An Introduction to FOCAL

Preliminary Programming Manual

for

PDP-8

PDP-8/S

PDP-8/I

March 1968

Order No. DEC-08-AJAA-D from Program Library, Maynard, Mass.
Direct comments concerning this manual to Software Quality Control, Maynard, Mass.

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

CONTENTS

CHAPTER 1
AN INTRODUCTION TO FOCAL

1.1	General	1-1
1.2	Equipment Requirements	1-1
1.3	Loading Procedure	1-1
1.4	Restart Procedure	1-2
1.5	Saving FOCAL Programs	1-3

CHAPTER 2
FOCAL LANGUAGE

2.1	Arithmetic Operations	2-1
2.2	Output Formats	2-2
2.3	Identifiers	2-3
2.4	Subscripted Variables	2-4
2.5	Indirect Commands	2-4
2.6	Handling Text	2-5
2.7	Error Detection	2-6
2.8	Corrections	2-7

CHAPTER 3
FOCAL COMMANDS

3.1	TYPE	3-1
3.2	ASK	3-2
3.3	WRITE	3-2
3.4	SET	3-3
3.5	ERASE	3-3
3.6	GO	3-3
3.7	GOTO	3-4
3.8	DO	3-4
3.9	IF	3-5
3.10	RETURN	3-6
3.11	QUIT	3-6
3.12	COMMENT	3-6
3.13	FOR	3-6

CONTENTS (cont)

3.14	MODIFY	3-8
3.15	Using the Trace Feature	3-9
3.16	Functions	3-9
	APPENDIX A FOCAL COMMAND SUMMARY	A-1
	APPENDIX B ASK/TYPE CONTROL CHARACTERS	B-1
	APPENDIX C ERROR DIAGNOSTICS	C-1
	APPENDIX D ESTIMATING THE LENGTH OF USER'S PROGRAM	D-1
	APPENDIX E LIMITATIONS AND RESTRICTIONS OF PRELIMINARY VERSION	E-1

ILLUSTRATIONS

1-1	FOCAL Loading Procedure	1-2

CHAPTER 1
AN INTRODUCTION TO FOCAL

1.1 GENERAL

FOCAL (for FOrmula CALculator) is an on-line, conversational, service program for the PDP-8 family of computers, designed to help scientists, engineers, and students solve numerical problems. The language consists of short imperative English statements which are relatively easy to learn. Mathematical expressions are typed, for the most part, in standard notation. No previous programming experience is needed either to understand this manual or to use FOCAL at the Teletype console. However, the best way to learn the FOCAL language is to sit at the Teletype and try the commands, starting with the examples given in this manual.

1.2 EQUIPMENT REQUIREMENTS

4K PDP-8/I, PDP-8, or PDP-8/S computer with ASR 33 Teletype.

1.3 LOADING PROCEDURE

The Binary Loader is used to load FOCAL. Check to see if the Binary Loader is in core. If not, refer to the Binary Loader manual (DEC-08-LBAA-D), and if needed, the RIM Loader manual (DEC-08-LRAA-D).

The procedure for loading FOCAL is itemized below.

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 tape reader, set bit 0 of the SWITCH REGISTER to 0.

4. Press the START key.
5. The tape will stop loading at about midpoint because the program is loaded in two sections for additional checksum protection. After the halt, the contents of the accumulator (AC) should be 0, otherwise reload the previous section of tape. If the AC is 0, press the CONTINUE key and the tape will continue loading.
6. Place 200 (the starting address of FOCAL) in the SWITCH REGISTER when the tape is completely loaded.
7. Press the LOAD ADDress key.
8. Press the START key.
9. FOCAL is correctly loaded and ready for user input when it types *?00.00 followed by an asterisk on the next line. If, FOCAL is incorrectly loaded, then *?02.16 is typed out. Reload the FOCAL tape starting with step 1 above.

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

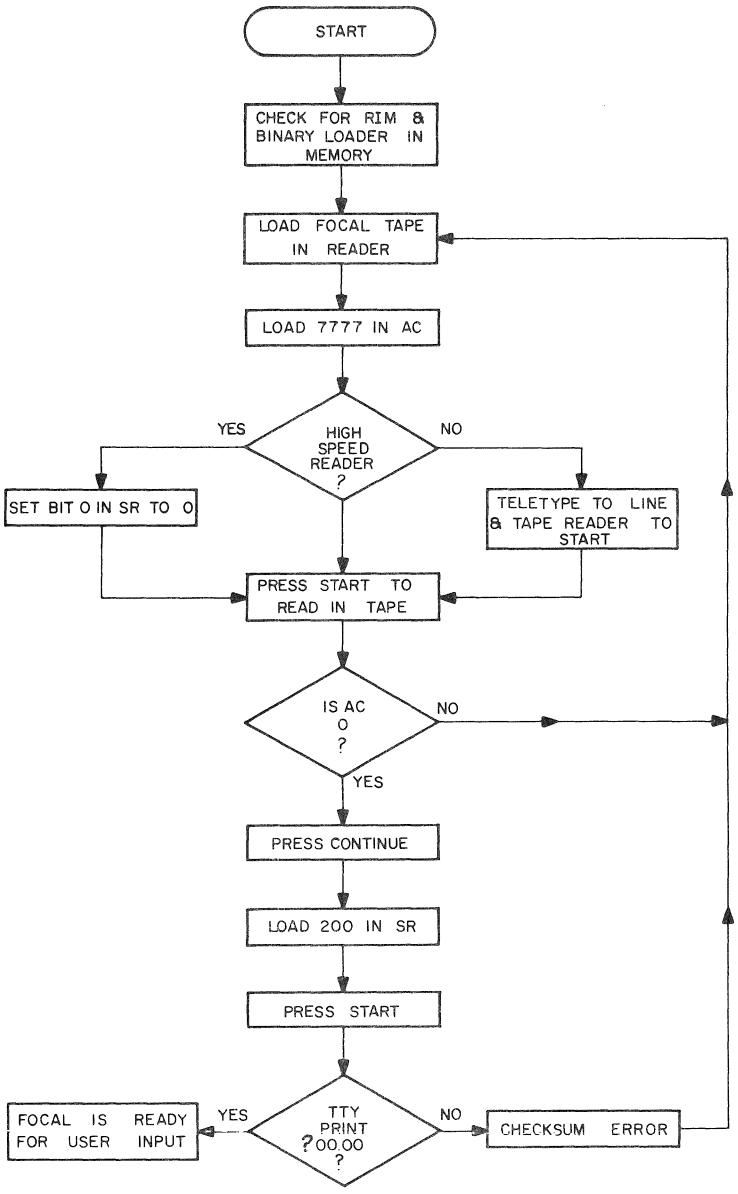


Figure 1-1 FOCAL Loading Procedure

1.4 RESTART PROCEDURE

The two methods of restarting the system are outlined below.

1. Type the CTRL/C (control and C) keys simultaneously at any time.
2. a. Put 200 in the SWITCH REGISTER
   b. Press the LOAD ADDress key
   c. Press the START key

d. The program will then type *?00.00 indicating a manual restart, and an asterisk on the next line indicating it is ready for user input.

1.5 SAVING FOCAL PROGRAMS

To save a FOCAL program, the user should

1. Respond to * by typing WRITE ALL
2. Turn on tape punch
3. Type several @ signs to get leader tape
4. Press RETURN (carriage return) key

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

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

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

CHAPTER 2
FOCAL LANGUAGE

After being loaded, FOCAL types

```
*$?00.00
*
```

The user may then type commands. Commands may be either direct or indirect. Direct commands such as

```
*TYPE 2+4↵
=+6*
```

where ↵ indicates a carriage return and * indicates that FOCAL is ready to accept another command from the user, are performed by FOCAL immediately. FOCAL types an equal sign, followed by the result.

The number of digits in the result can be specified by the user. Scientific or floating-point output may also be specified. FOCAL calculates to six significant digits. Exponents are computed to ±619. For example, to compute $9,999,999^{300}$ the user would type

```
*TYPE 9999999 t 300↵
```

and FOCAL types

```
=+0.9571895E+567*
```

Several statements may be typed on a single line, separated by semicolons, like this:

```
*TYPE A+B+C; SET D=5; DO 1.21↵
```

2.1 ARITHMETIC OPERATIONS

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

	Math Notation	FOCAL	
t Exponentiation	3^3	3t3	(Power must be positive integer)
* Multiplication	3·3	3*3	
/ Division	3÷3	3/3	
+ 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 in the list above. That is, exponentiation is done first followed by multiplication and division followed by addition and subtraction. Expressions with the same precedence 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,

```
*TYPE (A+B)* [C+D] *<E+F>↵
=+.157600E+20
```

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>↵
=+11*
```

2.2 OUTPUT FORMATS

One of the results shown above was typed by FOCAL in floating-point format. This is one format in which FOCAL produces output, but the format may be changed if the user types

```
TYPE %x.yy
```

where x is the total number of digits to be output, and yy is the number of digits to the right of the decimal point. x and y are positive integers, and x cannot exceed 19 digits. For example, if the desired output format is

```
mm.nn
```

the user may type

```
*TYPE %4.02, 12.22*2.37↵
```

and FOCAL will type

```
=+28.96*
```

Notice that the format operator must be followed by a comma.

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

Format Command	Result
%6.01	=+67823.0
%5	=+67823
%8.03	=+67823.000

To revert to flaating-point format, the user types TYPE %, as shown below.

```
*TYPE %, 67823↵
=+0.6782300E+05*
```

If the specified output format is too small to contain the number, FOCAL types Xs, as shown below.

```
*TYPE %3, 67823↵
=+XXX*
```

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

```
*TYPE %7, 67823↵
=+␣␣67823*
```

where the symbols␣␣ indicate spaces.

2.3 IDENTIFIERS

Symbolic names consisting of one or two alphanumeric characters may be used in FOCAL commands. The first character must be a letter; however, the first character must not be the letter F. Identifiers may be defined by using the SET command.

```
*SET PI=3.14159; SET E=2.71828; SET A1=7↵
*
```

These values are stored by FOCAL in its internal symbol table, and may then be used in expressions, as shown below.

```
*TYPE %4, PI*21.1↑2, 3071.2*E↵
=+1398=+8348*
```

The value of an identifier can be changed by retyping the identifier and giving it a new value.

```
*SET E=4.78, SET A1=3↑2↵
*SET E=3.37↵
*
```

The user may request FOCAL to type out all of the identifiers he has defined, in the order of definition, by typing

```
*TYPE $↵
```

If FOCAL's symbol table contains the above defined symbols, FOCAL types

```
PI(00)=+0.3141590E+01
E@(00)=+0.3370000E+01    (E was redefined)
A1(00)=+0.9000000E+01
*
```

If an identifier consists of only one letter, an @ is inserted as a second character in the symbol table printout, as shown in the second line 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.

2.4 SUBSCRIPTED VARIABLES

FOCAL also allows identifiers, or variable symbols, to be further identified by subscripts in the range 0 through 99, which are enclosed in parentheses immediately following the identifier in the SET statement. A subscript may also be an expression which must be reducible by FOCAL to two digits.

```
*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.

2.5 INDIRECT COMMANDS

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 15.99. 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,

```
*1.01 SET A=1
*1.03 SET B=2
*1.05 TYPE 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 1.01.

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

```
*GOTO 1.03
*
```

FOCAL will start 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 former sequence.

```
*1.1 SET A=1; SET B=2↵
*2.1 TYPE A+B↵
*2.2 SET C=3; SET B=4↵
*3.1 DO 2.1↵
*3.2 TYPE A-B↵
*GO↵
=+3=+5*
```

When the DO command in this sequence is reached, the command TYPE A+B is performed and then the program returns to line 3.2.

The DO command can also cause FOCAL to jump to a group of commands and then return automatically to the normal sequence. In the previous example, it the user changed line 3.1 to

```
*3.1 DO 2.0
```

FOCAL will execute the commands at all lines numbered 2.n (i.e., lines 2.1 and 2.2) and then return automatically to the command following the DO.

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

```
*2.1 SET A=1; SET B=2↵
*3.1 TYPE B/C↵
*2.2 SET C=1.31*.29↵
*GO↵
=+5*
```

where line number 2.2 has been added and will be executed following line 2.1.

2.6 HANDLING TEXT

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, but the exclamation mark (!) causes a carriage return-line feed and the number sign (#) causes a carriage return to be inserted as text.

Text strings may be included in TYPE and SET commands, as shown in the following examples.

```
*1.21 TYPE "VALUE IS ZERO"↵
*1.22 SET A=125↵
*1.23 DO 1.21↵
*1.24 TYPE A↵
*GO↵
VALUE IS ZERO VALUE IS ZERO=+125*
```

When FOCAL encounters steps 1.21 and 1.23, VALUE IS ZERO is typed out. At step 1.24, = +125 is typed out. A carriage return may be used to terminate a text string; therefore, the following example is equivalent to line 1.21 in the example above.

```
*1.21 TYPE "VALUE IS ZERO↵
```

The following example shows how the exclamation mark is used to generate carriage return-line feed sequences within a text string.

```
*1.2 TYPE "ALPHA! BETA! GAMMA!↵
*DO 1.2↵
(FOCAL types)
ALPHA
BETA
GAMMA
*
```

To generate a carriage return within a text string, but without a line feed, the user inserts the number sign (#). These operations are useful for formatting output.

2.7 ERROR DETECTION

FOCAL checks for a variety of errors, and if an error is detected, types a question mark followed by an error code. A complete list of these error codes is shown in Appendix C. The group number of an error message indicates the class or general type:

?00	Manual restart from console
?01	Interrupt from keyboard via ↑C.
?02	Excessive number or illegal mathematical operation.
?03	Miscellaneous
?04	Format errors
?05	Function not loaded

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

The trace feature is also useful in program checking. Any part of an indirect statement or program may be enclosed in question marks, and when that part of the program is executed that portion surrounded by question marks will be printed out. If only one question mark is inserted, the trace feature becomes operative, and the program is printed out from that point until completion, or until another question mark is encountered.

2.8 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 the wrong character(s). For example,

```
*1.01 P\TYPE X-Y↵
*1.02 SET $=13\\\\ X=13↵
*
```

The left arrow (←) erases everything which appears to its left, including the line number.

```
*1.1 TYPE X-Y←
```

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↵
```

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 2.0↵
```

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.

CHAPTER 3
FOCAL COMMANDS

3.1 TYPE

The TYPE command is used to request that FOCAL compute and type out the result of an expression, the value of an identifier, or a text string. The result of a TYPE command is numeric, in floating-point format, or another format specified by the user.

```
*4.14 TYPE 8.1+3.2-(29.3*5)/2.5↑7
*4.15 TYPE (2.2+3.5)*(7.2/3)/59.1↑3
*
```

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

```
*9.19 TYPE %3.3, A1*2, E+2↑5, 2.51*81.1
```

The output format may be included in the TYPE statement as shown in the example above and as explained in Chapter 2.

The user may request a typeout of all identifiers which he has defined by typing

```
TYPE $
```

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.

```
*SET L=33;SET B=87;SET Y=55;SET C9=91
*TYPE $
L@(00)=+0.3300000E+02
B@(00)=+0.8700001E+02
Y@(00)=+0.5499999E+02
C9(00)=+0.9100000E+02
*
```

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

```
*1.02 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 ASR 33 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 general form,

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

When step 11.99 is encountered by FOCAL, it types a colon; the user then types a value or expression for the first identifier, followed by a comma or a space; 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,X↵
*DO 11.99↵
:5, :4, :3
*
```

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

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

```
*1.10 ASK "HOW MANY APPLES DO YOU HAVE?" APPLES↵
*DO 1.10↵
HOW MANY APPLES DO YOU HAVE?:25↵                    (user typed 25)
*
```

The identifier AP now has the value 25.

3.3 WRITE

The WRITE command, without an argument, causes FOCAL to write out all indirect statements which the user has typed. Indirect commands 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 2.0↵        (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 is stored in the user's symbol table, and that value will be substitituted for the identifier when the identifier is encountered in the program.

```
*3.4 SET A=2.55; SET B=8.05
*3.5 TYPE A+B
*GO
=+0.1060000E+02*
```

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

The 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 2.0          (deletes all group 2 lines)
*ERASE 7.11         (deletes line 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.

```
*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
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 15.99.

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. If a RETURN is encountered, control is returned to the statement following the GOTO. (See FOR and DO for other GOTO operations.)

```
*GOTO 1.01
```

3.8 DO

The DO command transfers control 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 then control is automatically transferred back to the statement following the DO command, as shown in this example.

```
*1.1 TYPE "X"
*1.2 DO 2.3; TYPE "Y"
*1.3 TYPE "Z"
*2.3 TYPE "A"
*GO
XAYZA
*
```

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

If the DO command is written without an argument, FOCAL executes the entire indirect program, just as it would for a GO command.

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

When a GOTO or IF command is executed within a DO subroutine, control is transferred to the line specified by the GOTO or IF command and that line is executed. If the line is in the group specified in the DO command, any following lines in that group are executed and then control is returned to the statement following the DO. If, however, the GOTO command within a DO subroutine transfers control to a line outside the DO subroutine, FOCAL executes only the line specified by the GOTO command and returns control to the statement following the DO.

In the following example, the DO statement in line 1.20 makes a subroutine out of line 1.10, so that ANOTHER LOOP is printed until A is greater than 5.

```
*1.10 TYPE "ANOTHER LOOP"!
*1.20 FOR A=1,1,5; DO 1.10
*DO 1.20
ANOTHER LOOP
ANOTHER LOOP
ANOTHER LOOP
ANOTHER LOOP
ANOTHER LOOP
ANOTHER LOOP
*
```

A simple example of how FOCAL executes a DO command follows.

```
*1.10 TYPE %5.2, 5/2+6*3↑2
*DO 1.10
=+ 56.49
*
```

3.9 IF

In order to provide for transfer of control after a comparison, we have adopted from FORTRAN the IF statement format. The normal form of the IF statement consists of the word IF, a space, a parenthesized expression, and three line numbers separated by commas, in order. The expression is evaluated; then 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 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.

```
*2.10 TYPE "LESS THAN ZERO"; QUIT
*2.30 TYPE "EQUAL TO ZERO"; QUIT
*2.50 TYPE "GREATER THAN ZERO"; QUIT
*IF (25-25)2.10,2.30,2.50
EQUAL TO ZERO
*
```

The IF statement may be shortened by terminating 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 this 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.

3.10 RETURN

The RETURN command is used to exit from a DO, GOTO, or GO subroutine. When control is under a group subroutine and a RETURN command is encountered, the program exits from its subroutine status and returns to the line number following the DO or GOTO 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 string 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 string after the semicolon is executed again. This process is repeated until A is greater than D, and then FOCAL goes to the next sequential line.

The identifier A must be a single variable. B, C, and D may all be either expressions, variables, or numbers. If comma and the value C are omitted, then it is assumed that the increment is one. If C,D is omitted, this 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.

In the two examples below, the FOR command is combined with a DO command.

Example 1:

```
*1.10 FOR X = 1,1,5;DO 2.0
*2.10 TYPE! "          " %3,"X " X
*2.20 SET A =X+100.000
*2.30 TYPE! "          " %5.2 "A " A
*ERASE
*GO
      X =+   1
      A =+100.99
      X =+   2
      A =+101.99
      X =+   3
      A =+102.99
      X =+   4
      A =+104.00
      X =+   5
      A =+105.00
      X =+   6
      A =+105.99
```

Example 2:

```
*1.10 FOR A=10,10,80;DO 2
*2.10 TYPE ! "IF A" %3 A; TYPE %5  " THEN A SQUARED"  A↑2,
*ERASE
*GO
IF A=+10  THEN A SQUARED=+   100
IF A=+20  THEN A SQUARED=+   400
IF A=+30  THEN A SQUARED=+   899
IF A=+40  THEN A SQUARED=+  1600
IF A=+50  THEN A SQUARED=+  2500
IF A=+60  THEN A SQUARED=+  3600
IF A=+70  THEN A SQUARED=+  4899
IF A=+80  THEN A SQUARED=+  6400
```

3.14 MODIFY

Frequently, a few characters in a particular line require changes. To facilitate this job, and to eliminate the need to replace the entire line, the user may use the MODIFY command. Thus, to modify characters in a line, one would type MODIFY 5.41 in order to modify the characters of line 5.41. This command is terminated by a carriage return whereupon the program waits for the user to type that character at 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:

1. To type in new characters in addition to the ones that have already been typed out.
2. To type a form-feed; this will cause the search to proceed to the next occurrence, if any, of the search character.
3. To 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.
4. To use the RUBOUT key to delete ane character to the left.
5. To type a left arrow (←) to delete the line over to the left margin.
6. To type a carriage return to terminate the line at that point; the text to the right will be removed.
7. To type a LINE FEED to save the remainder of the line.

The ERASE ALL and MODIFY commands are generally used only in immediate mode since 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.

```
*7.01 JACK AND BILL W$NT UP THE HALL
*MODIFY 7.01
  JACK AND B\JILL W$\ENT UP THE HA\ILL
*WRITE 7.01
7.01 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. Again a search was made for an A character. This was changed to I the line was typed out correctly.

CAUTION

When the MODIFY command is used to edit any part of the user's stored program, the user's symbol table is erased. If the user defines his symbols by means of indirect commands prior to their use in his program, this will cause no difficulty, as symbols are entered in the symbol table when the statements defining them are executed.

If the user has defined symbols in direct statements, and then uses a MODIFY command, his symbol table entries will be erased, and must be redefined.

3.15 USING THE TRACE FEATURE

As explained in Chapter 2, 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.

```
*1.1 SET A=1
*1.2 SET B=5
*1.3 SET C=3
*1.4 TYPE %1, ?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
*
```

3.16 FUNCTIONS

The functions are provided to give extended arithmetic capabilities and to give the potential for expansion to additional input/output devices. There are three basic types of functions of which two types are included in the basic FOCAL program. The first type contains interger part, sign part, and absolute value functions.

A function call always consists of four letters beginning with the letter F and followed by a parenthetical expression:

```
FSGN(A-B*2)
```

The extended arithmetic functions are loaded at the option of the user. They will consume approximately 800 locations of his program storage area. These arithmetic functions are adapted from the extended arithmetic functions of the three-word floating-point package and are fully described with their limitations in the pertinent document.

An unorthodox and greatly skewed distribution is provided in the basic package for a random number generator (FRAN). It 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 five functions now available.

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

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

```
*TYPE %.2FSQT(4)
=+02
*TYPE FSQT(9)
=+03
*TYPE FSQT(144)
=+12
*
```

The SQT function computes the square root of the expression within the parentheses.

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

The ABS function outputs the absolute or positive value of the number in parentheses.

```
*TYPE FITR(5.2)↵
=+05
*TYPE FITR(55.66)↵
=+55
*TYPE FITR(77.434)↵
=+77
*TYPE FITR(-4.1)↵
+-05
*
```

The ITR function outputs the integer part of a number.

```
*TYPE % FRAN( )↵
=-0.2500000E+00
*TYPE FRAN( )↵
=-0.6235351E+00
*
```

The RAN function generates a random number. (The examples are typed in floating-point format.)

Future versions of FOCAL will also perform the following functions:

FSIN	Sine
FCOS	Cosine
FATN	Arctangent
FLOG	Logarithm ($\log_e$)
FEXP	Exponent
FADC	Analog-to-Digital
FDIS	Displays y coordinate on scope and intensifies the x-y point.
FDXS	Displays x coordinate on scope
FNEW	Available as a user-defined function

APPENDIX A

FOCAL COMMAND SUMMARY

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 former sequence.
		DO 4.0	Execute all group 4 lines; return to former line number sequence, 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;TYPE i	Where i is a symbol defined iteratively and 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.
GOTO	G	G3.4	Starts indirect program or transfers control to line 3.4. Must have argument.
IF	I	IF (X)ln, ln, ln;	Where X is a defined identifier, a value, or the result of 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.
MODIFY	M	MODIFY 1.15	Enables editing of any character on line 1.15.
QUIT	Q	QUIT	Returns control to the user.

Command	Abbreviation	Example of Form	Explanation
RETURN	R	RETURN	Terminates GOTO subroutines, returning to next line number in the original sequence.
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 text. May be followed by ! to generate carriage return-line feed, or # to generate carriage return.
WRITE	W	WRITE	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.

FOCAL Operations

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

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 optional operations.

1. Type in new characters. FOCAL will add these to the line at the point of insertion.
2. Type a FORM FEED. FOCAL will proceed to the next occurrence of the search character.
3. Type a CTRL/BELL. After this, the user may change the search character.
4. Type RUBOUT. This deletes characters to the left, one character for each time the user strikes the RUBOUT key.
5. Type ← . Deletes the line over to the left margin, but not the line number when ← is not the last character on the line.
6. Type carriage return. Terminates the line, deleting characters over to the right margin.
7. Type LINE FEED. This saves the remainder of the line from the point at which LINE FEED is typed over to the right margin.

Summary of 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.
*Cosine	FCOS(x)	FOCAL generates the cosine of x.
*Arctangent	FATN(x)	FOCAL generates the arctangent of x.
*Logarithm	FLOG(x)	FOCAL generates $\log_e(x)$.
*Analog-to-Digital	FADC	FOCAL reads from an analog-to-digital channel, the value of the function is that integer reading.

Scope Functions

*FDIS	Displays y coordinate on scope and intensifies x-y point.
*FDXS	Displays x coordinate on scope.

*Not available in preliminary version of FOCAL.

APPENDIX B
ASK/TYPE CONTROL CHARACTERS

%	Format delimiter
"	Text delimiter
!	Carriage return and line feed
#	Carriage return only
$	Type the symbol table contents
SPACE	Terminator for names
,	Terminator for expressions
;	Terminator for commands
↵ (Carriage Return)	Terminator for lines

APPENDIX C

ERROR DIAGNOSTICS

ERROR DIAGNOSTICS OF FOCAL., 1968

```
*?00.00-MANUAL START FROM CONSOLE
*?01.00-INTERRUPT FROM KEYBOARD VIA CONTROL-C
*?02.28-GROUP NUMBER OR LITERAL TOO LARGE
*?02.;0-ILLEGAL STEP NUMBER
*?02.46-NONEXISTANT LINE REFERENCED BY 'DO'
*?02.61-NONEXISTANT GROUP REFERENCED BY 'DO'
*?02.07-BAD LINE NUMBER FORMAT
*?02.29-ILLEGAL COMMAND USED
*?02.67-BAD ARGUEMENT FOR 'MODIFY'
*?02.;7-ILLEGAL OR MISSPELLED FUNCTION NAME
*?02.44-LINE NUMBER LARGER THAN ALLOWED
*?02.24-KEYBOARD-INPUT BUFFER OVERFLOW
*?02.87-COMMAND/INPUT BUFFER EXCEEDED
*?02.46-IMAGINARY SQUARE ROOTS
*?02.80-DIVISION BY ZERO
*?02.;3-NUMBER TOO LARGE TO BE MADE AN INTEGER
*?03.50-IMPROPER STEP NUMBER
*?03.79-VARIABLE STORAGE EXCEEDED
*?03.10-BAD ARGUEMENT FOR 'ERASE'
*?03.79-EXPONENT NOT A POSITIVE INTEGER
*?03.42-LOG OF ZERO REQUESTED
*?04.93-DOUBLE PERIODS IN A LINE NUMBER
*?04.:2-MULTIPLE PERIODS IN A LINE NUMBER
*?04.12-BAD ARGUEMENT IN 'IF'COMMAND
*?04.39-ERROR TO LEFT OF EQUALS SIGN
*?04.53-EXCESS RIGHT PARENTHESIS
*?04.61-ILLEGAL CHARACTER IN 'FOR'
*?04.18-BAD ARG IN 'FOR', 'SET', OR 'ASK'
*?04.13-MISSING OPERATOR IN AN EXPRESSION
*?04.33-OPERATOR MISSING BEFORE PARENTHESIS
*?04.;0-FUNCTION NOT FOLLOWED IMMEDIATELY BY PARENS
*?04.;9-DOUBLE OPERATORS IN EXPRESSION
*?04.45-PARENTHESES DO NOT MATCH
*?04.13-ILLEGAL E-FORMAT ON INPUT OR LITERAL
*?05.60-ERROR IN 'FOR' COMMAND FORMAT
*?05.11-NO ARGUEMENT IN 'IF' COMMAND
*?05.;6-FUNCTION NOT LOADED INTO CORE
*?05.28-COMMAND NOT AVAILABLE
*
```

CUT ALONG DOTTED LINE AND POST NEAR THE KEYBOARD.

APPENDIX D
ESTIMATING THE LENGTH OF USER'S PROGRAM

In 4K systems, after FOCAL is loaded, approximately 1700_8 locations are available for storage of the user's indirect program and the user's symbol table.

FOCAL requires five words for each identifier stored in the symbol table, and one word for each two characters of stored program. This may 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 of program area and symbol table area become too large, FOCAL types the error message

?02.80

When other systems are loaded you may get a ?03.80 error message.

FOCAL occupies core locations $1\text{-}3300_8$ and $5300_8\text{-}7576_8$. This leaves approximately 1000_{10} locations for the user's program (indirect program, identifiers, and push-down list).

APPENDIX E
ERRATA AND LIMITATIONS

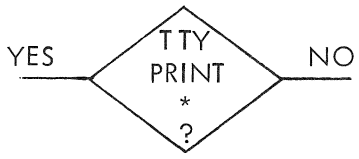
ERRATA

Chapter 1

Page 1-1, section 1.3, step 9 should be changed to read:

9. 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.

Page 1-2, Figure 1-1, the last decision symbol should be changed to read:



Chapter 2

Page 2-1, first paragraph should be changed to read:

After being loaded, FOCAL types

*

Page 2-1, third paragraph, fourth sentence should be changed to read:

For example, to compute 300 factorial, the user would type:

```
*1.1 SET A=1↵
*1.2 FOR I=1,300; SET A=A*I↵
*1.3 TYPE %, A↵
*GO↵
```

and FOCAL types

```
=+0.306051E+615*
```

Chapter 3

Page 3-11, tenth line from bottom of page should be changed to read:

FOCAL will also perform the following functions:

Appendix A

Page A-3, the footnote should be changed to read:

* These are called extended functions.

Appendix D

Page D-1, third paragraph should be changed to read:

If the total of ... types the error message

?03.79

The next sentence should be omitted.

LIMITATIONS

1. Identifiers must not begin with the letter F. Result, error diagnostic code ?04.16.

2. Subscripts over 99 should not be used. Example:

```
*SET A(101)=2
*TYPE $
A@(:)=+0.2000000E+01*
```

A subscript over 99 will be altered to some random character by the system.

3. When the MODIFY command is used, all identifiers previously set must be reset, as the command erases the symbol table contents.