator is followed by other data, a comma must be typed directly after the format operator.

In the following examples, the number 67823 is typed out in several different formats.

```
*SET A=67823
*TYPE %6.01,A
= 67823.0*
*TYPE %5,A
= 67823*
*TYPE %8.03,A
= 67823.000*
```

If the specified output format is too small to contain the number, FOCAL prints the number in floating-point format

```
*TYPE %3, 67823
= 0.678E+05*
```

If the specified format is larger than the number, FOCAL inserts leading spaces:

```
*TYPE %7,67823
=   67823*
```

Leading blanks and zeros in integers are always ignored by FOCAL.

```
*TYPE %8.04, 0016, 0.016, ., 007
=    16.0000=     0.0160=     0.0000=     7.0000*
```

2.3 FLOATING-POINT FORMAT

To handle much larger and much smaller numbers, the user may request output in exponential form, which is called floating-point or E format. This notation is frequently used in scientific computations, and is the format in which FOCAL performs its internal computations. The user requests floating-point format by including a %, followed by a comma, in a TYPE command. From that point on, until the user again changes the output format, results will be typed out in floating-point format.

```
*TYPE %, 11
= 0.110000E+02*
```

This is interpreted as .11 times 10^2, or simply 11. Exponents can be used to ±616. The largest number that FOCAL can handle is +0.999999 times 10^{616}, and the smallest is -0.999999 times 10^{616}.

To demonstrate FOCAL's power to compute large numbers, you can find the value of 300 factorial by typing the following commands. (The FOR statement, which will be explained later, is used to set I equal to each integer from 1 to 300.)

```
*SET A=1
*FOR I=1,300; SET A=A*I
*TYPE %, A
= 0.306051E+615*                    (wait for FOCAL to
                                     calculate the value)
```

When changing output formats in FOCAL, a rounding error may occur in the last digit.

2.4 ARITHMETIC OPERATIONS

FOCAL performs the usual arithmetic operations of addition, subtraction, multiplication, division, and exponentiation. These are written by using the following symbols

	Symbol		Math Notation	FOCAL	
	↑	Exponentiation	3^3	3↑3	(Power must be a positive integer)
	*	Multiplication	3·3	3*3	
	/	Division	3÷3	3/3	
Same priority	+	Addition	3+3	3+3	
	-	Subtraction	3-3	3-3	

These operations may be combined into expressions. When FOCAL evaluates an expression which may include several arithmetic operations, the order of precedence is the same as that in the list above. Addition and subtraction are the only two operators with the same priority; expressions with these two operators are evaluated from left to right.

A+B*C+D is A+(B*C)+D not (A+B)*(C+D) nor (A+B)*C+D

A*B+C*D is (A*B)+(C*D) not A*(B+C)*D nor (A*B+C)*D

X/2*Y is $\frac{X}{2Y}$

Expressions are combinations of arithmetic operations or functions which may be reduced by FOCAL to a single number. Expressions may be enclosed in properly paired parentheses, square brackets, and angle brackets (use the enclosures of your choice; FOCAL is impartial and treats them all merely as enclosures).

For example,

```
*SET A1=(A+B)*<C+D>*[E+G]
*
```

The [and] enclosures are typed using SHIFT/K and SHIFT/M, respectively.

Expressions may be nested. FOCAL computes the value of nested expressions by doing the innermost first and then working outward.

```
*TYPE %, [2+(3-<1*1>+5)+2]
= 0.110000E+02*
```

This result is expressed in floating-point format.

2.5 MORE ABOUT SYMBOLS

The value of a symbolic name or identifier is not changed until the expression to the right of the equal sign is evaluated by FOCAL. Therefore, before it is evaluated, the value of a symbolic name or identifier can be changed by retyping the identifier and giving it a new value.

```
*SET A1=3↑2
*SET A1=A1+1
*TYPE %2, A1
= 10*
```

NOTE

Symbolic names or identifiers must not begin with the letter F.

The user may request FOCAL to type out all of the user defined identifiers, in the order of definition, by typing a dollar sign ($) after a TYPE command.

```
*TYPE %6.05,$
```

The user's symbol table is typed out like this

```
A@(00)= 0.306051E+615
B@(00)= 1111.11
C@(00)= 39.0000
I@(00)= 301.000
A1(00)= 10.0000
D@(00)= 0.00000
E@(00)= 0.00000
G@(00)= 0.00000
*
```

If an identifier consists of only one letter, an @ is inserted as a second character in the symbol table printout, as shown in the example above. An identifier may be longer than two characters, but only the first two will be recognized by FOCAL and thus stored in the symbol table. Notice that for numbers with more than one integer part, the output format operator % 6.05 is ignored so that the whole number can be printed.

2.6 SUBSCRIPTED VARIABLES

FOCAL always allows identifiers, or variable symbols, to be further identified by subscripts (range ±2047) which are enclosed in parentheses immediately following the identifier. A subscript may also be an expression:

```
*SET A1(I+3*J)=2.71; SET X1(5+3*J)=2.79
```

The ability of FOCAL to compute subscripts is especially useful in generating arrays for complex programming problems. A convenient way to generate linear subscripts is shown in Section 4.7.

2.7 THE ERASE COMMAND

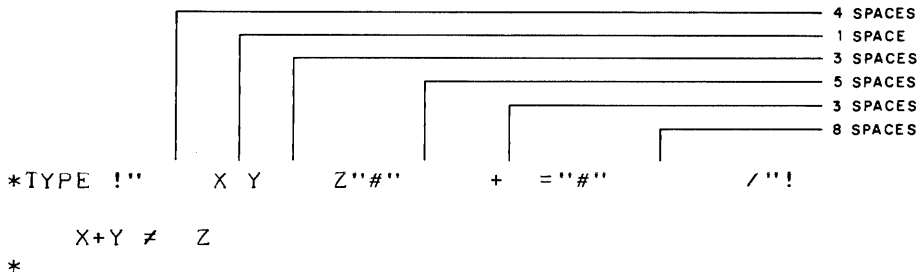
It is useful at times to delete all of the symbolic names which you have defined in the symbol table. This is done by typing a single command: ERASE. Since FOCAL does not clear the user's symbol table area in core memory when it is first loaded, it is good programming practice to type an ERASE command before defining any symbols. (See Section 3.5.)

2.8 HANDLING TEXT OUTPUT

Text strings are enclosed in quotation marks ("...") and may include most Teletype printing characters and spaces. The carriage return, line feed, and leader-trailer characters are not allowed in text strings. In order to have FOCAL type an automatic carriage return-line feed at the end of a text string, the user inserts an exclamation mark (!).

```
*TYPE "ALPHA"!"BETA"!"DELTA"!
ALPHA
BETA
DELTA
*
```

To get a carriage return without a line feed at the end of a text typeout, the user inserts a number sign (#) as shown below.



The number sign operator is useful in formatting output and in plotting another variable along the same coordinate.

2.9 INDIRECT COMMANDS

Up to this point only commands which are executed immediately by FOCAL have been discussed.

However, if a Teletype line is prefixed by a line number, that line is not executed immediately; instead, it is stored by FOCAL for later execution, usually as part of a sequence of commands. Line numbers must be in the range 1.01 to 31.99. The numbers 1.00, 2.00, etc., are illegal line numbers; they are used to indicate the entire group. The number to the left of the point is called the group number; the number to the right is called the step number. For example,

```
*ERASE
*1.1 SET A=1
*1.3 SET B=2
*1.5 TYPE %1,A+B
*
```

Indirect commands are executed by typing GO, GOTO, or DO commands, which may be direct or indirect.

The GO command causes FOCAL to go to the lowest numbered line to begin executing the program. If the user types a direct GO command after the indirect commands above, FOCAL will start executing at line 1.1.

```
*GO
= 3*
```

The GOTO command causes FOCAL to start the program by executing the command at a specified line number. If the user types

```
*GOTO 1.3
= 2*
```

FOCAL started executing the program at the second command in the example above.

The DO command is used to transfer control to a specified step, or group of steps, and then return automatically to the command following the DO command.

```
*ERASE ALL
*1.1 SET A=1; SET B=2
*1.2 TYPE "STARTING"
*1.3 DO 3.2
*2.1 TYPE " FINISHED"
*3.1 SET A=3; SET B=4
*3.2 TYPE %1, A+B
*GO
STARTING= 3 FINISHED= 7*
```

When the DO command at line 1.3 was reached, the command TYPE %1, A+B was performed and then the program returned to line 2.1.

The DO command can also cause FOCAL to jump to a group of commands and then return automatically to the normal sequence, as shown in the example below.

```
*ERASE ALL
*1.1 TYPE "A "
*1.2 TYPE "B "
*1.3 TYPE "C "
*1.4 DO 5.Ø
*1.5 TYPE " END"; GOTO 6.1
*5.1 TYPE "D "
*5.2 TYPE "E "
*5.3 TYPE "F "
*6.1 TYPE "."
*GO
A B C D E F  END.*
```

When the DO command at line 1.4 was reached, FOCAL executed lines 5.1, 5.2, and 5.3 and then returned to line 1.5.

An indirect command can be inserted in a program by using the proper sequential line number. For example,

```
*ERASE ALL
*4.8 SET A=1; SET B=2
*6.3 TYPE %5.4,B/C+A
*4.9 SET C=1.31*.29
*GO
= 6.2646*
```

where line 4.9 will be executed before line 6.3 and after line 4.8. FOCAL arranges and executes indirect commands in numerical sequence by line number, starting with the smallest line number and going to the largest.

2.10 ERROR DETECTION

During execution, FOCAL checks all input for a variety of errors. When an error is detected FOCAL stops execution and types a question mark (?) followed by an error code (Refer to Appendix B for a list of all error messages and their meanings). An asterisk (*) is typed after each error message and FOCAL waits for more user input.

When the error occurs in a direct statement, FOCAL types the error message immediately after the user terminates that line, because direct statements are executed immediately after the line terminator. The error code message is in the form ?XX.XX, where XX.XX is an arbitrary number indicating the type of error. For example,

```
*SET A=2; PET B=4; TYPE A+B
?03.28
*
```

PET is not a FOCAL command; therefore, FOCAL issued the error code ?03.28, indicating an illegal command was used.

When an error occurs in an indirect statement, the error message is typed when FOCAL encounters that statement during program execution. In addition to the error code, FOCAL types the line number containing the error in the form ?XX.XX@gg.ss, where XX.XX is the error code and gg.ss is the line number containing the error. For example,

```
*1.10 SET A=2; TYPE "A",A,!
*1.20 SET B=4; TYPE "B",B,!
*1.30 GOTO 1.01
*1.40 TYPE "A+B",A+B
*GO
A=     2.0000
B=     4.0000
?03.05 @ 01.30
*
```

FOCAL executed lines 1.10 and 1.20 and then recognized that 1.01 is an illegal line number used after GOTO. Therefore, it issued the error message 03.05 @ 1.30 to indicate that an illegal line was called by a GOTO command.

The WRITE command without an argument can be used to cause FOCAL to print out the entire indirect program, allowing the user to visually check it for errors.

NOTE

Errors ?01.00, ?00.00, and ?11.35 may be followed by nonexistent or false line numbers because these errors are time dependent.

2.11 CORRECTIONS

If the user types the wrong character, or several wrong characters, he can use the RUBOUT key, which echoes a backslash (\) for each RUBOUT typed, to erase one character to the left each time the RUBOUT key is depressed. For example,

```
*ERASE ALL
*1.1 P\TYPE X-Y
*1.2 SET $=13\\\\X=13
*WRITE
C-FOCAL,1969

01.10 TYPE X-Y
01.20 SET X=13
*
```

The left arrow (←) erases everything which appears to its left on the same line, except when being used to correct a value typed after a colon (:) in response to an ASK command (see Section 3.2).

```
*1.3 TYPE A,B,C←
*WRITE
C-FOCAL,1969

01.10 TYPE X-Y
01.20 SET X=13
*
```

A line can be overwritten by repeating the same line number and typing the new command.

```
*14.99 SET C9(N+3) = 15
*
```

is replaced by typing

```
*14.99 TYPE C9/Z5-2
*WRITE 14.99
14.99 TYPE C9/Z5-2
*
```

A line or group of lines may be deleted by using the ERASE command with an argument. For example, to delete line 2.21, the user types

```
*ERASE 2.21
*
```

To delete all of the lines in group 2, the user types

```
*ERASE 2.Ø
*
```

Used alone, without an argument, the ERASE command causes FOCAL to erase the user's entire symbol table. Since FOCAL does not zero memory when loaded, it is good practice to ERASE before defining symbols. The command ERASE ALL erases all user input.

Typing WRITE after making corrections causes FOCAL to print the indirect program as altered. This is useful for checking commands and for obtaining a "clean" program printout.

The MODIFY command is another valuable feature. It may be used to change any number of characters in a particular line, as explained in Section 3.14.

2.12 ALPHANUMERIC NUMBERS (Using Letters as Numbers)

Numbers must start with a numeral but may contain letters. FOCAL interprets as a number any character string beginning with a numeral, 0 through 9. An alphanumeric number is a string of alphanumeric characters (excluding symbols) which starts with a number. For example,

0ABC 23CAT 9XYZ

Each letter in an alphanumeric number is taken as a number, with each letter A through Z having the value of 1 through 26 respectively, except for E which has special meaning and is explained below.

A = 1	J = 10	S = 19
B = 2	K = 11	T = 20
C = 3	L = 12	U = 21
D = 4	M = 13	V = 22
E = (exponentiation)	N = 14	W = 23
F = 6	O = 15	X = 24
G = 7	P = 16	Y = 25
H = 8	Q = 17	Z = 26
I = 9	R = 18	

An easy way to give FOCAL numerical valued letters is to start with the number 0, as in the following example.

```
*TYPE %2, ØAB
= 12*
```

After 0, A = 1 and B = 2; therefore, AB = 12. Also,

```
*TYPE ØAB+ØC
= 15*
```

After 0, A = 1, B = 2, and C = 3, then 12 + 3 = 15. Therefore,

```
*TYPE ØXYZ+1
= 2677*
```

because

```
X = 24
Y =  25
Z =   26
+1  +  1
------
  2677
```

The above example can be solved using the following algorithm.

$$(X \text{ times } 10^2) + (Y \text{ times } 10^1) + (Z \text{ times } 10^0) + 1 = 2677$$

or

$$(24 \times 100) + (25 \times 10) + (26 \times 1) + 1 = 2677$$

Taken as a numeral, the letter E has special meaning. It denotes exponentiation, where the subsequent alphanumerics are taken as the exponent of the preceding alphanumerics.

```
*TYPE %8,ØAEC
=      1ØØØ*
*TYPE ØAEG
= 1ØØØØØØØ*
```

Only one E is allowed in any one alphanumeric number.

Alphabetic characters may be used when assigning numerical values to identifiers or variables in response to an ASK statement. An example of this use can be found in Appendix F.

CHAPTER 3

FOCAL COMMANDS

3.1 TYPE

The TYPE command is used to request that FOCAL compute and type out a text string, the result of an expression, or the value of an identifier. For example

```
*4.14 TYPE 8.1+3.2-(28.3*5)/2.5↑7,!
*4.15 TYPE (2.2+3.5)*(7.2/3)/59.1↑3,!
*
```

NOTE

The exclamation point (!) indicates a carriage return/ line feed pair. The number sign (#) indicates a carriage return without a line feed.

Several expressions may be computed in a single TYPE command, with commas terminating each expression.

```
*ERASE
*9.19 TYPE %4.01,A1*2,E+2↑5,2.51*81.1
*DO 9.19
=    0.0=   32.0= 203.6*
```

The output format may be included in the TYPE statement as shown in the example above and as explained in Section 2.2.

The user may request a typeout of all identifiers which he has defined by typing TYPE $ and a carriage return. This causes FOCAL to type out the identifiers with their values, in the order in which they were defined. The $ may follow other statements in a TYPE command, but must be the last operation on the line.

```
*ERASE
*SET L=33; SET B=87; SET Y=55; SET C9=91
*TYPE $
L@(00)=   33.0
B@(00)=   87.0
Y@(00)=   55.0
C9(00)=   91.0
*
```

A text string enclosed in quotation marks may be included in a TYPE command. A carriage return may replace the terminating quotation mark, as shown below:

```
*1.2 TYPE "X SQUARED=
*
```

A text string or any FOCAL command or group of commands may not exceed the capacity of a Teletype line, which is 72 characters on the 33 ASR Teletype. A line may not be continued on the following line. To print out a longer text, each line must start with a TYPE command.

FOCAL does not automatically perform a carriage return after executing a TYPE command. The user may insert a carriage return-line feed by typing an exclamation mark (!). To insert a carriage return without a line feed, the user types a number sign (#). Spaces may be inserted by enclosing them in quotation marks. These operations are useful in formatting output.

3.2 ASK

The ASK command is normally used in indirect commands to allow the user to input data at specific points during the execution of his program. The ASK command is written in the form

```
*11.99 ASK X,Y,Z
*
```

When line 11.99 is encountered by FOCAL, it types a colon (:). The user then types a value in any format for the first identifier, followed by a terminator†. FOCAL then types another colon and the user types a value for the second identifier. This continues until all the identifiers or variables in the ASK statement have been given values,

```
*11.99 ASK X,Y,Z
*DO 11.99
:5:4:3*
```

where the user typed 5, 4, and 3 as the values, respectively, for X, Y, and Z.

FOCAL recognizes the value when its terminator is typed. Therefore, a value can be changed but only before typing its terminator. This is done by typing a left arrow (←) immediately after the value, and

†Terminators are space, comma, ALT MODE, and RETURN keys.

then typing the correct value followed by its terminator. This is the exception to the use of the left arrow, as explained in Section 2.11.

The ALT MODE, when used as a terminator, is nonspacing and leaves the previously defined variable unchanged, as shown below.

```
*SET A=5
*ASK A
:123*                    (user depressed the ALT MODE Key after typing 123)
*TYPE A
=    5.Ø*
```

ALT MODE is frequently used when the user does not wish to change the value of one or more identifiers in an ASK command.

```
*11.99 ASK X,Y,Z
*DO 11.99
:5,:4,:3,*
*DO 11.99
:8,,:1Ø,*                (User did not wish to enter new value for Y, so he typed
*TYPE X,Y,Z               ALT MODE in response to second colon.)
=         8=         4=         1Ø*
```

A text string may be included in an ASK statement by enclosing the string in quotation marks.

```
*1.1Ø ASK "HOW MANY APPLES DO YOU HAVE?" APPLES
*DO 1.1Ø
HOW MANY APPLES DO YOU HAVE?:25
*
```

The identifier AP (FOCAL recognizes the first two characters only) now has the value 25.

3.3 WRITE

A WRITE command without an argument causes FOCAL to write out all indirect statements which the user has typed. Indirect statements are those preceded by a line number.

A group of line numbers, or a specific line, may be typed out with the WRITE command using arguments, as shown below.

```
*7.97 WRITE 2.Ø          (FOCAL types all group 2 lines)
*7.98 WRITE 2.1          (FOCAL types line 2.1)
*7.99 WRITE              (FOCAL types all numbered lines)
*
```

3.4 SET

The SET command is used to define identifiers. When FOCAL executes a SET command, the identifier and its value are stored in the user's symbol table, and that value will be substituted for the identifier when the identifier is encountered in the program.

```
*ERASE ALL
*3.4 SET A=2.55; SET B=8.Ø5
*3.5 TYPE %,A+B
*GO
= Ø.1Ø6ØØØE+Ø2*
```

An identifier may be set equal to previously defined identifiers, which may be used in arithmetic expressions.

```
*3.7 SET G=(A+B)*2.2↑5
*
```

3.5 ERASE

An ERASE command without an argument is used to delete all identifiers, with their values, from the symbol table.

If the ERASE command is followed by a group number or a specific line number, a group of lines or a specific line is deleted from the program.

```
*ERASE 2.Ø
*ERASE 7.11
*
```

The ERASE ALL command erases all the user's input.

In the following example, an ERASE command is used to delete line 1.50.

```
*ERASE ALL
*1.20 SET B=2
*1.30 SET C=4
*1.40 TYPE B+C
*1.50 TYPE B-C
*ERASE 1.50
*WRITE ALL
C-FOCAL,1969

01.20 SET B=2
01.30 SET C=4
01.40 TYPE B+C
*
```

3.6 GO

The GO command requests that FOCAL execute the program which starts with the lowest numbered line. The remainder of the program will be executed in line number sequence. Line numbers must be in the range 1.01 to 31.99.

3.7 GOTO

The GOTO command causes FOCAL to transfer control to a specific line in the indirect program. It must be followed by a specific line number. After executing the command at the specified line, FOCAL continues to the next higher line number, executing the program sequentially.

```
*ERASE ALL
*1.1 TYPE "A"
*1.2 TYPE "B"
*1.3 TYPE "C"
*1.4 TYPE "D"
*GOTO 1.2
BCD*
```

3.8 DO

The DO command transfers control momentarily to a single line, a group of lines, or the entire indirect program. If transfer is made to a single line, the statements on that line are executed, and control is transferred back to the statement following the DO command. Thus, the DO command makes a subroutine of the commands transferred to, as shown in this example.

```
*ERASE ALL
*1.1 TYPE "F"
*1.2 DO 2.3; TYPE "CA"
*1.3 TYPE "L"!
*1.4 QUIT
*2.3 TYPE "O"
*GO
FOCAL
*
```

If a DO command transfers control to a group of lines, FOCAL executes the group sequentially and returns control to the statement following the DO command.

If DO is written without an argument, FOCAL executes the entire indirect program.

DO commands cause specified portions of the indirect program to be executed as closed subroutines. These subroutines may also be terminated by a RETURN command.

If a GOTO or an IF command is executed within a DO subroutine, two actions are possible:

a. If a GOTO or IF command transfers to a line inside the DO group, the remaining commands in that group will be executed as in any subroutine before returning to the command following the DO.

b. If transfer is to a line outside the DO group, that line is executed and control is returned to the command following the DO; unless that line contains another GOTO or IF.

```
*ERASE ALL
*1.1 TYPE "A"; SET X=-1; DO 3.1; TYPE "D"; DO 2
*1.2 DO 2
*
*2.1 TYPE "G"
*2.2 IF (X)2.5,2.6,2.7
*2.5 TYPE "H"
*2.6 TYPE "I"
*2.7 TYPE "J"
*2.8 TYPE "K"
*2.9 TYPE %2.Ø1, X; TYPE " "; SET X=X+1
*
*3.1 TYPE "B"; GOTO 5.1; TYPE "F"
*
*5.1 TYPE "C"
*5.2 TYPE "E"
*5.3 TYPE "L"
*GO
```

(FOCAL types the answer)

```
ABCDGHIJK=-1.Ø GIJK= Ø.Ø GJK= 1.Ø BCEL*
```

3.9 IF

In order to transfer control after a comparison, FOCAL contains a conditional IF statement. The normal form of the IF statement consists of the word IF, a space, a parenthesized expression or variable, and three line numbers in order, separated by commas. The expression is evaluated, and the program transfers control to the first line number if the expression is less than zero, to the second line number if the expression has a value of zero, or to the third line number if the value of the expression is greater than zero. The IF expression or variable must be enclosed in parentheses.

The program below transfers control to line number 2.10, 2.30, or 2.50, according to the value of the expression in the IF statement.

```
*2.1 TYPE "LESS THAN ZERO"; QUIT
*2.3 TYPE "EQUAL TO ZERO"; QUIT
*2.5 TYPE "GREATER THAN ZERO"; QUIT
*IF (25-25)2.1,2.3,2.5
EQUAL TO ZERO*
```

The IF statement may be shortened by terminating it with a semicolon or carriage return after the first or second line number. If a semicolon follows the first line number, the expression is tested and control is transferred to that line if the expression is less than zero. If the expression is not less than zero, the program continues with the next statement,

```
*2.20 IF (X)1.8; TYPE "Q"
*
```

In the above example, when line 2.20 is executed, if X is less than zero, control is transferred to line 1.8. If not, Q is typed out.

```
*3.19 IF (B)1.8,1.9
*3.20 TYPE B
*
```

In this example, if B is less than zero, control goes to line 1.8, if B is equal to zero, control goes to line 1.9. If B is greater than zero, control goes to the next statement, which in this case is line 3.20, and the value of B is typed out.

IF commands are useful in programs that require a keyboard response during program execution. Refer to Appendix F for use of this feature.

3.10 RETURN

The RETURN command is used to exit from a DO subroutine. When a RETURN command is encountered during execution of a DO subroutine, the program exits from its subroutine status and returns to the command following the DO command that initiated the subroutine status.

3.11 QUIT

A QUIT command causes the program to halt and return control to the user. FOCAL types an asterisk and the user may type another command.

3.12 COMMENT

Beginning a command string with the letter C will cause the remainder of that line to be ignored so that comments may be inserted into the program. Such lines will be skipped over when the program is executed, but will be typed out by a WRITE command.

3.13 FOR

This command is used for convenience in setting up program loops and iterations. The general format is

```
*FOR A=B,C,D;(COMMAND)
```

The identifier A is initialized to the value B, then the command following the semicolon is executed. When the command has been executed, the value of A is incremented by C and compared to the value of D. If A is less than or equal to D, the command after the semicolon is executed again. This process is repeated until A is greater than D, and FOCAL goes to the next sequential line.

The identifier A must be a single variable. B, C, and D may be either expressions, variables, or numbers. If comma and the value C are omitted, it is assumed that the increment is one. If C, D is omitted, it is handled like a SET statement and no iteration is performed.

The computations involved in the FOR statement are done in floating-point arithmetic, and it may be necessary, in some circumstances, to account for this type of arithmetic computation.

Example 1 below is a simple example of how FOCAL executes a FOR command. Example 2 shows the FOR command combined with a DO command.

Example 1:

```
*ERASE ALL
*1.1 SET A=100
*1.2 FOR B=1,1,5; TYPE %5.02, "B IS"B+A,!
*GO
B IS= 101.00
B IS= 102.00
B IS= 103.00
B IS= 104.00
B IS= 105.00
*
```

Example 2:

```
*1.1 FOR X=1,1,5;DO 2.0
*1.2 GOTO 3.1
*
*2.1 TYPE !"      "%3,"X"X
*2.2 SET A=X+100.000
*2.3 TYPE !"      "%5.02,"A"A
*
*3.1 QUIT
*GO

     X=    1
     A= 101.00
     X=    2
     A= 102.00
     X=    3
     A= 103.00
     X=    4
     A= 104.00
     X=    5
     A= 105.00*
```

3.14 MODIFY

Frequently, only a few characters in a particular line require changes. To facilitate this job, and to eliminate the need to replace the entire line, the FOCAL programmer may use the MODIFY command. Thus, in order to modify the characters in line 5.41, the user types MODIFY 5.41. This command is terminated by a carriage return whereupon the program waits for the user to type that character in the position in which he wishes to make changes or additions. This character is not printed. After he has typed the search character, the program types out the contents of that line until the search character is typed.

At this point, the user has seven options:

a. Type in new characters in addition to the ones that have already been typed out.

b. Type a form-feed (CTRL/L); this will cause the search to proceed to the next occurrence, if any, of the search character.

c. Type a CTRL/BELL; this allows the user to change the search character just as he did when first beginning to use the MODIFY command.

d. Use the RUBOUT key to delete one character to the left each time RUBOUT is depressed.

e. Type a left arrow (←) to delete the line over to the left margin.

f. Type a carriage return to terminate the line at that point, removing the text to the right.

g. Type a LINE FEED to save the remainder of the line.

The ERASE ALL and MODIFY commands are generally used only in immediate mode because they return to command mode upon completion. (The reason for this is that internal pointers may be changed by these commands.)

During command input, the left arrow will delete the line numbers as well as the text if the left arrow is the rightmost character on the line.

Notice the errors in line 7.01 below.

```
*7.Ø1 JACK AND BILL W$NT UP THE HALL
*MODIFY 7.Ø1
JACK AND B\JILL W$\ENT UP THE HA\ILL
*WRITE 7.Ø1
Ø7.Ø1 JACK AND JILL WENT UP THE HILL
*
```

To modify line 7.01, a B was typed by the user to indicate the character to be changed. FOCAL stopped typing when it encountered the search character, B. The user typed the RUBOUT key to delete the B, and then typed the correct letter J. He then typed the CTRL/BELL keys followed by the $, the next character to be changed. The RUBOUT deleted the $ character, and the user typed an E. Again a search was made for an A character. This was changed to I. A LINE FEED was typed to save the remainder of the line.

3.14.1 Caution

When the MODIFY command is used the values in the user's symbol table are reset to zero. Therefore, if the user defines his symbols in direct statements and then uses a MODIFY command, the values of his symbols are erased and must be redefined.

However, if the user defines his symbols by means of indirect statements prior to using a MODIFY command, the values will not be erased because these symbols are not entered in the symbol table until the statements defining them are executed.

Notice in the example below that the values of A and B were set using direct statements. The use of the MODIFY command reset their values to zero and listed them after the defined symbols.

```
*ERASE ALL
*SET A=1
*SET B=2
*1.1 SET C=3
*1.2 SET D=4
*1.3 TYPE A+B+C+D; TYPE !; TYPE $
*MODIFY 1.1
SET C=3\5
*GO
=   9.00
C@(00)=   5.00
D@(00)=   4.00
A@(00)=   0.00
B@(00)=   0.00
*
```

3.15 USING THE TRACE FEATURE

The trace feature is useful in checking an operating program; those parts of the program which the user has enclosed in question marks will be printed out as they are executed.

In the following example, parts of 3 lines are printed.

```
*ERASE ALL
*1.1 SET A=1
*1.2 SET B=5
*1.3 SET C=3
*1.4 TYPE %2, ?A+B-C?,!
*1.5 TYPE ?B+A/C?,!
*1.6 TYPE ?B-C/A?
*GO
A+B-C=   3
B+A/C=   5
B-C/A=   2*
```

When only one ? is inserted, the trace feature becomes operative when FOCAL encounters the ? during execution, and the program is printed out from that point until another ? is encountered (the program may loop through the same ?), until an error is encountered (execution stops and an error message is typed), or until program completion.

```
*ERASE ALL
*1.1 ?SET A=ØB; TYPE %3,A!
*1.2 FOR B=1,1,4; TYPE B+A!
*GO
SET A=ØB; TYPE %3,A!
=    ØFOR B=1,1,4; TYPE B+A!
=    1 TYPE B+A!
=    2 TYPE B+A!
=    3 TYPE B+A!
=    4*
```

In this example, FOCAL encountered the ? as it entered line 1.1 and traced the entire program.

3.16 MATHEMATICAL FUNCTIONS

The functions are provided to give extended arithmetic capabilities and to give the potential for expansion to additional input-output devices. A standard function call consists of four letters beginning with the letter F and followed by a parenthetical expression.

FSGN(A-B*2)

There are three basic types of functions, two of which are included in the basic FOCAL program. The first type contains integer part, sign part, and absolute value functions.

In the second type, the extended arithmetic functions, are loaded at the option of the user. They will consume approximately 800 locations of the users program storage area. These arithmetic functions are adapted from the extended arithmetic functions of the PDP-8 three-word floating-point package and are fully described in the Floating Point System Manual (Order No. DEC-08-YQYA-D).

The input-output functions are the third type. These include a nonstatistical random number generator (FRAN). This function uses the FOCAL program itself as a table of random numbers. An expanded version could incorporate the random number generator from the DECUS library. Following are examples of the functions now available.

a. The square root function (FSQT) computes the square root of the expression within parentheses.

```
*TYPE %2, FSQT(4)
=  2*
*TYPE FSQT(9)
=  3*
*TYPE FSQT(144)
= 12*
```

b. The absolute value function (FABS) outputs the absolute or positive value of the number in parentheses.

```
*TYPE FABS(-66)
= 66*
*TYPE FABS(-23)
= 23*
*TYPE FABS(-99)
= 99*
```

c. The sign part function (FSGN) outputs the sign part (+ or -) of a number and the integer part becomes a 1.

```
*TYPE FSGN(4-6)
=- 1*
*TYPE FSGN(4-4)
=  1*
*TYPE FSGN(-7)
=- 1*
```

d. The integer part function (FITR) outputs the integer part of a number up to 2046.

```
*TYPE FITR(5.2)
=  5*
*TYPE FITR(55.66)
= 55*
*TYPE FITR(77.434)
= 77*
*TYPE FITR(-4.1)
=- 4*
```

e. The random number generator function (FRAN) computes a nonstatistical pseudo-random number between ±1.

```
*TYPE %, FRAN( )
= Ø.607295E+ØØ*
*TYPE FRAN( )
= Ø.737615E+ØØ*
```

f. The exponential function (FEXP) computes e to the power within parentheses. (e = 2.718281)

```
= Ø.194829E+Ø1*
*TYPE FEXP(.666953)
= Ø.194829E+Ø1*
*TYPE FEXP(1.23456)
= Ø.343687E+Ø1*
*TYPE FEXP(-1.)
= Ø.367879E+ØØ*
```

g. The logarithm function (FLOG) computes the natural logarithm ($\log_e$) of the number within parentheses.

```
*TYPE FLOG(1.00000)
= 0.000000E+00*
*TYPE FLOG(1.98765)
= 0.686953E+00*
*TYPE %5.03, FLOG(2.065)
=  0.725*
```

h. The arc tangent function (FATN) calculates the angle in radians whose tangent is the argument within parentheses.

```
*TYPE FATN(1.)
= 0.785398E+00*
*TYPE FATN(.31305)
= 0.303386E+00*
*TYPE FATN(3.141592)
= 0.126263E+01*
```

i. The sine function (FSIN) calculates the sine of an angle in radians.

```
*TYPE %, FSIN(3.14159)
= 0.333786E-05*
*TYPE FSIN(1.400)
= 0.985448E+00*
```

Since FOCAL requires that angles be expressed in radians, to find a function of an angle in degrees, the conversion factor, $\pi/180$, must be used. To find the sine of 15 degrees,

```
*SET PI=3.14159; TYPE FSIN(15*PI/180)
= 0.258819E+00*
*TYPE FSIN(45*3.14159/180)
= 0.707106E+00*
```

j. The cosine function (FCOS) calculates the cosine of an angle in radians.

```
*TYPE FCOS(2*3.141592)
= 0.999996E+00*
*TYPE FCOS(.50000)
= 0.877582E+00*
*TYPE FCOS(45*3.141592/180)
= 0.707107E+00*
```

CHAPTER 4
EXAMPLES OF FOCAL PROGRAMS

The programs in this chapter reveal some of FOCAL's features in various applications. The examples show that FOCAL finds practical application in any situation.

a. FORTRAN-type problems are handled easily with little programming time.

b. FOCAL's easy-to-learn language allows the user to concentrate more on his problem than on programming.

c. FOCAL, as a tool, is easier to learn and use than a slide rule or any other desk calculator, and it offers vastly more than any combination of previous problem solving tools.

d. FOCAL can calculate complex problems and print/display the results in "one fell swoop."

The example programs included in this chapter were first punched onto paper tape. Each program was loaded into core using FOCAL's high-speed paper tape reader input feature. The WRITE command was then used to get the program printout for inclusion here. Then the GO command was issued to execute each program.

When using the WRITE command, FOCAL immediately identifies the version of the FOCAL tape being used--in this case, C-FOCAL, 1969. The C preceding FOCAL, 1969 is the comment line indicator.

4.1 TABLE GENERATION USING FUNCTIONS

The ability to evaluate simple arithmetic expressions and to generate values with the aid of the extended functions is one of the first benefits to be obtained from learning the FOCAL language. In this example, a table of the sine, cosine, natural logarithm, and exponential values is generated for a series of arguments. As one becomes familiar with these and other library functions, it becomes easy to combine them with the standard arithmetic operations of addition, subtraction, multiplication, division, and

exponentiation. The user should then be able to evaluate any given formula for a single value or for a range of values as in this example.

Although FOCAL allows the typing of more than one command per line, each command in this example has been typed on a separate line to maintain clarity. In this example, line 01.05 outputs the desired column headings. Line 01.10 is the loop to generate values for I, beginning with the value 1.00000 and continuing in increments of .00001 up through the value 1.00010; the DO 2.05 command at the end of this second line causes line 02.05 to be executed for each value of I. Line 02.05 is the command to evaluate the various library functions for the I arguments; the %7.06 specifies that all output results up to the next % symbol are to appear in fixed-point format with one digit position to the left of the decimal point and six digit positions to the right: the second % symbol reverts the output mode back to floating point for the remaining values - FLOG(1) and FEXP(1). Line 01.20 (optional) returns control to the user.

Several techniques can be noted in this example.

a. FOCAL commands can be abbreviated to the first letter of the command followed by a space, as shown by the use of T instead of TYPE. This technique can be used to shorten command strings.

b. Arguments can be enclosed in various ways: (), < >, [ ]. This ability is useful in matching correctly when a number of such enclosures appear in a command.

c. Spaces can be inserted in an output format by enclosing the appropriate number of spaces within quotation marks. Such use of spacing is recommended to improve the readability of the output results.

d. FOCAL's accuracy makes possible the use of very small loop increments (in this example, .00001).

```
*
C-FOCAL,1969

01.05 T "     I          SINE       COSINE         LOG              E"!
01.10 FOR I=1,.00001,1.0001; DO 2.05
01.20 QUIT

02.05 T %7.06,I," ",FSIN(I)," ",FCOS<I>," ",%,FLOG[I]," ",FEXP(I),!!
*
*
*GO
```

```
      I          SINE       COSINE        LOG            E
=  1.000000 = 0.841471 = 0.540302 = 0.000000E+00 = 0.271828E+01

=  1.000010 = 0.841476 = 0.540294 = 0.977507E-05 = 0.271831E+01

=  1.000020 = 0.841481 = 0.540286 = 0.195501E-04 = 0.271834E+01

=  1.000030 = 0.841486 = 0.540278 = 0.293249E-04 = 0.271836E+01

=  1.000040 = 0.841492 = 0.540269 = 0.390997E-04 = 0.271839E+01

=  1.000050 = 0.841497 = 0.540261 = 0.488744E-04 = 0.271842E+01

=  1.000060 = 0.841502 = 0.540253 = 0.586490E-04 = 0.271844E+01

=  1.000070 = 0.841508 = 0.540245 = 0.684235E-04 = 0.271847E+01

=  1.000080 = 0.841513 = 0.540236 = 0.781979E-04 = 0.271850E+01

=  1.000090 = 0.841518 = 0.540228 = 0.879723E-04 = 0.271852E+01

=  1.000100 = 0.841523 = 0.540220 = 0.977465E-04 = 0.271855E+01

*
```

4.2 FORMULA EVALUATION FOR CIRCLES AND SPHERES

In this example, FOCAL is used to calculate, label, and output the following values for an indefinite number of radii typed in by the user.

Given: radius(R)

Program calculates: circle diameter $2R$
circle area πR^2
circle circumference $2\pi R$
sphere volume $4/3\pi R^3$
sphere surface area $4\pi R^2$

Although the American system of inches is used in this example, conversions to other systems (metric, for example) could be very easily incorporated into the program, thus eliminating any need for hand-calculated conversions.

The program is very straightforward. ASK is used to allow the user to type in the radius value to be used in the calculations. SET is used to supply the value of π (PI). TYPE is used for all calculations and output. Note that if a value (such as PI in this example) is to be entered once and then used in repeated calculations, it should be entered by a SET command which is outside the calculation loop, otherwise, the variable would be set at the beginning of each pass through the loop. However, if the value of the variable changes during each iteration, then it must be calculated either by a SET or TYPE command within the loop.

The use of the GOTO command (line 01.60) results in an infinite loop of lines 01.10 through 01.60. This technique is used when the number of desired repetitions is not known. The looping process can be terminated at any time by typing CTRL/C. If, however, the number of desired repetitions is known (e.g., 10), the following method can be used.

```
*SET PI=3.14159
*1.1 ASK ...
  .
  .
  .
*1.6 TYPE !!!!!              (Eliminate GOTO 1.1)
*
*FOR I=1,10; DO 1            (Direct command; causes all
                              steps in group 1 to be ex-
                              ecuted 10 times)
```

The ability to choose between these methods provides great flexibility in actually running FOCAL programs.

```
*
C-FOCAL,1969

01.01 SET PI=3.141592
01.10 ASK "A RADIUS OF", R, " INCHES"
01.20 TYPE %8.04, !, " GENERATES A CIRCLE OF:", !
01.21 TYPE "           DIAMETER", 2*R, "  INCHES", !
01.30 TYPE "           AREA", PI*R↑2, "  SQUARE INCHES", !
01.35 TYPE "           CIRCUMFERENCE", 2*PI*R, "  INCHES", !
01.40 TYPE !, " AND A SPHERE OF:", !
01.49 TYPE "           VOLUME", (4/3)*PI*R↑3, "  CUBIC INCHES", !
01.50 TYPE "           AND SURFACE AREA", 4*PI*R↑2, "  SQUARE INCHES"
01.60 TYPE !!!; GOTO 1.1
*
*GO
```

```
A RADIUS OF:1  INCHES
 GENERATES A CIRCLE OF:
          DIAMETER=     2.0000  INCHES
          AREA=     3.1416  SQUARE INCHES
          CIRCUMFERENCE=     6.2832  INCHES

 AND A SPHERE OF:
          VOLUME=     4.1888  CUBIC INCHES
          AND SURFACE AREA=    12.5664  SQUARE INCHES


A RADIUS OF:1.414  INCHES
 GENERATES A CIRCLE OF:
          DIAMETER=     2.8280  INCHES
          AREA=     6.2813  SQUARE INCHES
          CIRCUMFERENCE=     8.8844  INCHES

 AND A SPHERE OF:
          VOLUME=    11.8423  CUBIC INCHES
          AND SURFACE AREA=    25.1252  SQUARE INCHES


A RADIUS OF:
```

4.3 TEMPERATURE CONVERSION

Measurement system conversions are time consuming in many lines of work. A short FOCAL program, such as the one illustrated in the following example, eliminates hours of repeated calculations. In this particular example, the problem is to convert temperatures from degrees Fahrenheit to degrees Centigrade, using the formula:

$$T°C = 5/9(T°F - 32)$$

This routine is quite similar in structure to the "Table Generation" example. The one basic difference is that here the user can input the loop parameters which govern the generation of the output. Thus, provision has been made for output of properly labeled requests for starting, ending, and incrementing values and their input for use by the program.

The ability for loop paramenters to be negative, zero, fractional, or expressions, provides power beyond many other similar languages in simplifying the routine's structure. It also reemphasizes the flexibility and control over FOCAL programs at the time they are run.

```
C-FOCAL,1969

02.10 ASK "FROM",START," TO",END," DEGREES FAHRENHEIT",!
02.20 ASK "     IN INCREMENTS OF",INCR," DEGREES",!
02.30 TYPE "THE APPROPRIATE FAHRENHEIT TO CENTIGRADE CONVERSIONS ARE:"
02.40 FOR T=START,INCR,END; TYPE !; DO 2.5
02.45 QUIT
02.50 TYPE "     ",T," FAHR. DEG.....",(T-32)*5/9," CENTIGRADE DEG."
*
*DO 2
FROM:-40  TO:80  DEGREES FAHRENHEIT
     IN INCREMENTS OF:20  DEGREES
THE APPROPRIATE FAHRENHEIT TO CENTIGRADE CONVERSIONS ARE:
     =-  40.0000 FAHR. DEG.....=-  40.0000 CENTIGRADE DEG.
     =-  20.0000 FAHR. DEG.....=-  28.8889 CENTIGRADE DEG.
     =    0.0000 FAHR. DEG.....=-  17.7778 CENTIGRADE DEG.
     =   20.0000 FAHR. DEG.....=-   6.6667 CENTIGRADE DEG.
     =   40.0000 FAHR. DEG.....=    4.4445 CENTIGRADE DEG.
     =   60.0000 FAHR. DEG.....=   15.5556 CENTIGRADE DEG.
     =   80.0000 FAHR. DEG.....=   26.6667 CENTIGRADE DEG.*
```

4.4 ONE-LINE FUNCTION PLOTTING

This example demonstrates the use of FOCAL to present, in graphic form, some given function over a range of values. In this example, the function used is

$$y = 30 + 15(\text{SIN}(x))e^{-.1x}$$

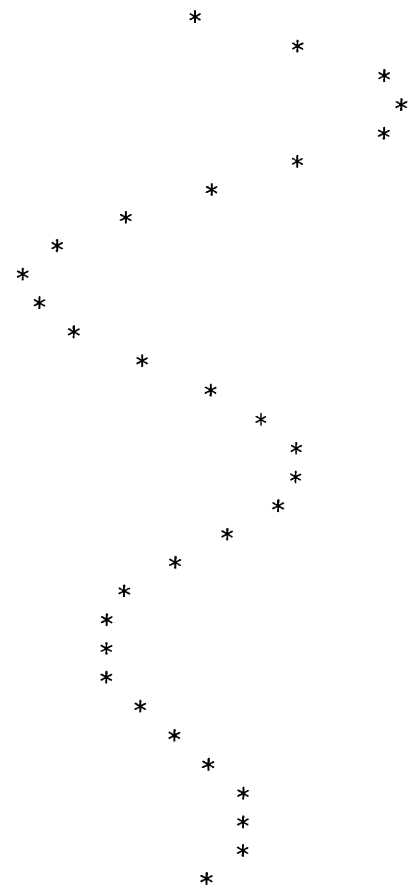
with x ranging from 0 to 15 in increments of .5. This damped sine wave has many physical applications, especially in electronics and mechanics (for example, in designing the shock absorbers of a car).

In the actual coding of the example, the variables I and J were used in place of x and y, respectively; any two variables could have been used. The single line 08.01 contains a set of nested loops for I and J. The J loop types spaces horizontally for the y coordinate of the function; the I loop prints the * symbol and the carriage return and line feeds for the x coordinate. The function itself is used as the upper limit of the J loop, again showing the power of FOCAL commands.

The technique illustrated by this example can be used to plot any desired function. Although the * symbol was used here, any legal FOCAL character is acceptable.

```
C-FOCAL,1969

08.01 F I=0,.5,15; T "*",!; F J=0,30+15*FSIN(I)*FEXP<-.1*I>; T " "
*
*DO 8.01
*
```



4.5 EXTENSIONS TO PLOTTING

In this next example, the wave form is the same as that shown under One-Line Function Plotting, i.e., a damped sinusoid; however, both the x and y axes have been added to increase the readability of the output. Since amplitude and the damping factor can be varied, a series of such plots shows their relative effects and can be used as a learning tool, or for trial and error solutions.

Another FOCAL command, IF, is used to position the x axis in the output. Two different options are used once the comparison within the parentheses is made. In line 02.10, the statement numbers separated by commas indicate where to branch if the expression is negative, zero, or positive,

respectively. However, in line 03.10 a branch is made to 3.3 if the expression is negative, or 3.2 if the expression is zero, but for the positive case (J is less than 31) the remainder of the line is executed. Whereas, in previous examples only single lines were executed as subroutines by DO commands (e.g., DO 2.05). This routine contains DO commands which execute a group of lines as a subroutine before returning to the statement following the DO (e.g., DO 2, DO 3).

It is often useful to superimpose one function plot upon another. This can be accomplished in FOCAL by replacing the exclamation point (representing a carriage return, line feed) with the number sign (representing a carriage return only) in certain TYPE commands. A very large number of functions using different plot symbols could be superimposed in this way. Often it is useful to follow each line in a function plot with the value of the function at that point, thus producing analog and digital output together. One can see from the spacing of the dots on the x and y axes that the Teletype produces a scale with a horizontal to vertical ratio of 5-to-3 (i.e., five horizontal spaces = 3 vertical line feeds). This factor must be taken into account when plotting closed curves such as circles.

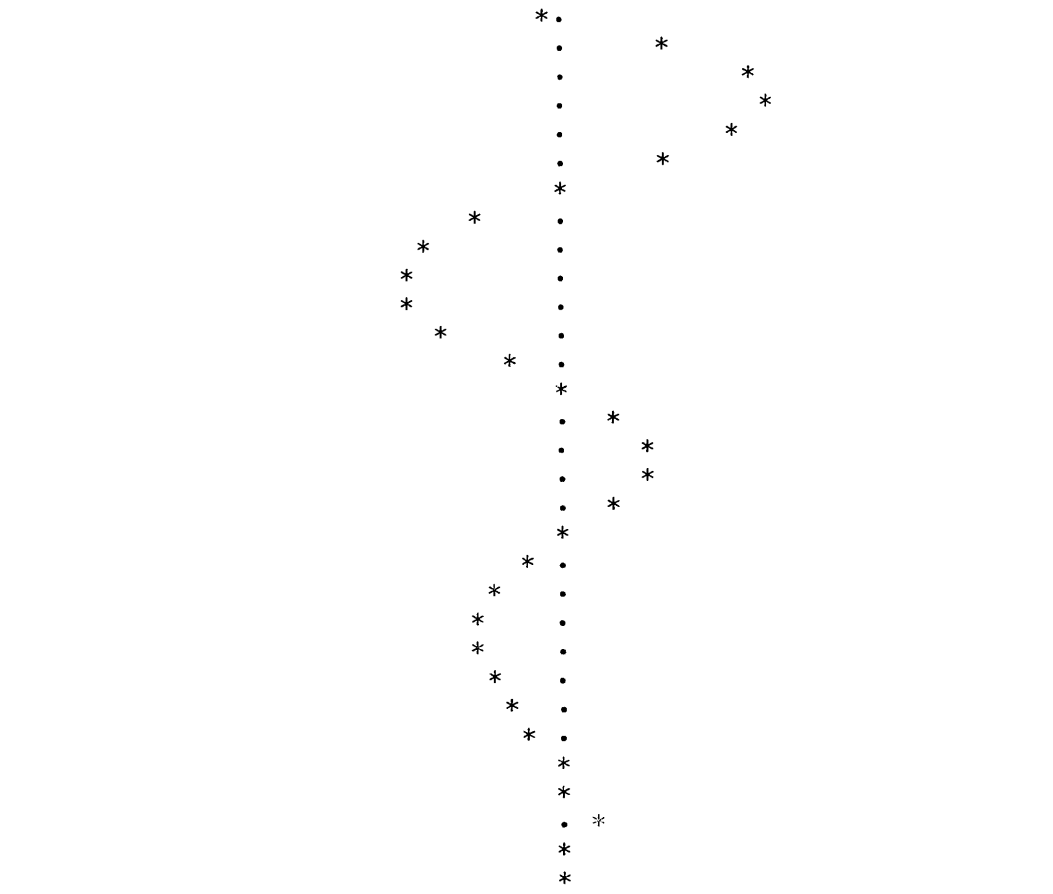
It should be noted that FOCAL library functions already provide for output displays on oscilloscopes as well as for analog-to-digital conversions.

```
C-FOCAL,1969

01.03 ASK "SINE WAVE AMPLITUDE", AMPL, !
01.04 ASK "DAMPING FACTOR COEFFICIENT", T, !
01.05 FOR K=0,60; TYPE "."
01.06 TYPE !; FOR I=0,.5,15; DO 1.11; TYPE "*"; DO 3
01.07 QUIT
01.11 FOR J=0,30+AMPL*FSIN(I)*FEXP(-T*I); DO 2; T " "

02.10 IF (J-32) 2.3, 2.2, 2.3
02.20 TYPE "."
02.30 RETURN

03.10 IF (31-J) 3.3, 3.2; FOR K=J,30; TYPE " "
03.20 TYPE "."
03.30 TYPE !; RETURN
*
*GO
SINE WAVE AMPLITUDE:15
DAMPING FACTOR COEFFICIENT:.135
```



4.5.1 Plotting on the Oscilloscope

This is an example of using the FDIS function for plotting on the oscilloscope. The function is used in the SET command of statement number 01.40 as shown below, which is equivalent to

SET H = FDIS (X,Y)

where X and Y are the X and Y coordinates of the point to be plotted on the scope.

The program will plot a sine wave with a user-determined number of complete cycles (Q).

```
C-FOCAL,1969

01.10 ASK "NO. OF CYCLES" Q
01.20 F I=0,50; S X(I)=20*I; S Y(I)=(FSIN(3.14159*I/(25/Q))+1)*500
01.30 T "READY TO PLOT" ,!
01.40 F I=0,50; S H=FDIS(X(I),Y(I))
01.50 G 1.4
*
*GO
NO. OF CYCLES:2
READY TO PLOT
```

4.6 DEMONSTRATION DICE GAME

Sooner or later, people who have access to a computer will try to "match brains" with it or use it for their own enjoyment. Such pastimes are usually keyboard oriented and FOCAL lends itself nicely to these ends. The following example uses the random number generator, FRAN( ), to produce dice combinations, plus IF logic to check bets and winning combinations.

Note again the use of initials to abbreviate commands throughout the example (remenber that each such abbreviation must be followed by a space). Lines beginning with a C indicate that the line is to be treated as a comment and is not to be interpreted or executed. If a comment statement is preceded by a statement number, the line is stored as part of the program but does not affect the program logic.

The random number generator must be modified for use with statistical or simulation programs to achieve true randomness. However, it is sufficiently random for most applications in its present form.

NOTE

We naturally cannot assume any responsibility for the use of this or any similar routines.

```
C-FOCAL,1969

01.10 S B=0;T !!"DICE GAME"!,"HOUSE LIMIT OF $1000
01.13 T ".  MINIMUM BET IS $1"!!
01.20 ASK "YOUR BET IS"A;IF (1000-A) 3.1
01.22 I (A-1)3.4,1.26,1.26
01.26 I (A-FITR(A))3.5,1.3,3.5
01.30 ASK M;DO 2;S D=C;DO 2;T " ";S D=D+C
01.32 I (D-7)1.42,3.2,1.42
01.40 I (D-2)1.5,3.3,1.5
01.42 I (D-11)1.4,3.2,3.3
01.50 I (D-3) 1.6,3.3,1.6
01.60 ASK M;DO 2;S E=C;DO 2;T " ";S E=E+C
01.72 I (E-7) 1.74,3.3,1.74
01.74 I (E-D)1.6,3.2,1.6

02.10 S C=FITR(10*FABS(FRAN()));I (C-6)2.2,2.2,2.1
02.20 I (C-1)2.1;T %1,"  "C;RETURN

03.10 T "HOUSE LIMITS ARE $1000"!!; G 1.2
03.20 S B=B+A;T %7,!"YOU WIN. YOUR WINNINGS ARE      "B,!!;GOTO 1.2
03.30 S B=B-A;T %7,!"SORRY, YOU LOSE. YOUR WINNINGS ARE"B,!!;G 1.2
03.40 T "MINIMUM BET IS $1"!!;G 1.2
03.50 T "NO PENNIES, PLEASE"!!;GOTO 1.2
*
*GO


DICE GAME
HOUSE LIMIT OF $1000.  MINIMUM BET IS $1

YOUR BET IS:.50 MINIMUM BET IS $1

YOUR BET IS:15 :
  = 6  = 3 :
  = 1  = 4 :
  = 4  = 5
YOU WIN. YOUR WINNINGS ARE      =       15

YOUR BET IS:5 :
  = 2  = 2 :
  = 6  = 1
SORRY, YOU LOSE. YOUR WINNINGS ARE =       10

YOUR BET IS:10 :
  = 6  = 5
YOU WIN. YOUR WINNINGS ARE      =       20

YOUR BET IS:  I'LL QUIT WHILE I'M AHEAD. THANKS!
```

4.7 SIMULTANEOUS EQUATIONS AND MATRICES

Many disciplines use subscripted variables for vectors in one, two, or more dimensions to store and manipulate data. A common use is the 2-dimensional array or matrix for handling sets of simultaneous equations. For example,

Given:
$$1X_1 + 2X_2 + 3X_3 = 4$$
$$4X_1 + 3X_2 + 2X_3 = 1$$
$$1X_1 + 4X_2 + 3X_3 = 2$$

Find: The values of X_1, X_2, and X_3 to satisfy all three equations simultaneously.

The solution can be reduced to simple mathematics between the various elements of the rows and columns until correct values of X are found.

Since FOCAL uses only a single subscript, the handling of two or more dimensions requires the generation of a linear subscript which represents the correct position if it were stored in normal order; i.e., leftmost subscript moving fastest.

4.7.1 In One Dimension

ARRAY( )		
A	0	Element D could be represented as ARRAY(3);
B	1	any element in this array can be represented
C	2	by a subscript in the range 0 through 4. The
D	2	first element in an array always has a sub-
E	4	script of 0.

4.7.2 In Two Dimensions

ARRAY(row, column) or A(I,J)

This must be reduced to the form A(G), where G is a function of I and J; that is, A(I,J) = A(G). Consider the diagram

	J = 0	1	2
I = 0	0	5	10
1	1	6	11
2	2	7	12
3	3	8	13
4	4	9	14

The numbers along the outside edges of the box above are the 2-dimensional subscripts; the numbers inside the box are the linear subscripts. Thus each combination of I and J can be given a unique value, e.g., for I = 2 and J = 1 the element is 7.

Notice that for a constant I, increasing the value of J by one increases the value of the linear subscript by five. Similarly, for a constant J, increasing the value of I by one increases the linear subscript by one.

The array, above, has five rows and three columns, so two values can be defined:

IMAX = 5 and JMAX = 3

The total number of elements is IMAX * JMAX = 15. To generate the number G in any box, using the corresponding values of I and J, the formula is

G = I + IMAX * J or A(G)

which is equivalent to A(I+IMAX*J). The example of solving simultaneous equations, above, uses this algorithm for subscripts merely by replacing I, IMAX, and J with J, L, and K, respectively, so as to form the equation

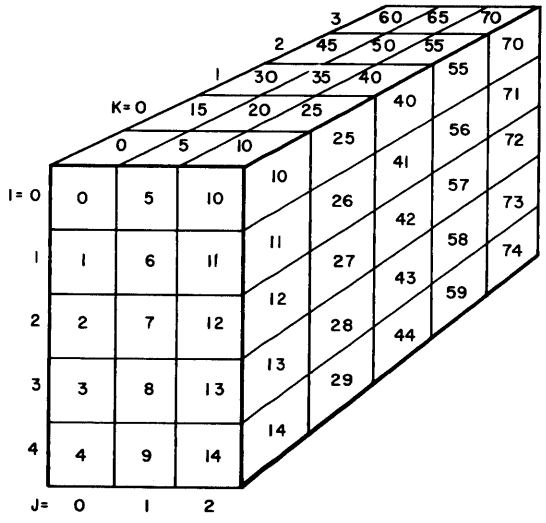
A(J + L * K)

Each element in a 2-dimensional array represents an area.

4.7.3 In Three Dimensions

ARRAY(row, column, plane) = A(I,J,K) = A(G)

In a 3-dimensional array, each array represents a volume.



This matrix has dimensions of five rows, three columns, and five planes; thus, IMAX = 5, JMAX = 3, and KMAX = 5. Each plane is numbered exactly as in the 2-dimensional example, except with the addition of 15 times K (with K = the number of planes back from the first) to each subscript in the first plane. For example,

Upper lefthand square, back one plane from the first = 15

$$I = 0,\ J = 0,\ K = 1;\ I + (IMAX*J) + (IMAX*JMAX*K) = 15 = G$$

or

$$A(0,\ 0,\ 1) = A(15)$$

4.7.4 In Four Dimensions

ARRAY (row, column, plane, cube) = A(I, J, K, L) = A(G)

Assign the values for IMAX, JMAX, KMAX; a method similar to the one used above yields

$$G = I + (IMAX*J) + (IMAX*JMAX*K) + (IMAX*JMAX*KMAX*L)$$

This process can be extended indefinitely to n-dimensions!

Example 1:

```
*FOR J=Ø,4; TYPE %2,!; FOR I=Ø,2; TYPE J+5*I

=  Ø=  5= 1Ø
=  1=  6= 11
=  2=  7= 12
=  3=  8= 13
=  4=  9= 14*
*
```

Example 2:

```
*1.Ø5 TYPE "ENTER 3 ROWS AND 4 COLUMNS OF NUMBERS."!
*1.1Ø FOR J=Ø,2; TYPE !; FOR K=Ø,3; ASK NO(J+3*K)
*1.15 SET MAX=NO(Ø)
*1.2Ø FOR J=Ø,2; FOR K=Ø,3; DO Ø2.ØØ
*1.25 TYPE !, "LARGEST NUMBER IS ", MAX; QUIT
*
*2.Ø5 IF (MAX-NO(J+3*K)) 2.1Ø; RETURN
*2.1Ø SET MAX=NO(J+3*K); RETURN
*
*GO
ENTER 3 ROWS AND 4 COLUMNS OF NUMBERS.

:Ø :5 :8 :9

:1 :2 :3 :4

:9 :8 :7 :6

LARGEST NUMBER IS =  9*
*
*GO
ENTER 3 ROWS AND 4 COLUMNS OF NUMBERS.

:A :B :C :D

:A :B :C :D

:A :B :C :D

LARGEST NUMBER IS =  4*
*
*GO
ENTER 3 ROWS AND 4 COLUMNS OF NUMBERS.

:A :B :C :D

:4 :3 :2 :1

:A :4 :C :2

LARGEST NUMBER IS =  4*
*
```

Example 3:

```
C-FOCAL,1969

01.02 TYPE !"ROUTINE TO SOLVE MATRIX EQ. AX=B FOR X"!
01.04 ASK "ENTER DIMENSION OF A, THEN
01.05 TYPE !"ENTER COEFF'S A(J,K)...A(J,N) AND B(J)"!
01.10 ASK L,!; SET N=L-1; SET I=-1
01.11 FOR K=0,N; SET R(K)=K+1
01.12 FOR J=0,N; TYPE !; FOR K=0,L; ASK A(J+L*K)
01.14 SET M=1E-6
01.16 FOR J=0,N; FOR K=0,N; DO 4
01.17 SET R[P]=0.
01.18 FOR K=0,L; SET A[P+L*K]=A<P+L*K>/M
01.20 FOR J=0,N; DO 5
01.22 SET I=I+1
01.23 IF (I-N) 1.14, 1.26, 1.14
01.26 FOR J=0,N; FOR K=0,N; DO 7
01.28 FOR K=0,N; TYPE !%2,"X("K,")  ",%8.05,X(K)
01.29 TYPE !!; QUIT

04.05 IF (R<J>) 0, 4.3, 4.1
04.10 IF (FABS[A<J+L*K>] - FABS[M]) 4.3;
04.20 SET M=A(J+L*K)
04.22 SET P=J; SET Q=K
04.30 RETURN

05.10 IF (J-P) 5.2,5.4,5.2
05.20 SET D=A(J+L*Q)
05.30 FOR K=0,L; SET A<J+L*K>=A[J+L*K]-A(P+L*K)*D
05.40 RETURN

07.10 IF (1E-6-FABS[A(J+L*K)]) 7.2; RETURN
07.20 SET X(K)=A(J+L*L)
*
*GO

ROUTINE TO SOLVE MATRIX EQ. AX=B FOR X
ENTER DIMENSION OF A, THEN
ENTER COEFF'S A(J,K)...A(J,N) AND B(J)
:3

:1 :2 :3 :4
:4 :3 :2 :1
:1 :4 :3 :2
X(=  0)  =   0.00000
X(=  1)  =-  1.00000
X(=  2)  =   2.00000
```

CHAPTER 5

ADVANCED FOCAL SYSTEMS

5.1 THE SYSTEMS

The user who has mastered the fundamental FOCAL system can appreciate the expanded FOCAL capabilities. Primarily, there are two ways to increase FOCAL's powers:

a. share FOCAL on a single computer with more than one person (multi-user segments)

b. expand a single user system to allow longer programs, improved accuracy, and graphic display (additional segments).

Table 5-1 describes all segments of FOCAL. Each is available on binary coded paper tape. A simple one-pass loading procedure adds these extra capabilities to the FOCAL system.

Table 5-1
FOCAL Segments

Segment Name	Function
Interpreter System	
FOCAL	The interpreter and teletype input/output handler.
FLOAT	Modified floating-point package.
INIT	The symbolic source for the initial dialogue program.
Multi-User Segments	
LIBRA	Allows multiple users (up to seven) to run and save FOCAL programs on an 8K PDP-8 with Disk.
QUAD	Allows multiple users (up to four) to share FOCAL on an 8K PDP-8.

Table 5-1 (Cont)
FOCAL Segments

Segment Name		Function
Additional Segments		
4-Word	Utility Package	Extended accuracy overlay to FLOAT (gives 10 digits).
8K	Utility Package	Allows one user to take advantage of an 8K PDP-8.
CLINE	Graphics Package	Permits scope to interact with FOCAL to display vectors, arcs and cursors.
PLOTR	Graphics Package	For use with an incremental plotter.
GRAPH	Graphics Package	For use with KV8/I.

FOCAL and FLOAT must be assembled and loaded together for all program configurations. They are separated for editing convenience.

5.2 MULTI-USER SEGMENTS

FOCAL can be shared simultaneously by more than one user by parcelling computer time among the various users. Such a system, referred to as time-sharing, permits one computer to serve several persons, allowing each user to feel he has the system all to himself. No detectable delays occur under normal operating conditions. With a very heavy workload, some users may detect only a slight delay, less than a second, in response to their commands to FOCAL.

The two multi-user systems associated with FOCAL are detailed below. Loading procedures for each are explained in Appendix E.

5.2.1 QUAD (Four-User FOCAL)

QUAD permits from one to four persons to use FOCAL simultaneously on an 8K PDP-8, -8/L, or -8/I Computer. Up to four Teletype consoles and appropriate PT08 or DC02 (for 8/L) communicating units are required.

5.2.2 LIBRA (Seven-User FOCAL)

LIBRA allows up to seven persons to use FOCAL efficiently on one 8K PDP-8, 8/I, or 8/L Computer. LIBRA requires, in addition to from one to seven Teletype consoles and appropriate PT08's or DC02's, at least one disk (RF08 or DF32). There are two versions of LIBRA available, depending on the user's disk system, i.e., and RF08 or DF32 version. A disk initialization routine, DISKIN, prepares the disk for use by LIBRA. (Refer to Appendix E.) With LIBRA user programs can be saved, retrieved, or deleted from the disk by library capabilities. Each program is assigned a name by the user, and a three-word command tells LIBRA what to do with that program. In all cases, the name of a program must be one to four characters. An accurate directory of saved (stored) user programs can also be listed by LIBRA.

5.2.2.1 LIBRA Commands - There are four LIBRA commands to perform the LIBRARY functions (where ↵ represents typing the RETURN key):

a. To save a program on the disk, use the command

LIBRARY SAVE name ↵

This will store the entire program on the disk for future use.

b. To call a stored program from the disk, the command

LIBRARY CALL name ↵

will bring the named program into the user's area for immediate operation. Execution begins at the first line of the called program, as if the user had typed a GO command.

c. If a program is stored and will no longer be needed, it may be removed from the disk by

LIBRARY DELETE name ↵

d. The user may wish to have LIBRA list the names of all the programs it has stored on the disk. Use

LIBRARY LIST ↵

to obtain the directory of stored program names. Note that this command cannot be abbreviated to L L. Note also that the LIST command destroys any program in the active user's area by an ERASE ALL.

While using these library commands to the disk, very few errors are possible. When saving a program (refer to subparagraph (a) above), LIBRA may find a program with an identical name in its directory list. Because each stored program must have a different name, the present program cannot be stored until the user gives it a new name. Also, the directory may be full already; therefore, this program cannot be stored.

After the program has been stored and the user wants to call or delete it from the disk, the only error possible is that LIBRA may find no such program name in its directory. Check your typing to be sure you spelled the program name correctly. The error codes for the above have the same format as normal FOCAL error code. They are listed in Appendix B.

5.2.2.2 Common Storage Function - LIBRA has swapping abilities which permit users to trade programs and data. The FCOM function allows a program to pass up to five to another program. It is used as follows:

FCOM (J,Z) stores element Z in array element J
FCOM (J) retrieves array element J

Index J has a range of $0 < J < 4$.

The FCOM function is explained fully in LIBRA System Specifications (DEC-08-AJCA-DL).

5.2.2.3 Limitation on FOCAL with LIBRA - When operating at full capacity, LIBRA places only a few limitations on FOCAL, none of which interfere with normal system operation. These limitations are:

a. The command for the high-speed reader is inoperable.

b. The FADC (analog to digital conversion) and FDIS (display) functions cannot be used.

c. None of the additional FOCAL segments (see Section 5.3) are compatible with LIBRA.

d. The use of the trace feature should be limited to prevent delaying execution of other users' commands. Trace only as few characters as necessary. To stop a program while using the trace feature, it may be necessary to depress CTRL/C more than once.

e. The search character of the MODIFY command is echoed.

LIBRA is described more fully in LIBRA System Specifications (DEC-08-AJCA-DL).

5.3 ADDITIONAL SEGMENTS

FOCAL's capabilities can be expanded to provide greater accuracy, larger user program, size, increased variable storage, and improved graphic display by loading an additional section of paper-tape to FOCAL. These tapes are called overlays and are provided in binary coded format. The powers and uses of the segments are described below. Loading procedures for all overlays are included in Appendix E.

5.3.1 Utility Package

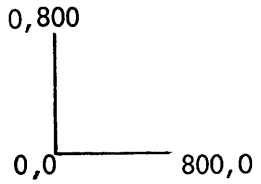
The utility package includes:

a. 4WORD - To increase FOCAL's accuracy to 10 digits for arithmetic operations. Because of this increased accuracy, there is a small decrease in the number of program variables that can be stored. Note that extended functions, trigonometric and exponential, are accurate to 6 places.

b. 8K - To increase program size, the 8K overlay activates an additional 4K of core memory. This permits significantly longer programs to be used with no decrease in the number of program variables that can be stored. The user's system must have 8K hardware for this segment. The 8K system has all the capabilties of 4K FOCAL, with the exception that the MODIFY command and other text changes do not erase variables. Only an ERASE command will clear the storage area in 8K FOCAL.

5.3.2 CLINE Graphics Package

By interfacing a PDP-8 system with a VC8 control unit which will handle a variety of display devices (34D, Tektronix 611) and loading the CLINE overlay, vectors and arcs are displayed for visual inspection. The coordinate systems vary for these instruments; programs in this manual are based on a 34D scope, the coordinate system of which is:



CLINE is especially useful for numerical control. CLINE can produce two basic types of lines: vectors and arcs; each type has a fundamental command string and the two can be combined at any time. CLINE will display a vector, the starting and ending points of which (X0, Y0 and X,Y, respectively) have been defined. To display a line, a DO 17 command must be incorporated into the program after each set of ending points has been assigned. The following four lines must be added to the program to display a line:

```
*16.2 SET P=X-XØ; SET Q=Y-YØ; SET R=FSQT(Q↑2+P↑2)
*16.3 SET Z=FDIS(6.3*R*C,P,Q,XØ,YØ,S/R)
*16.4 SET XØ=X; SET YØ=Y
*17.1 DO 16.2; SET Z=FDIS(R,P/R,Q/R,XØ,YØ,Ø); DO 16.4
```

Note that line 17.1 resets the values of X0 and Y0 to the previous ending points, X, Y, by a reference to line 16.4; thus, the end of one line automatically becomes the start of the next line. To start the next line from a different point, assign new values to both the starting and ending points.

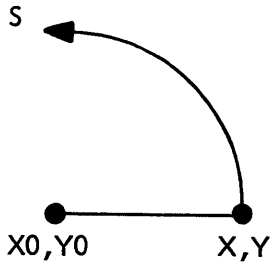
To display an arc, the following variables must be defined:

X0, Y0	center
X,Y	starting point ($-\sqrt{(X-X0)^2 + (Y-Y0)^2} = \text{radius}$)
C	circumference, where

C = .5 is a semicircle; C = 1 is a full circle

direction, where

S = +1 counterclockwise; S = -1 clockwise



C = .25

S = +1

After assigning values to the above variables and creating the routine to perform, increment, and restart the display pattern, a DO 16 command must be put in the program to perform the actual display function evaluation for each set of values and then project the results on the scope. The group 16 commands for arc generation are as follows:

```
*16.2 SET P=X-XØ; SET Q=Y-YØ; SET R=FSQT(Q↑2+P↑2)
*16.3 SET Z=FDIS(6.3*R*C,P,Q,XØ,YØ,S/R)
*16.4 SET XØ=X; SET YØ=Y
```

As with line drawing, the last values assigned to X and Y, now the starting point of the arc, become the values of X0 and Y0, which here define the center of the next arc to be drawn.

For example, the following routine will display an arc enclosed within a square on a 34D scope.

```
*1.1Ø SET C=.5; SET S=1; SET A=8ØØ
*1.2Ø SET XØ=Ø; SET YØ=Ø
*1.3Ø SET X=A; SET Y=YØ; DO 17; SET Y=A; DO 17; SET X=Ø; DO 17
*1.4Ø SET Y=Ø; DO 17
*1.5Ø SET XØ=4ØØ; SET YØ=4ØØ; SET Y=YØ; SET X=2ØØ; DO 16
*1.6Ø GOTO 1.2Ø
*16.2 SET P=X-XØ; SET Q=Y-YØ; SET R=FSQT(Q↑2+P↑2)
*16.3 SET Z=FDIS(6.3*R*C,P,Q,XØ,YØ,S/R)
*16.4 SET XØ=X; SET YØ=Y
*17.1 DO 16.2; SET Z=FDIS(R,P/R,Q/R,XØ,YØ,Ø); DO 16.4
*GO
```

Lines 1.1 and 1.5 defined a semicircle to be displayed inside the box described in lines 1.2, 1.3, and 1.4. When line 1.6 is reached, CLINE will first display the box, and then jump to the semicircle. The above program produces the pattern shown in Figure 5-1.

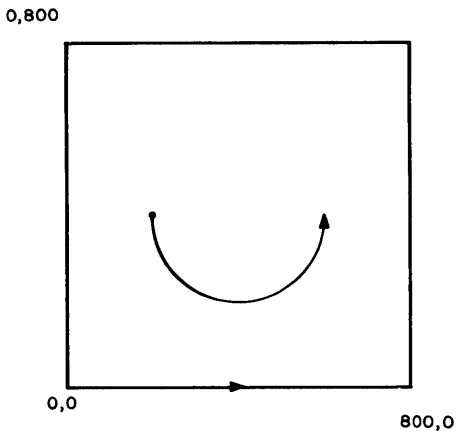


Figure 5-1 CLINE Example

5.3.2.1 Additional Graphics Segments - The CLINE function can be augmented for use with other display devices by merely running in an additional paper tape. (See Table 5-2.) A two portion graphics function is available for the appropriate KV 8 device with a joystick cursor (H306). The overlay, GRAPH, contains an X-Coordinate Cursor Read function, FCOM(0), and a Y-Coordinate Cursor Read function, FCOM(1). These two functions can be combined with the other GRAPH functions which are in the form FX( ), for complete graphic control, including display of vectors and arcs, use of the cursors, and erasure of the screen. The original CLINE FDIS functions can also be used with GRAPH.

An incremental plotter can be added to a graphics system by using the PLOTR overlay with CLINE (refer to Table 5-2). The three original CLINE functions (FDIS group) are available to the user. The PLOTR system permits the user to design and debug a picture on a scope before obtaining a hard copy.

5.3.3 Workable Overlay Combinations

These overlays can be combined with 4K FOCAL to produce a system with vastly increased powers that will optimize the individual user's system. Table 5-3 shows the workable FOCAL segment combinations. The initial dialogue and Disk Monitor System are included in these combinations to illustrate the many sets possible.

Table 5-2
CLINE Graphics Systems

Name	Function Call	Results	Device
(none)	FDIS (X,Y)	points	VC8 and display LAB-8 (AX08)
CLINE	FDIS (R,P/R,Q/R,X,Y,0)	vectors	VC8 and display
	FDIS (6.3*R*C,P,Q,X,Y,S/R)	arcs	
	FDIS (, , ,X,Y,)	points	
CLINE and LAB-8 (manual patches)	same as CLINE	same as CLINE	LAB-8 (AX08)
CLINE and PLOTR	same as CLINE	same as CLINE	Incremental Plotter and VC8
CLINE and GRAPH	same as CLINE	same as CLINE	KV8 and displays
	FCOM (0)	value is cursor x coordinate	H306 joystick, KV8 and display
	FCOM (1)	value is cursor y coordinate	H306 joystick, KV8 and display
	group 31 called by interrupt bar	synchronizes x, y cursors with external events	KV8 and display
	FX (0,441,X,Y,X0,Y0)	vectors	
	FX (C*64,211,X0,Y0)	arcs	KV8 and displays
	FX (, 4, , , ,)	erases screen	
	FX (, 1400, , , ,)	displays cursor	

In addition to directing graphics instruments, FOCAL can direct various other instruments using special programs written by the user. The extended functions† option allows the user to tailor the FOCAL system to his own specifications.

Table 5-3
Allowable FOCAL Systems

1 - Must be loaded into field one
0 - Must be loaded into field zero
  - Cannot be accepted
Y - Command may be used if disk system is built
N - Command is illegal
* - Command is different

Binary Segment	Allowed Combinations and Subsets are Indicated by Entries in Vertical Columns	Minimum Hardware Required
FOCAL	0 0 0 0 1 1 1 1	4K
INIT (optional)	0 0 0 0	
4WORD	0 0 1 1	4K
8K	0 0	8K
QUAD or PENT (non-8/S)	0 0 0 0	8K/PT08s
LIBRA (non-8/S)	0 0	8K/PT08s/DF32
CLINE (optional)	0 0	Graphics Terminal
PLOTR (calcomp)	0 0	
GRAPH (KV 8/I)	0 0	
LIBRARY COMMAND (for Disk Monitor)	Y Y Y Y N N * *	DF32
FOCAL is always loaded first in the proper field.		

5.4 8K OVERLAY

The 8K overlay permits a single user to run FOCAL in a greater core area than the 4K configuration. For notes on running 8K FOCAL, see Appendix E, section E.5.3 and Illustration E.5.

To increase the size of acceptable FOCAL programs, the 8K overlay uses the upper 4K for storage of the user's source text. The maximum number of variables remains the same as for 4K FOCAL; therefore, it is possible to get a variables buffer overflow condition long before the text buffer overflows.

†See DECUS Document FOCAL-17, How to Write New Subroutines.

Load the overlay after answering the initial dialogue with the 4K version.

5.5 CURRENT FOCAL TAPES AND DOCUMENTS

The following program tapes and documents comprise the FOCAL 1969 software package currently offered by DEC as of December 12, 1969. This list is subject to revision at any time.

FOCAL-8 Manual	DEC-08-AJAD-D
FOCAL, 1969 + INIT (4K, INIT)	DEC-08-AJAE-PB
Listing (Includes Utility Overlays)	DEC-08-AJAE-LA
Utility Overlays for FOCAL, 1969 Tape (4WORD,8K)	DEC-08-AJ1E-PB
Advanced FOCAL Technical Specifications	DEC-08-AJBB-DL
Graphic Overlays for FOCAL, 1969 (CLINE, PLOTR, GRAPH)	DEC-08-AJ3E-PB
Listing (of above)	DEC-08-AJ3E-LA
Extended Functions for FOCAL, 1969 (REPLACE, REMOVE)	DEC-08-AJ4E-PB
Multi-user Overlays for FOCAL, 1969 (LIBRA.DF32, DISKIN.DF32)	DEC-08-AJ5E-PB
Listing (of above)	DEC-08-AJ5E-LA
Multi-user Overlays for FOCAL, 1969 (LIBRA.RF08, DISKIN.RF08)	DEC-08-AJ6E-PB
Listing (of above)	DEC-08-AJ6E-LA
Four User Overlay for FOCAL, 1969 (QUAD.PT08)	DEC-08-AJ7E-PB
Listing (of above)	DEC-08-AJ7E-LA
Four User Overlay for FOCAL, 1969 (QUAD.DC02)	DEC-08-AJ8E-PB
Listing (of above)	DEC-08-AJ8E-LA
"Figure Eight Plot", A FOCAL Tape (Runs on any configuration)	DEC-08-AJ0E-PA
"King of Sumeria", A FOCAL Tape	DEC-08-AJ9E-PA

All of the above articles may be purchased from the Program Library.

APPENDIX A

COMMAND AND OPERATION SUMMARY

Table A-1
Commands

Command	Abbreviation	Example of Form	Explanation
ASK	A	ASK X, Y, Z	FOCAL types a colon for each variable; the user types a value to define each variable.
COMMENT	C	COMMENT	If a line begins with the letter C, the remainder of the line will be ignored.
CONTINUE	C	C	Dummy lines
DO	D	DO 4.1	Execute line 4.1; return to command following DO command.
		DO 4.0 DO ALL	Execute all group 4 lines; return to command following DO command, or when a RETURN is encountered.
ERASE	E	ERASE	Erases the symbol table.
		ERASE 2.0	Erases all group 2 lines.
		ERASE 2.1	Deletes line 2.1.
		ERASE ALL	Deletes all user input.
FOR	F	For i=x,y,z;(commands) FOR i=x,z;(commands)	Where the command following is executed at each new value. x=initial value of i y=value added to i until i is greater than z.
GO	G	GO	Starts indirect program at lowest numbered line number.

Table A-1 (Cont)
Commands

Command	Abbreviation	Example of Form	Explanation
GO?	G?	GO?	Starts at lowest numbered line number and traces entire indirect program until another ? is encountered, until an error is encountered, or until completion of program.
GOTO	G	GOTO 3.4	Starts indirect program (transfers control to line 3.4). Must have argument.

IF	I	IF (X)Ln, Ln, Ln
		IF (X)Ln, Ln; (commands)
		IF (X)Ln; (commands)
			Where X is a defined identifier, a value, or an expression, followed by three line numbers.
			If X is less than zero, control is transferred to the first line number.
			If X is equal to zero, control is to the second line number.
			If X is greater than zero, control is to the third line number.

LIBRARY CALL	L C	LIBRARY CALL name	Calls stored program from the disk.
LIBRARY DELETE	L D	LIBRARY DELETE name	Removes program from the disk.
LIBRARY LIST	L L	LIBRARY LIST	Types directory of stored program names.
LIBRARY SAVE	L S	LIBRARY SAVE name	Saves program on the disk.
LINK	L	L	For disk monitor system; FOCAL types 4 locations indicating start and end of text area, end of variable list and bottom of push-down list.
LOCATIONS	L	L	For paper-tape system; types same locations as LINK.
MODIFY	M	MODIFY 1.15	Enables editing of any character on line 1.15 (see below).
QUIT	Q	QUIT	Returns control to the user.
RETURN	R	RETURN	Terminates DO subroutines, returning to the original sequence.

Table A-1 (Cont)
Commands

Command	Abbreviation	Example of Form	Explanation
SET	S	SET A=5/B*C;	Defines identifiers in the symbol table.
TYPE	T	TYPE A+B-C;	Evaluates expression and types out = and result in current output format.
		TYPE A-B, C/E;	Computes and types each expression separated by commas.
		TYPE "TEXT STRING"	Types test. May be followed by ! to generate carriage return-line feed, or # to generate carriage return.
WRITE	W	WRITE WRITE ALL	FOCAL types out the entire indirect program.
		WRITE 1.0	FOCAL types out all group 1 lines.
		WRITE 1.1	FOCAL types out line 1.1.

A.1 FOCAL OPERATIONS

A.1.1 Format

To set output format,	TYPE %x.y	where x is the total number of digits, and y is the number of digits to the right of the decimal point.
	TYPE %6.3, 123.456	FOCAL types: = 123.456
	TYPE %	Resets output format to floating point.
To type symbol table,	TYPE $	Other statements may not follow on this line

The user can switch the input device to the high-speed paper tape reader by typing an asterisk in the first position, or immediately following the line number in an indirect command. The following statements cause FOCAL to read a tape from the high-speed reader.

**	The second * was typed by the user. Input is from the high-speed reader until occurrence of next *. FOCAL types * for each line number read in from the reader.
1.10;	User typed 1.10*;. Input is taken from the high-speed reader until occurrence of next *

To switch back to the keyboard, the user types another *. If there is no tape in the high-speed reader, or when an end-of-tape condition is reached, FOCAL automatically switches back to keyboard input. This feature is useful for loading FOCAL programs and for inputting large amounts of data during execution of a FOCAL program. When the following statement is executed, FOCAL accepts four pieces of data from the high-speed reader.

```
*1.10*; FOR I=1,4; ASK HR(I)
*1.11*
*DO 1.10
:::::*
```

The user typed an asterisk after line 1.11 to return control to the keyboard.

A.1.2 MODIFY Operations

After a MODIFY command, the user types a search character, and FOCAL types out the contents of that line until the search character is typed. The user may then perform any of the following operations.

a. Type in new characters. FOCAL will add these to the line at the point of insertion.

b. Type a CTRL/L. FOCAL will proceed to the next occurrence of the search character.

c. Type a CTRL/BELL. After this, the user may change the search character.

d. Type RUBOUT. This deletes characters to the left, one character for each time the user strikes the RUBOUT key.

e. Type ←. Deletes the line over to the left margin, but not the line number

f. Type RETURN. Terminates the line, deleting characters over to the right margin.

g. Type LINE FEED. Saves the remainder of the line from the point at which LINE FEED is typed over to the right margin.

A.1.3 The Trace Feature

Special Character	Example of Form	Explanation
?	?...? or ?...	Those parts of the program enclosed in question marks will be printed out as they are executed. If only one ? is inserted, the trace feature becomes operative, and the program is printed out from that point until another ? is encountered, until an error is encountered, or until program completion

A.1.4 Special Characters

a. Mathematical operators:

↑	Exponentiation
*	Multiplication
/	Division
+	Addition
-	Subtraction

b. Control characters:

%	Output format delimiter	
!	Carriage return and line feed	
#	Carriage return	
$	Type symbol table contents	
( )	Parentheses	(mathematics)
[ ]	Square brackets	
< >	Angle brackets	
" "	Quotation marks	(text string)
? ?	Question marks	(trace feature)
*	Asterisk	(high-speed reader input)

c. Terminators:

	SPACE key (names)	(nonprinting)
	RETURN key (lines)	
	ALT MODE key (with ASK statement)	
,	Comma (expressions)	
;	Semicolon (commands and statements)	

A.2 FUNCTIONS

A.2.1 Mathematical Functions

Square Root	FSQT(x)	where x is a positive number or expression greater than zero.
Absolute Value	FABS(x)	FOCAL ignores the sign of x.
Sign Part	FSGN(x)	FOCAL evaluates the sign part only, with 1.0000 as integer.
Integer Part	FITR(x)	FOCAL operates on the integer part of x, ignoring any fractional part.
Random Number Generator	FRAN( )	FOCAL generates a random number.
†Exponential Function (e^x)	FEXP(x)	FOCAL generates e to the power x. (2.71828^x)
†Sine	FSIN(x)	FOCAL generates the sine of x in radians.
†Cosine	FCOS(x)	FOCAL generates the cosine of x in radians.
†Arc Tangent	FATN(x)	FOCAL generates the arc tangent of x in radians.
†Logarithm	FLOG(x)	FOCAL generates $\log_e(x)$.
Analog-to-Digital	FADC(n)	FOCAL reads from an analog-to-digital channel, the value of the function is that integer reading.

A.2.2 Scope Function

FDIS(x,y)	Displays x and y coordinates on scope and intensifies x-y point.

A.2.3 Additional Functions

FCOM( )	LIBRA common storage function.
FX( )	For KV 8 (GRAPH)
FNEW	User defined function. Refer to DEC-08-AJBB-DL, Advanced FOCAL Technical Specifications.

†These are known as extended functions.

APPENDIX B

ERROR MESSAGES

Error messages are typed in the following format:

?nn.nn@nn.nn (error code @ line number)

Table B-1
FOCAL Error Messages †

Code	Meaning
?00.00	Manual start given from console
?01.00	Interrupt from keyboard via CTRL/C.
?01.40	Illegal step or line number used.
?01.78	Group number is too large.
?01.96	Double periods found in a line number.
?01.:5	Line number is too large.
?01.;4	Group zero is an illegal line number.
?02.32	Nonexistent group referenced by 'DO'.
?02.52	Nonexistent line referenced by 'DO'.
?02.79	Storage was filled by push-down-list.
?03.05	Nonexistent line used after 'GOTO' or 'IF'.
?03.28	Illegal command used.
?04.34	Left of "=" in error in 'FOR' or 'SET'.
?04.52	Excess right terminators encountered.
?04.60	Illegal terminator in 'FOR' command.
?04.:3	Missing argument in display command.
?05.48	Bad argument to 'MODIFY'.
?06.06	Illegal use of function or number.
?06.54	Storage is filled by variables.
?07.22	Operator missing in expression or double 'E'.
?07.38	No operator used before parenthesis.
?07.:9	No argument given after function call.
?07.;6	Illegal function name or double operators.
?08.47	Parentheses do not match.

† For FOCAL, 1969 only.

Table B-1 (Cont.)
FOCAL Error Messages †

Code	Meaning
?09.11	Bad argument in 'ERASE'.
?10.:5	Storage was filled by text.
?11.35	Input buffer has overflowed.
?20.34	Logarithm of zero requested.
?23.36	Literal number is too large.
?26.99	Exponent is too large or negative.
?28.73	Division by zero requested.
?30.05	Imaginary square roots required.
?31.<7	Illegal character, unavailable command, or unavailable function used.

Table B-2
LIBRA Error Messages

Code	Meaning
?25.84	FCOM INDEX out of range
?30.71	Undefined library command
?30.<0	Bad argument or missing argument to library command.
?31.42	No such name in library directory.
?31.43	Attempt to enter a duplicate name in the directory.
?31.44	Library directory is full.

Table B-3
System Error Appendix

1. Excess right parenthesis, right brackets, greater than signs, or equal signs in a type command cause an infinitely long line of zero values to be typed.

2. No indication is given that the user has produced a mathematical overflow or underflow condition.

3. A rounding error can appear in certain instances. Example:

```
T  %5,-0.4        produces -1
T  %3.02,1.4142 produces -1.42
```

4. If the user continues to type on the keyboard while the program is making computations, physical evidence of the error is indicated by failure of the computer to echo characters as the user types. The material being typed in will not be entered to the computer's input buffer.

† For FOCAL, 1969 only.

5. Error ?01.00 could also occur when reading a paper tape program into the input buffer via the low-speed reader. Since the program is unable to stop the low-speed reader, if output hardware should be slower than input hardware, the program will be unable to empty the reader buffer as quickly as it is being filled. To prevent this type of error with long input tapes, carriage returns may be followed by some blank tape which is ignored by the input routines, but which will give the output routines time to catch up.

6. Outside of the FOCAL program, the remaining storage is used for text storage, variable storage, and push-down storage, in that order. The overflow condition will occur only when one of these lists exceeds the remaining storage. This could happen in the case of complex programs which have multiple levels or recursive subroutines.

NOTE

This storage allocation scheme permits flexibility in the trade off of text size, number of variables, and complexity of the program, rather than restricting the user to a fixed number of statements or characters, to a fixed number of subroutine calls, or to a limited number of variables.

APPENDIX C

ESTIMATING THE LENGTH
OF USER'S PROGRAM

FOCAL requires five words for each identifier stored in the symbol table, and one word for each two characters of stored program. This can be calculated by

$$5s + \frac{c}{2} \cdot 1.01 = \text{length of user's program}$$

where s = Number of identifiers defined

c = Number of characters in indirect program

If the total program area or symbol table area becomes too large, FOCAL types an error message.

FOCAL occupies core locations 1_8 through 3200_8 and 4600_8 through 7576_8. This leaves approximately 700_{10} locations for the user's program (indirect program, identifiers, and push-down list). The extended functions occupy locations 4600-5377. If the user decides not to retain the extended functions at load-time, there will be space left for approximately 1100_{10} characters for the user's program.

The following routine allows the user to find out how many core locations are left for his use.

```
*FOR I=1,300; SET A(I)=I
?06.54                                  (disregard error code)
*TYPE %4,I*5,"LOCATIONS LEFT"
=   705LOCATIONS LEFT*
```

A LOCATIONS command can be given with a paper-tape system to determine how much space remains in core for user programs and variables. Execution of this command causes FOCAL to print four octal numbers (core memory works on a base 8 number system) representing the following locations within core:

a. start of text buffer } space for user's program
b. end of text buffer }
c. end of variable list } space for storing variables assigned during program
d. bottom of push-down list } space for subroutines

The LOCATIONS command permits the user to optimize his available storage space and to determine program length. If an 8K paper-tape system is being used, the values of a and b point to field 1. Locations c and d always point to field 0. The LOCATIONS command, for example, can be used after the three possible initial dialogue responses to indicate how much core each allows the user.

Dialogue response	Yes/Yes	No/Yes	No/No
Locations	*L	*L	*L
	3206	3206	3206
	3217	3217	3217
	3217	3217	3217
	4617	5177	5377

To get another * in order to continue with FOCAL after it has printed the four locations, the paper-tape system user must put 5177 in location 7600. If this is neglected, a manual restart is necessary.

Disk Monitor System users also have a command that indicates storage allocation: LINK. The LINK command for the Disk Monitor System is more limited than the LOCATIONS command for the paper tape system. LINK must be used only to return to the disk monitor; it cannot be used arbitrarily to determine core allocation. LINK types out four locations, in the same fashion as the LOCATIONS command, but then types a period, indicating that control has been transferred to the Disk Monitor. A command to the disk must then follow.

When storing a program on the disk, the LINK command maximizes storage space by specifying the exact amount of memory that is filled by text, variables, and subroutines. The core locations printed out by LINK are used in calls to the disk monitor.

APPENDIX D

CALCULATING TRIGONOMETRIC FUNCTIONS IN FOCAL

Function	FOCAL Representation	Argument Range	Function Range
Sine	FSIN(A)	$0 \le \lvert A \rvert < 10\uparrow 4$	$0 \le \lvert F \rvert \le 1$
Cosine	FCOS(A)	$0 \le \lvert A \rvert < 10\uparrow 4$	$0 \le \lvert F \rvert \le 1$
Tangent	FSIN(A)/FCOS(A)	$0 \le \lvert A \rvert < 10\uparrow 4$ $\lvert A \rvert \neq (2N+1)\pi/2$	$0 \le \lvert F \rvert < 10\uparrow 6$
Secant	1/FCOS(A)	$0 \le \lvert A \rvert < 10\uparrow 4$ $\lvert A \rvert \neq (2N+1)\pi/2$	$1 \le \lvert F \rvert < 10\uparrow 6$
Cosecant	1/FSIN(A)	$0 \le \lvert A \rvert < 10\uparrow 4$ $\lvert A \rvert \neq 2N\pi$	$1 \le \lvert F \rvert < 10\uparrow 6$
Cotangent	FCOS(A)/FSIN(A)	$0 \le \lvert A \rvert < 10\uparrow 4$ $\lvert A \rvert \neq 2N\pi$	$0 \le \lvert F \rvert < 10\uparrow 440$
Arc sine	FATN(A/FSQT(1-A↑2)	$0 \le \lvert A \rvert < 1$	$0 \le \lvert F \rvert \le \pi/2$
Arc cosine	FATN(FSQT(1-A↑2)/A)	$0 < \lvert A \rvert \le 1$	$0 \le \lvert F \rvert \le \pi/2$
Arc tangent	FATN(A)	$0 \le A \le 10\uparrow 6$	$0 \le F < \pi/2$
Arc secant	FATN(FSQT(A↑2-1))	$1 \le A < 10\uparrow 6$	$0 \le F < \pi/2$
Arc cosecant	FATN(1/FSQT(A↑2-1))	$1 < A < 10\uparrow 300$	$0 < F < \pi/2$
Arc cotangent	FATN(1/A)	$0 < A < 10\uparrow 615$	$0 < F < \pi/2$
Hyperbolic sine	(FEXP(A)-FEXP(-A))/2	$0 \le \lvert A \rvert < 700$	$0 \le \lvert F \rvert \le 5*10\uparrow 300$
Hyperbolic cosine	(FEXP(A)+FEXP(-A))/2	$0 \le \lvert A \rvert < 700$	$1 \le F < 5*10\uparrow 300$
Hyperbolic tangent	(FEXP(A)-FEXP(-A))/ (FEXP(A)+FEXP(-A))	$0 \le \lvert A \rvert < 700$	$0 \le \lvert F \rvert \le 1$
Hyperbolic secant	2/(FEXP(A)+FEXP(-A))	$0 \le \lvert A \rvert < 700$	$0 < F \le 1$
Hyperbolic cosecant	2/(FEXP(A)-FEXP(-A))	$0 < \lvert A \rvert < 700$	$0 < \lvert F \rvert < 10\uparrow 7$
Hyperbolic cotangent	(FEXP(A)+FEXP(-A))/ (FEXP(A)-FEXP(-A))	$0 < \lvert A \rvert < 700$	$1 \le \lvert F \rvert < 10\uparrow 7$
Arc hyperbolic sine	FLOG(A+FSQT(A↑2+1))	$-10\uparrow 5 < A < 10\uparrow 600$	$-12 < F < 1300$
Arc hyperbolic cosine	FLOG(A+FSQT(A↑2-1))	$1 \le A < 10\uparrow 300$	$0 \le F < 700$
Arc hyperbolic tangent	(FLOG(1+A)-FLOG(1-A))/2	$0 \le \lvert A \rvert < 1$	$0 \le \lvert F \rvert < 8.31777$
Arc hyperbolic secant	FLOG((1/A)+FSQT((1/A↑2)-1))	$0 < \lvert A \rvert \le 1$	$0 \le F < 700$
Arc hyperbolic cosecant	FLOG((1/A)+FSQT((1/A↑2)+1))	$0 < \lvert A \rvert < 10\uparrow 300$	$0 \le \lvert F \rvert < 1400$
Arc hyperbolic cotangent	(FLOG(X+1)-FLOG(X-1))/2	$1 < A < 10\uparrow 616$	$0 \le F < 8$

APPENDIX E

LOADING PROCEDURES

E.1 LOADERS

E.1.1 Read-In Mode (RIM) Loader

The RIM Loader is a program used to load the Binary Loader. The RIM Loader must be toggled into memory using the switches on the computer console.

To load the RIM Loader, follow the procedure below.

a. Check to see if the RIM Loader program is in memory correctly by examining the following locations for the appropriate instructions (contents).

	Instruction	
Location	33 ASR Reader	High-Speed Reader
7756	6032	6014
7757	6031	6011
7760	5357	5357
7761	6036	6016
7762	7106	7106
7763	7006	7006
7764	7510	7510
7765	5357	5374
7766	7006	7006
7767	6031	6011
7770	5367	5367
7771	6034	6016
7772	7420	7420
7773	3776	3776
7774	3376	3376
7775	5356	5357
7776	0000	0000

b. If the instruction in any location does not agree with the above list, deposit the correct instruction into that location.

E.1.2 Binary Format (BIN) Loader

The BIN Loader is a program used to load FOCAL into memory. The BIN Loader tape is loaded by the RIM Loader as explained below.

The BIN Loader is loaded into locations 7612-7616, 7626-7752, and 7777, with its starting address at location 7777. For a detailed description of the BIN Loader, refer to document DEC-08-LBAA-D.

To load the BIN Loader, follow the procedure below.

a. Check the RIM Loader for correctness, and correct if necessary.

b. Put Binary Loader tape in reader (always put leader-trailer code over reader head, never blank tape).

c. Turn reader ON.

d. Set Switch Register (SR) to 7756 (the starting address of the RIM Loader).

e. Depress LOAD ADDress switch on computer console.

f. Depress START switch on computer console.

g. Tape should begin reading in, if not, check the RIM Loader and start again at step a.

h. After program is read in, depress STOP switch on the computer console.

E.2 PAPER-TAPE SYSTEM

E.2.1 FOCAL Loading Procedure

The Binary Loader is used to load FOCAL. Check to see if the Binary Loader is in core. If location 7777 contains 5301, the Binary Loader is in core; if not, refer to Appendix E.

The procedure for loading FOCAL is detailed below.

Step	Procedure
1	Place the FOCAL binary tape in the tape reader.
2	Put 7777 (the starting address of the Binary Loader) in the SWITCH REGISTER.
3	Press the LOAD ADDress key.

To use the high speed paper tape reader, put 3777 in the SWITCH REGISTER.

4	Turn the Teletype to LINE.
5	Press the START key.

Step	Procedure
6	The tape will stop twice during loading because the program is loaded in two sections for additional checksum protection. After each halt, the contents of the accumulator (AC) should be 0; if the AC $\neq$ 0, reload the previous section of tape. If the AC is 0, press the CONTinue key and the tape will continue loading.
7	Place 0200 (the starting address of FOCAL) in the SWITCH REGISTER when the tape is completely loaded.
8	Press the LOAD ADDress key.
9	Press the START key. The initial dialogue will begin.
10	FOCAL is correctly loaded and ready for user input when it types an asterisk. If FOCAL is incorrectly loaded, reload the FOCAL tape starting with step 1 above.

The FOCAL loading procedure is illustrated in the flowchart (Figure E-1).

E.2.2 Restart Procedure

Two methods for restarting the system are outlined below.

a. The CTRL/C keys at any time[†]. FOCAL will type ?01.00 indicating a keyboard restart, and an asterisk on next line indicating it is ready for user input.

b. From the computer console:

Step	Procedure
1	Depress the STOP switch
2	Put 0200 in the SWITCH REGISTER
3	Depress the LOAD ADDress switch
4	Depress the START switch
5	FOCAL will then type *?00.00 indicating a manual restart, and an asterisk on the next line indicating it is ready for user input.

[†]CTRL/C indicates holding down the Control key while depressing the C key. This convention is used throughout the loading procedures.

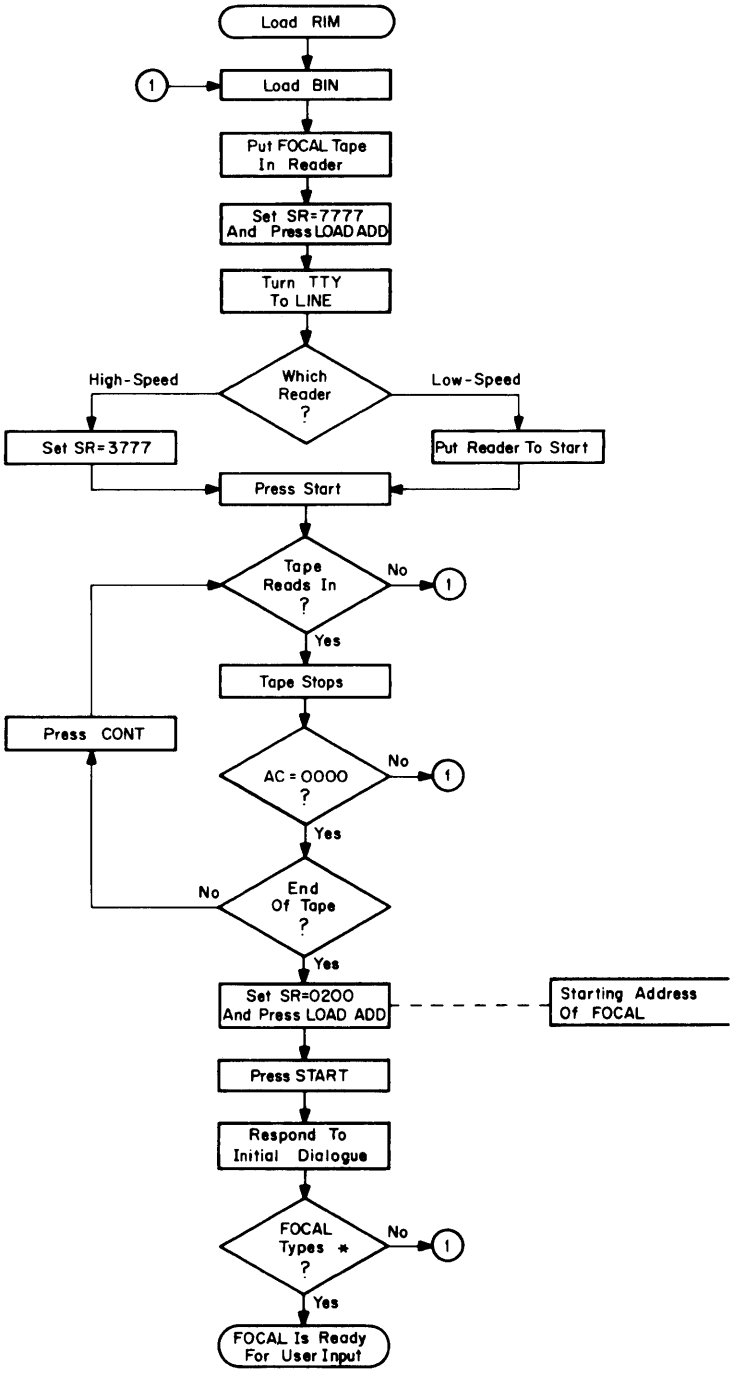


Figure E-1 FOCAL Loading Procedure

E.2.3 Saving FOCAL Programs

To save a FOCAL program on-line, proceed as follows:

Step	Procedure
1	Respond to * by typing WRITE ALL (do not depress the RETURN key).
2	Turn on low-speed tape punch
3	Type several @ signs to get leader tape (press the Shift, REPEAT, and P keys in that order; release in the reverse order).
4	Depress RETURN key

When the user's program has been typed and punched out

5	Type several more @ signs to get trailer tape.
6	Turn off tape punch

The user may now continue with another FOCAL program. The previous FOCAL program is still in the computer and waiting to operate on user input.

E.3 MULTI-USER SYSTEMS

E.3.1 LIBRA Loading Procedure

LIBRA is loaded after FOCAL. For this system, FOCAL is loaded into field 1. The binary loader, however, must be in field 0 (location 7777 should contain 5301). Do not load FOCAL's initial dialogue. The Disk WRITE LOCK switch must be off.

To load FOCAL into field 1:

Step	Procedure
1	Place the FOCAL binary tape in the reader.
2	Put 7777 in the Switch Register; put 1 in the DATA FIELD; put 0 in the INST FIELD.
3	Press LOAD ADDress key.

(Put 3777 in the Switch Register at this point if using the high-speed reader.)

4 Turn the teletype to LINE.

5 Press START key.

6 When the tape halts for the first time, remove it from the reader.

LIBRA goes into field 0:

7 Place the LIBRA binary tape in the reader.

8 Put 7777 in the Switch Register, 0 in both the DATA FIELD and INST FIELD.

9 Press the LOAD ADDress key.

To use the high-speed paper tape reader, put 3777 in the Switch Register.

Step	Procedure
10	Turn the teletype to LINE.
11	Press the START key.
12	Put the DISKIN binary tape in the reader.
13	Put 7777 (the starting address of the Binary Loader) in the Switch Register.
14	Press the LOAD ADDress key.

To use the high-speed paper tape reader, put 3777 in the Switch Register.

15 Turn the teletype to LINE.

16 Press the START key.

17 When the tape is loaded, put 0200 in the Switch Register.

18 Press the LOAD ADDress key.

19 Press the START key.

20 Answer DISKIN questions.

21 FOCAL/LIBRA is now ready to be shared by 7 users.

The program may be restarted at address 200.

To stop the system during operation, hold switch 11 (last switch on the right of the Switch Register) down until the system halts.

After running LIBRA or DISKIN, two locations in the Binary Loader must be restored before the loader can be used again. Load 1355 into location 7750, and load 5743 into location 7751.

E.3.2 DISKIN Loading Procedure

The disk initialize overlay, like LIBRA, has two versions, one to clear a DF32 Disk System and the other for an RF08 Disk System. DISKIN does not save the contents of the disk; because it clears a significant portion of the disk, use the overlay with care.

The overlay is kept as short as possible and can be quickly loaded with BIN. It will not destroy the FOCAL system that is currently in core. This allows the disk to be cleared at any time without re-loading the entire FOCAL system.

Step	Procedure
1	Load the overlay tape into field 0 using the binary loader. (Steps 12 to 16 of LIBRA Loading Procedure.)
2	Start at 0200 in field 0.
3	The overlay types FOCAL DISK (DF32) INITIALIZE? for a DF32 System and FOCAL DISK (RF08) INITIALIZE? for an RF08 System.
4	To clear the disk directory, type Y. Any other answer causes the program to go to step 9.
5	The overlay types the question NO. DISK SURFACE?, meaning the number of disks to be used.
6	Type a 1, 2, 3, or 4 for the number of disk surfaces to be used. Any other answer goes back to step 5.
7	The overlay then types the number of free blocks available for use in the form XXX FREE BLOCKS, where XXX is the octal number of programs that can be stored according to the number of surfaces selected.
8	The overlay then cleans the directory and confirms by typing DIRECTORY WRITTEN.
9	The overlay types SWAP AREA INITIALIZE?
10	If the swapping areas are to be initialized, type Y. Any other answer goes to FOCAL.

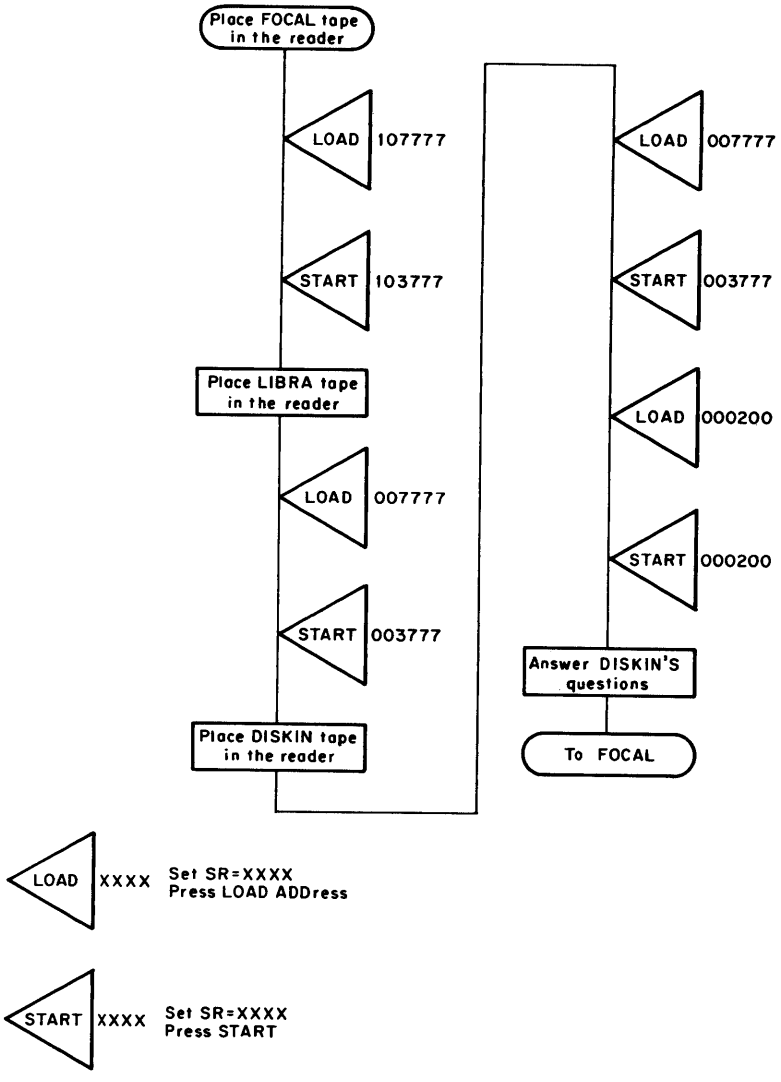


Figure E-2 LIBRA Loading Procedure

Step	Procedure
11	The overlay cleans the swapping areas and confirms success by typing SWAP AREAS WRITTEN and goes to FOCAL.
12	The message TO FOCAL is typed before entering the FOCAL system.

Typing a CTRL/C aborts the dialogue and transfers to FOCAL. The program will not respond to a CTRL/C during Steps 7, 8, and 11.

The message DISK WRITE ERROR indicates that the Disk is broken or WRITE LOCKED. Remedy the cause of the error and reload the initialize overlay, or, if core is intact, press CONTinue to try five more times.

E.3.3 QUAD Loading Procedure

Step	Procedure
1	Place the FOCAL binary tape in the reader.
2	Put 7777 in the Switch Register, put 1 in the DATA FIELD, and put the field number of the BIN Loader in the INST FIELD. (Since the 8/L has a maximum of 2 core fields, the INST FIELD will be 1 on the 8/L and whatever number is valid for other members of the PDP-8 family of computers.)
3	Press LOAD ADDress key.

(To use the high-speed reader, put 3777 in the Switch Register at this point.)

4	Turn the Teletype to LINE.
5	Press START key.
6	When the tape halts for the first time, remove it from the reader.
7	Place 0200 in the Switch Register.
8	Press the LOAD ADDress key.
9	Press the START key.
10	Place the QUAD binary tape in the reader.
11	Put 7777 in the Switch Register.
12	Press LOAD ADDress key.

(To use the high-speed paper tape reader, put 3777 in the Switch Register.)

13	Turn the Teletype to LINE.
14	Press the START key.

Step	Procedure
15	Place 0200 in the Switch Register.
16	Press the LOAD ADDress key.
17	Press the START key.

QUAD is ready for four users.

E.4 ADDITIONAL SYSTEM SEGMENTS

E.4.1 Utility Package Loading Procedure:

4WORD and 8K

Step	Procedure
1	Place the FOCAL binary tape in the tape reader.
2	Put 7777 (the starting address of the Binary Loader) in the Switch Register.
3	Press the LOAD ADDress key.

(To use the high-speed paper tape reader, put 3777 in the Switch Register.)

4	Turn the teletype to LINE.
5	Press the START key.
6	The tape stops twice during loading because the program is loaded in two sections for additional checksum protection. After each halt, the contents of the accumulator (AC) should be 0; if the AC $\neq 0$, reload the previous section of tape. If the AC is 0, press the CONTinue key and the tape will continue loading.
7	Place 0200 (the starting address of FOCAL) in the Switch Register when the tape is completely loaded.
8	Press the LOAD ADDress key.
9	Press START key. The initial dialogue will begin. Answer its questions.
10	FOCAL is correctly loaded and ready for user input when it types an asterisk. If FOCAL is incorrectly loaded, reload the FOCAL tape starting with Step 1 above.
11	Place the 4WORD or 8K binary tape in the tape reader.
12	Put 7777 (the starting address of the Binary Loader) in the Switch Register.
13	Press the LOAD ADDress key.

(To use the high-speed paper tape reader, put 3777 in the Switch Register.)

14 Turn the teletype to LINE.

15 Press the START key. The 4WORD or 8K tape will read in.

16 Put 0200 in the Switch Register.

17 Press the LOAD ADDress key.

18 Press the START key.

4WORD or 8K is now loaded.

E.4.2 Graphics Package Loading Procedure: CLINE, PLOTR, GRAPH

For a 34D scope:

Step	Procedure
1	Place the FOCAL binary tape in the tape reader.
2	Put 7777 (the starting address of the Binary Loader) in the Switch Register.
3	Press the LOAD ADDress key.

(To use the high-speed paper tape reader, put 3777 in the Switch Register.)

4	Turn the teletype to LINE.
5	Press the START key.
6	The tape stops twice during loading because the program is loaded in two sections for additional checksum protection. After each halt, the contents of the accumulator (AC) should be 0; if the AC $\neq$ 0, reload the previous section of tape. If the AC is 0, press the CONTinue key and the tape will continue loading. Remove tape from reader when it stops reading in.
7	Put 0200 in the Switch Register.
8	Press the LOAD ADDress key.
9	Press the START key. The initial dialogue will begin. Answer its questions.
10	FOCAL is correctly loaded and ready for user input when it types an asterisk. If FOCAL is incorrectly loaded, reload the FOCAL tape starting with Step 1 above.
11	Place the CLINE binary tape in the tape reader.
12	Put 7777 (the starting address of the Binary Loader) in the Switch Register.
13	Press the LOAD ADDress key.

(To use the high-speed paper tape reader, put 3777 in the Switch Register.)

14	Turn the Teletype to LINE.
15	Press the START key.

Step	Procedure
16	Place 0200 in the Switch Register.
17	Press the LOAD ADDress key.
18	Press the START key.

CLINE is ready for user input.

Load and start PLOTR after CLINE, in the same way, if using an incremental plotter.

Load and start GRAPH after CLINE, in the same way, if using a KV8 system.

E.5 DISK MONITOR SYSTEM

E.5.1 4K FOCAL

To use FOCAL with the DISK Monitor:

Step	Procedure
1	Build the Disk Monitor System on the disk.
2	Load FOCAL with the DISK system by using the LOAD command.

(Refer to DEC-D8-SDAB-D, Disk Monitor System, Programmer's Reference Manual.) Give a starting address of 200.

3	Complete the initial dialogue.
4	Press STOP; load address 7600, and press START.
5	Initialize the Disk by typing the following SAVE commands after Disk's period.

```
.SAVE START ! 4600-7577;200
.SAVE FOCAL ! 0-3377;
```

To run a FOCAL program, call FOCAL on the Disk.

```
.FOCAL
.START
?00.00          (FOCAL prints the error code for a console restart and an
*                asterisk to indicate it is ready to accept commands.)
```

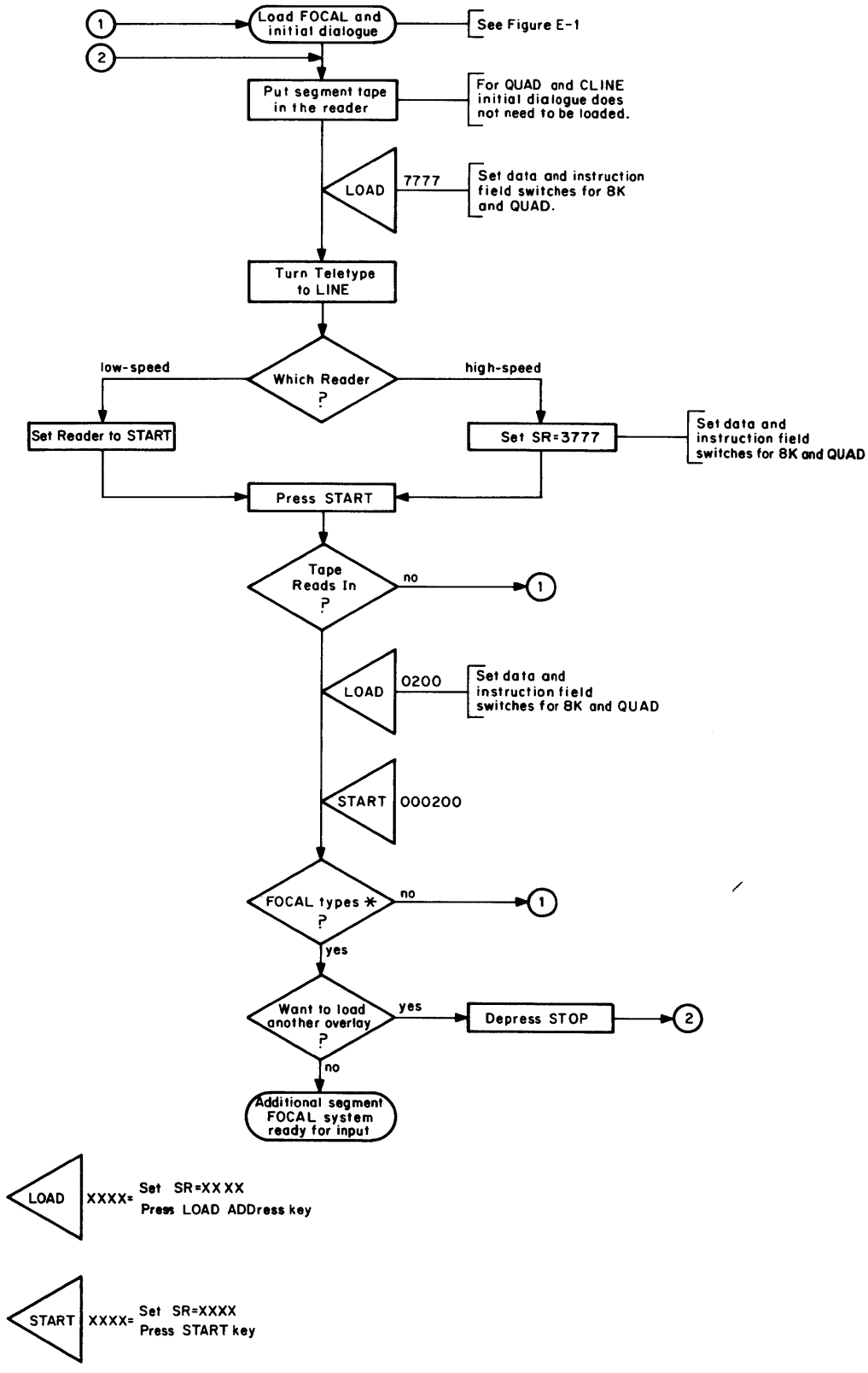


Figure E-3 Additional System Segments Loading Procedure

To save a program, it must be given a name. Note that page 0 is also being saved.

```
*LOCATIONS (returns command to Disk Monitor)      (User)
3206   (a)
3217   (b)                                        (FOCAL)
3217   (c)                                        (Disk Monitor)
4577   (d)

.SAVE (name); 0, (a) - (b)                        (User)
```

To continue to use the same program, add the command

```
.START
?00.00
*
```

After FOCAL types the error code and asterisk, the user can continue with the same (saved) FOCAL program.

To run a program that has been stored (as previously described), load address 7600, press START, and type the following routine.

```
.FOCAL
.CALL (name)
.START      (line feed will not occur)
?00.00      (FOCAL types the error code for a manual restart and an
*           asterisk to indicate it is ready to accept commands.)
```

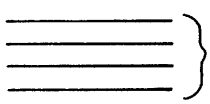
E.5.2 FOCAL Without Some Extended Functions

To use FOCAL without some of the extended functions, load FOCAL and start at address 7600, as in Paragraph E.5.1. Issue the following commands to the Disk:

```
.SAVE START ! 4600-7577;200
.SAVE INIT: 0,3200-4577;
.CALL INIT
.START
```

Answer YES to the initial dialogue.

```
*LOCATIONS
```



4 core locations generated by LOCATIONS command

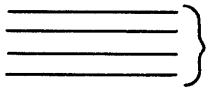
```
.SAVE FOCAL ! 0-3377;
```

To reinitialize and use the system without some of the extended functions, call the initial dialogue from the Disk:

```
.FOCAL
.CALL INIT
.START                (Initial dialogue begins)
```

To save sine and cosine only, type NO and YES to the dialogue's questions, following its termination with:

```
*LOCATIONS
```



4 core locations generated by LOCATIONS command

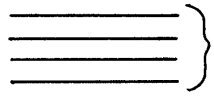
```
.SAVE STNY! 5200-7577; 200
```

This sine and cosine version is called by the commands:

```
.FNY
.STNY
?00.00
*
```

To delete all of the extended functions, call the initial dialgoue (as previously described), answer NO to both dialogue questions, and then type:

```
*LOCATIONS
```



4 core locations generated by LOCATIONS command

```
.SAVE STNN! 3400-7577;200
```

This version is called by the sequence:

```
.FNN
.STNN
?00.00
*
```

To call a program with deleted functions from the disk, use the correct startup commands to indicate the unwanted functions. For example, to use a version with no exponential function, use the sequence:

```
.FOCAL
.CALL NEXP          (previously created by the user)
.STNY
?00.00
*
```

A no cosine version requires:

```
.FOCAL
.CALL NCOS          (previously created by the user)
.STNN
?00.00
*
```

Note that the exponential function cannot be saved without the logarithmic function, nor can the cosine function be saved without the sine function.

E.5.3 8K FOCAL

To use 8K FOCAL on the disk, prepare the system as follows:

Step	Procedure
1	Build the disk.
2	Load FOCAL, using the binary loader in field 1.
3	Start at 0200.
4	Answer the initial dialogue.
5	Load the 8K overlay.

Save 8K FOCAL by issuing the following commands:

```
        *LOCATIONS
(a)     0100
(b)     0121
(c)     3217
(d)     XXXX
```

4 core locations generated by LOCATIONS command

```
.SAVE ST8K!     (d) - 7577;200
.SAVE FCL8!     0-3377;
.SAVE NUL8!     10100;10113   (to initialize program text area)
.SAVE NAME:     10100-(b);10113
```

To run a FOCAL program requiring 8K of memory, get FOCAL from the Disk.

```
.FCL8
.NUL8
.ST8K
?00.00
*
```

The above 3 commands to DISK are the appropriate starting sequence for a FOCAL program.

The save procedure for a finished 8K program is similar to 4K.

```
*ERASE
*LOCATIONS
```

4 core locations generated by LOCATIONS command

```
.SAVE NAME: 1 (a) -1 (b);10113
```

Add the following command to save a set of variables in field 0.

```
.SAVE DAT8:0;3200-(c);
```

The .SAVE DAT8 command stores a set of data (variables) located in field 0.

To set up a new program with a particular data set, type:

```
.FCL8
.CALL DAT8
.CALL NAM8
.ST8K
?00.00
*
```

Refer to DEC-08-SDAB-D, Disk Monitor System Programmer's Reference Manual for additional information.

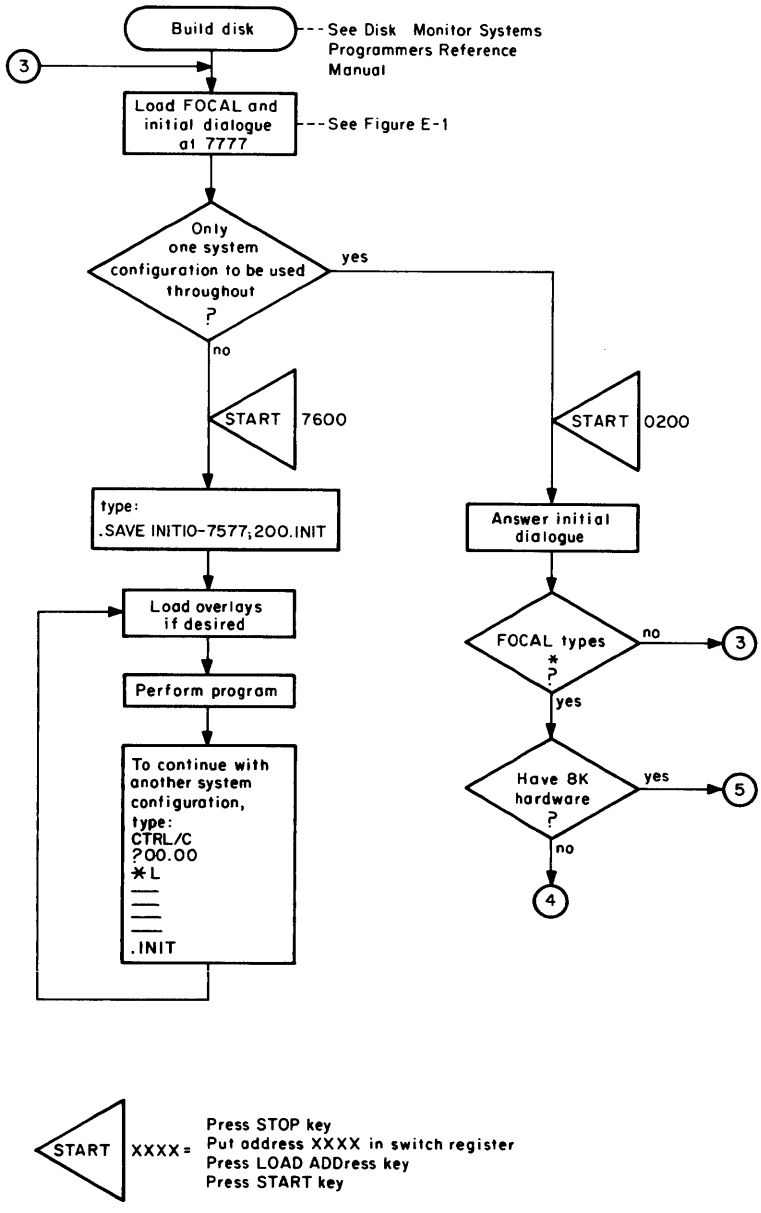


Figure E-4 Using FOCAL with Disk Monitor System (Sheet 1)

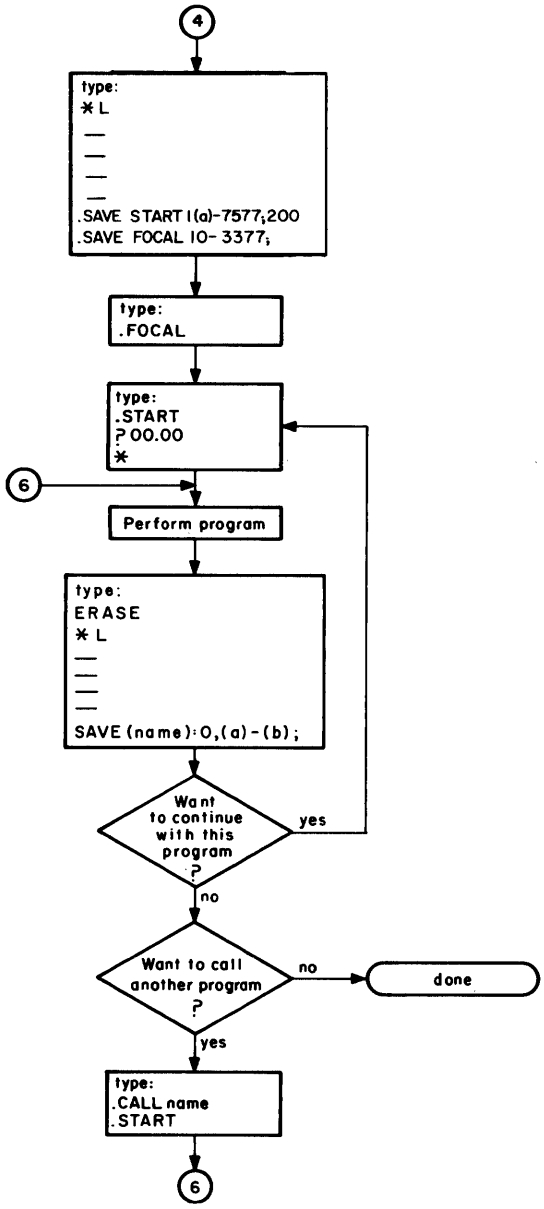


Figure E-4 Using FOCAL with Disk Monitor System
(Sheet 2)

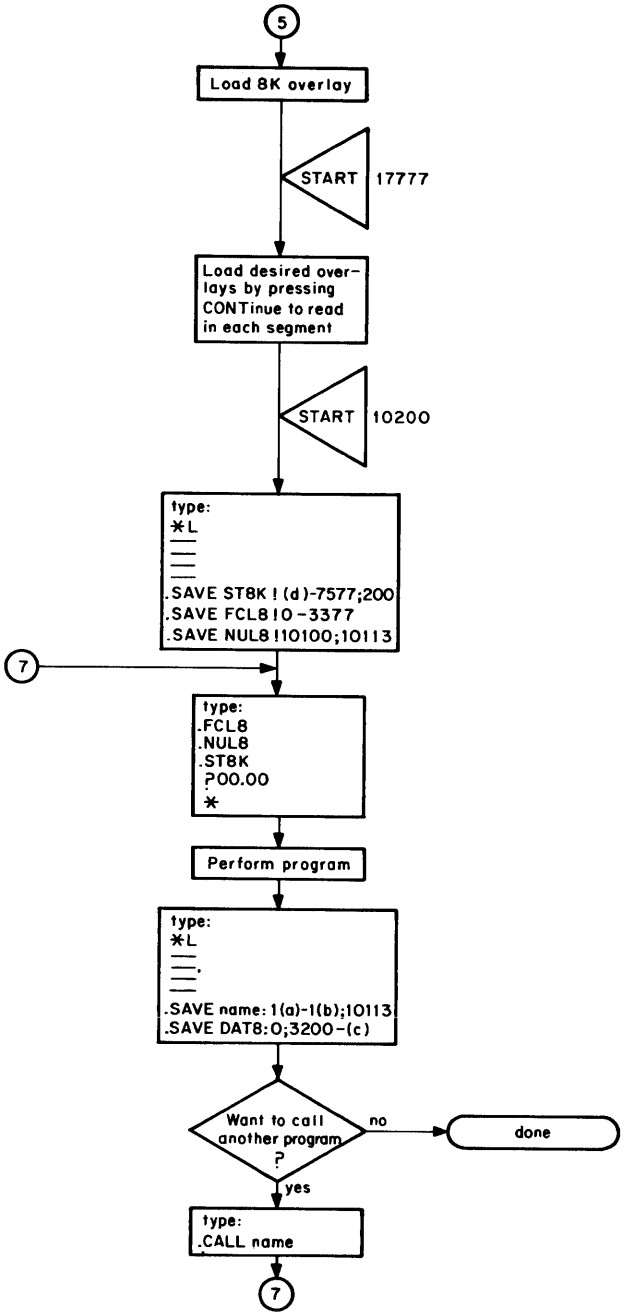
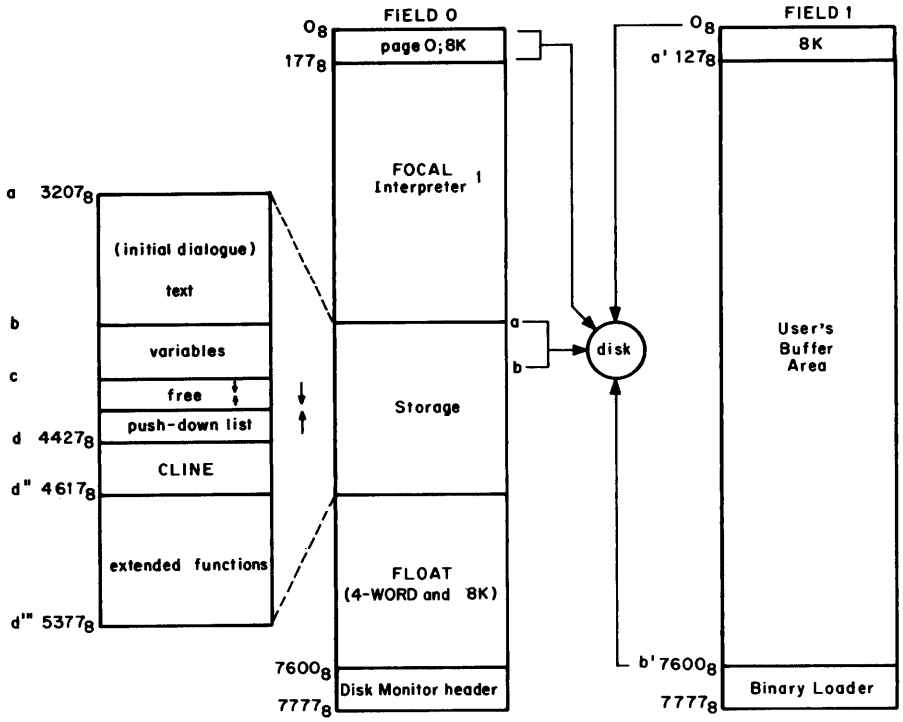


Figure E-4 Using FOCAL with Disk Monitor System
(Sheet 3)



a,b,c,d above point to the values indicated by the LOCATIONS command.

a. Start of text buffer.

b. End of text buffer.

c. End of variable list

d. Bottom of push-down list. Value depends on which functions are retained and which additional system segments are used.

a' and b' point to the corresponding locations in field 1.

Figure E-5 Core Map for FOCAL with Disk Monitor System

[1]The FOCAL interpreter is in field 1 when using QUAD.

APPENDIX F
HELPFUL PROGRAMMING SUGGESTIONS

To decrease program length, maximize available core area, and assist in preparing complex routines, the experienced programmer can implement the following suggestions:

a. All commands can be abbreviated to their first letter.

b. A string of commands, except WRITE, RETURN, MODIFY, QUIT, T $, and ERASE, can be combined on one line (up to 72 characters), with each command separated by a semicolon.

c. When creating a lengthy program, it is a good programming practice to leave free line numbers scattered throughout the body of the program. This will permit insertion of additional commands without complicated referencing routines. Remember that programs are executed sequentially by line number; consequently, an addition to the program placed physically at the end will be executed in turn. Line numbers must be in the range 1.01 to 31.99.

d. Some programs may require a keyboard response of YES or NO to a question asked during program execution. The answer typed to the question determines the next command to be executed (for example, in the initial dialogue). For this purpose, alphanumeric numbers are used in an IF statement to direct the execution.

```
*1.1 TYPE "DO YOU WANT A LINE?",!
*1.2 ASK "TYPE YES OR NO",ANS,!
*1.3 IF (ANS-ØYES)2.1,2.2,2.1
*
*2.1 QUIT
*2.2 TYPE "--------------",!
*2.3 GOTO 1.1
*GO
DO YOU WANT A LINE?
TYPE YES OR NO:YES

--------------
DO YOU WANT A LINE?
TYPE YES OR NO:NO

*
```

If the user types the answer YES, the identifier ANS is given the alphanumeric value of YES. When the IF statement is executed, the parenthetical expression YES-0YES equals zero, and the command at line 2.2 is executed. If the user types YES in answer to the ASK question, then when its alphanumeric value is substituted in the parenthetical expression, the expression will not equal zero and line 2.1 will be executed. Note that for YES/NO responses, the sign of the parenthetical expression is irrelevant; only its zero or non-zero value is of interest.

e. To avoid filling storage with the push-down list (error ?02.79) during long routines, it is helpful to limit the number of levels of nested expressions in a command. Use of abbreviations and limited number of variable names will maximize storage space. An FNEW function to increase variable storage is explained in DEC-08-AJBB-DL, Advanced FOCAL Technical Specifications.

INDEX

L

M

N

O

P

Q

R

S

T

U

W

Z

FOCAL 8
PROGRAMMING MANUAL
DEC-08-AJAD-D

READER'S COMMENTS

Digital Equipment Corporation maintains a continuous effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback – your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability.

_____

_____

_____

_____

_____

Did you find errors in this manual? _____

_____

_____

_____

_____

How can this manual be improved? _____

_____

_____

_____

_____

_____

DEC also strives to keep its customers informed of current DEC software and publications. Thus, the following periodically distributed publications are available upon request. Please check the appropriate box(s) for a current issue of the publication(s) desired.

☐ Software Manual Update, a quarterly collection of revisions to current software manuals.

☐ User's Bookshelf, a bibliography of current software manuals.

☐ Program Library Price List, a list of currently available software programs and manuals.

Please describe your position. _____

Name _____ Organization _____

Street _____ Department _____

City _____ State _____ Zip or Country _____

..... Fold Here .....

..... Do Not Tear - Fold Here and Staple .....

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Digital Equipment Corporation
Software Quality Control
Building 12
146 Main Street
Maynard, Mass. 01754

Digital Equipment Corporation
Maynard, Massachusetts

printed in U.S.A.