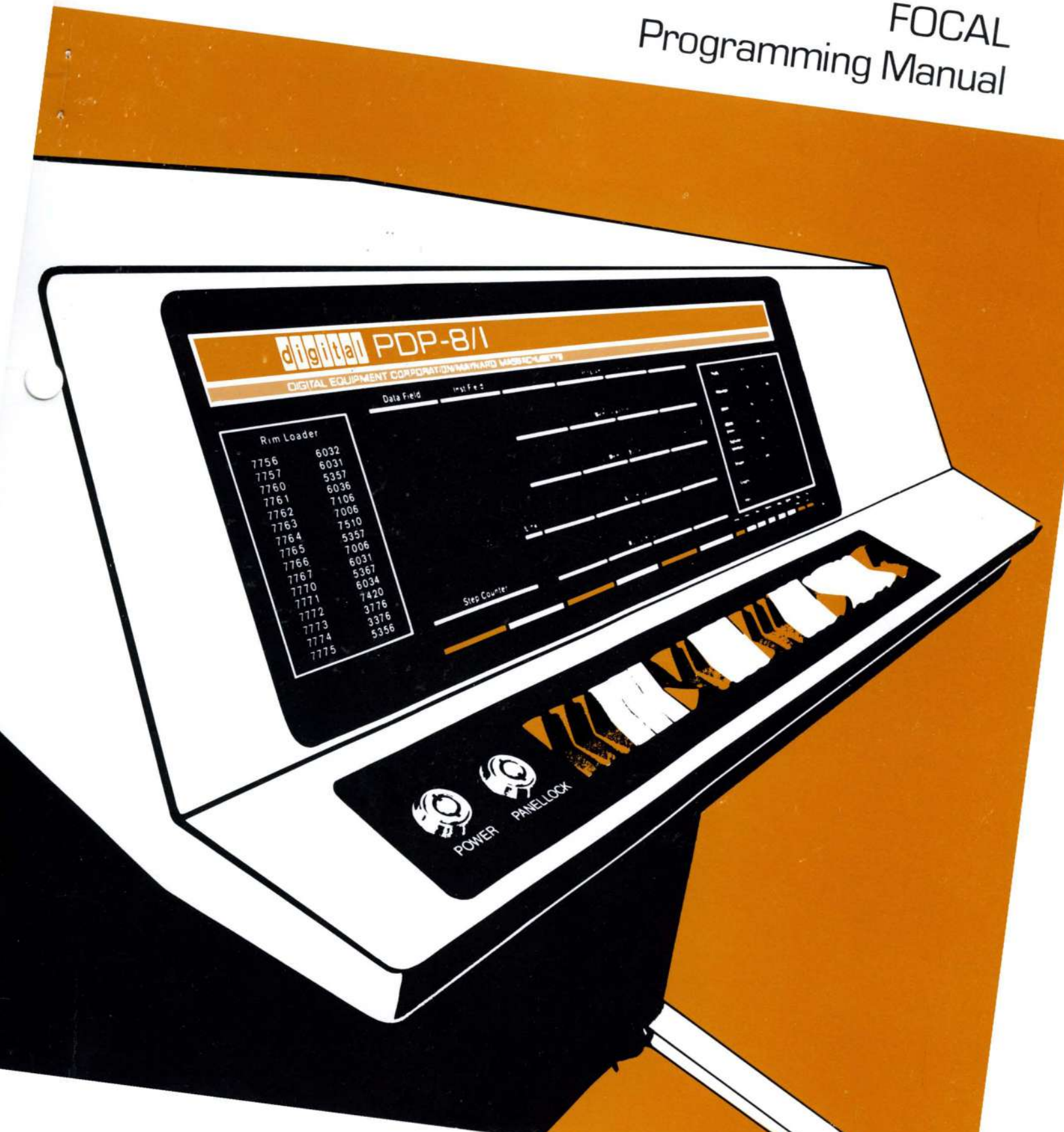
PDP-8/I

digital

FOCAL Programming Manual



FOCAL
Programming Manual
for
PDP-8
PDP-8/S
PDP-8/I
LAB-8
LINC-8

For additional copies Order No. DEC-08-AJAB-D from Program Library,
Digital Equipment Corporation, Maynard, Mass. Price $1.00

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

Printed in U.S.A.

CONTENTS

CONTENTS (cont.)

PREFATORY NOTE

The purpose of this manual is to introduce FOCAL to the scientific and engineering, and educational fraternities. Later, Digital will publish two specialized FOCAL manuals; one for professional scientists and engineers, the other for students.

CHAPTER 1
AN INTRODUCTION TO FOCAL*

FOCAL (for Formulating On-Line Calculations in Algebraic Language) 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. 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.

FOCAL puts the full calculating power and speed of the computer at your fingertips. FOCAL is used for simulating mathematical models, for curve plotting, for handling sets of simultaneous equations in n-dimensional arrays, and many other kinds of problems. Some of the kinds of problems that have been solved by FOCAL are described in Chapter 4, Sample Programs.

1.1 EQUIPMENT REQUIREMENTS

FOCAL operates on a 4K PDP-8/I, PDP-8, PDP-8/S, LAB-8, or LINC-8 computer with ASR 33 Teletype. Optional: Analog-to-Digital Converter (AF01A); Oscilloscope Display (VC8/I or 34D).

1.2 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 F.

The procedure for loading FOCAL is detailed below.

a. Place the FOCAL binary tape in the tape reader.

b. Put 7777 (the starting address of the Binary Loader) in the SWITCH REGISTER.

c. Press the LOAD ADDress key.

To use the high speed paper tape reader, set bit 0 of the SWITCH REGISTER to 0.

d. Press the START key.

e. The tape will stop twice during loading because the program is loaded in three 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.

f. Place 200 (the starting address of FOCAL) in the SWITCH REGISTER when the tape is completely loaded.

* FOCAL is a trademark of Digital Equipment Corporation.

g. Press the LOAD ADDress key.

h. Press the START key. The initial dialogue will begin. This is a question and answer sequence, with FOCAL asking questions and the user providing the answers. The first question offers an option. If you want a full explanation, press the RETURN key on your Teletype. The present version of FOCAL operates in 4K, and so states; future versions will include an option to utilize an additional 4K of memory.

i. 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 a above.

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

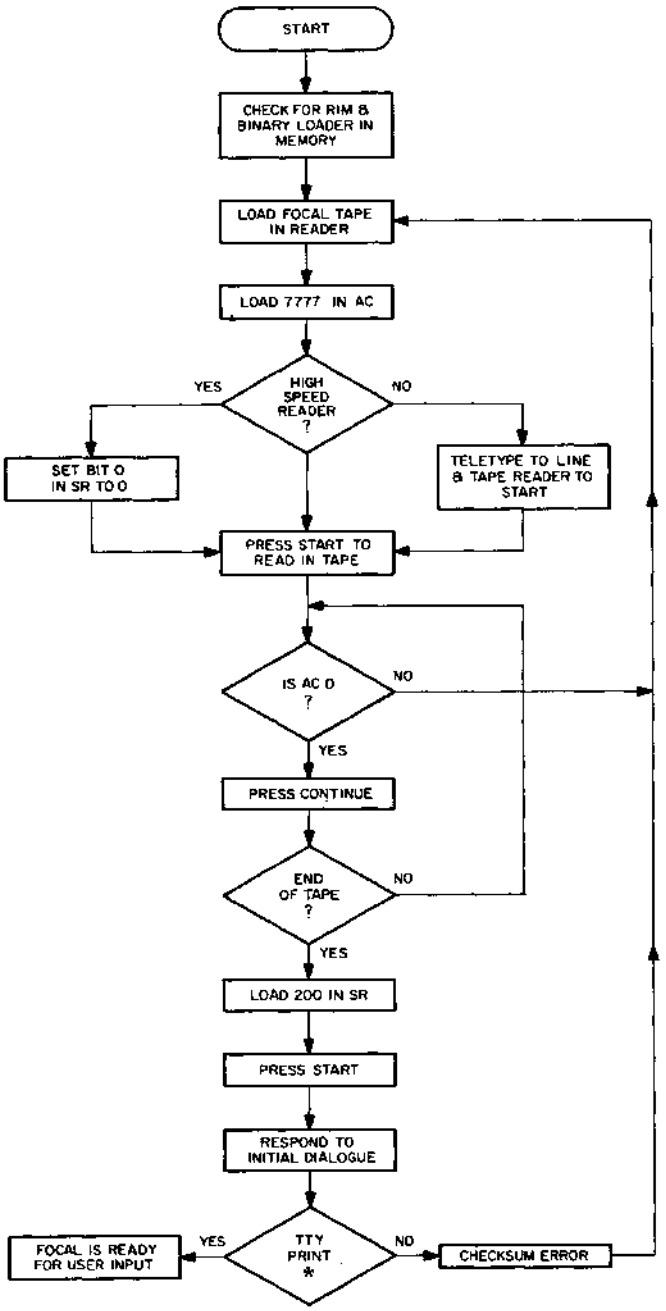


Figure 1-1 FOCAL Loading Procedure

1.3 RESTART PROCEDURE

Two methods for restarting the system are outlined below.

a. The CTRL/C keys at any time.[1] 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:

(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.

1.4 SAVING FOCAL PROGRAMS

To save a FOCAL program on-line, the user should

a. Respond to * by typing WRITE ALL (do not depress the RETURN key).

b. Turn on tape punch

c. Type several @ signs to get leader tape (press the Shift, REPEAT, and P keys in that order; release in the reverse order).

d. Depress RETURN key

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

e. Type several more @ signs to get trailer tape.

f. 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.

[1] CTRL/C indicates holding down the Control key while depressing the C key. This convention is used throughout this manual.

CHAPTER 2
FOCAL LANGUAGE

After the initial dialogue has been concluded, FOCAL types

```
*
```

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 SIMPLE 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
?Ø2.29
*TYPE 2++4
?Ø4.;9
*
```

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.

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 mm.nn

the user may type

```
*TYPE %4.Ø2, 12.22+2.37
```

and FOCAL will type

```
=+14.59*
```

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.

```
*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 types X's, as shown below.

```
*TYPE %3, 67823
=+XXX*
```

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 by 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 ±619. The largest number that FOCAL can handle is +0.999999 times 10^{619}, and the smallest is -0.999999 times 10^{619}.

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   (wait for FOCAL to type *)
*TYPE %, A
=+0.306051E+615*
```

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
+ 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. 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,

```
*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*
```

Note that this number is expressed in floating-point format.

2.5 MORE ABOUT SYMBOLS

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*
```

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

```
*TYPE %6.5, $
```

The user's symbol table is typed out like this

```
A@(00)=+XXXXXX
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.

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

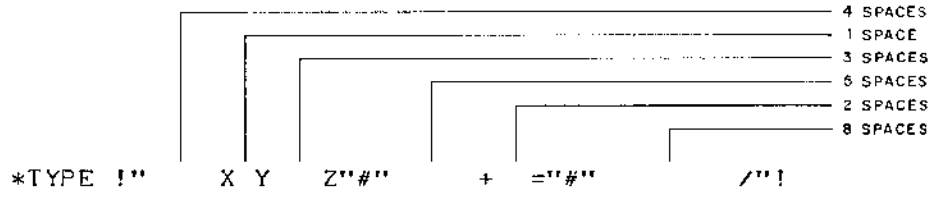
It is useful at times to delete all of the symbolic names which you have defined in the symbol table. This is done by typing a single command: ERASE. Since FOCAL does not clear the user's symbol table area in core memory when it is first loaded, it is good programming practice to type an ERASE command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

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 (see Section 4.5).

2.9 INDIRECT COMMANDS

Up to this point we have discussed commands which are executed immediately by FOCAL. Next, we shall see how indirect commands are written.

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 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 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 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 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
=+6.2645*
```

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

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 B. The group number of an error message indicates the class or general type:

?00 Manual restart from console

?01 Interrupt from keyboard via CTRL/C

?02 Excessive number or illegal mathematical operation.

?03 Miscellaneous

?04 Format errors

?05 Function or command 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 invaluable in program debugging. 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.

The trace feature may also be used to follow program control and to create special formats (see Section 3.15).

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., 1968

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

The left arrow (←) erases everything which appears to its left on the same line.

```
*1.3 TYPE A, B, C←
*WRITE
C-FOCAL., 1968

Ø1.1Ø TYPE X-Y
Ø1.2Ø 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.

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.

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-(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.

```
*ERASE
*9.19 TYPE %4.01, A1*2, E+2↑5, 2.51*81.1
*DO 9.19
=+   0.0=+  32.0=+ 204*
```

The output format may be included in the TYPE statement as shown in the example above and as explained in Section 2.6.

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 ASR33 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,

```
*1.02 TYPE "X SQUARED =
*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.

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 (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 2.0          (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 substituted for the identifier when the identifier is encountered in the program.

```
*ERASE ALL
*3.4 SET A=2.55; SET B=8.05
*3.5 TYPE %, A+B
*GO
=+0.106000E+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

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 2.0                    (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.

```
*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., 1968

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.

```
*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 "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 returns control to the statement following the DO command.

If Do is written without an argument, FOCAL execute 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 IF command is executed within a DO subroutine, two actions are possible:

1. 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.

2. 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 %2.01, 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=-1.0   GIJK=+0.0   GJK=+1.0   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 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.

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 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 right most 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
*GC
=+   9.00
C@(00)=+   5.00
D@(00)=+   4.00
A@(00)=+   0.00
B@(00)=+   0.00
*
```

3.15 USING THE TRACE FEATURE

As stated in Section 2.10, 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 %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*
```

Also GO? Will trace the entire program.

3.16 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 always 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 with their limitations in the pertinent document.

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)
=- 5*
```

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

```
*TYPE %, FRAN( )
=-Ø.250000E+ØØ*

*TYPE FRAN( )
=-Ø.623535E+ØØ*
```

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

```
*TYPE FEXP(6.66953E-1)
=+Ø.194829E+Ø1*

*TYPE FEXP(.666953)
=+Ø.194829E+Ø1*

*TYPE FEXP(1.23456)
=+Ø.343687E+Ø1*

*TYPE FEXP(-1.)
=+Ø.367879E+ØØ*
```

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

```
*TYPE %, FSIN(3.14159)
=+Ø.238419E-Ø5*

*TYPE FSIN(1.4ØØ)
=+Ø.985450E+ØØ*
```

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*
```

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

```
*TYPE FCOS(2*3.141592)
=+0.100000E+01*

*TYPE FCOS(.50000)
=+0.877582E+00*

*TYPE FCOS(45*3.141592/180)
=+0.707107E+00*
```

i. 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*
```

j. 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*
```

CHAPTER 4
EXAMPLES OF FOCAL PROGRAMS

4.1 TABLE GENERATION USING LIBRARY FUNCTIONS

The ability to evaluate simple arithmetic expressions and to generate values with the aid of library 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 and because of the length of several of the commands. 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(I) and FEXP(I). Line 01.20 (optional) returns control to the user.

Several techniques can be noted in line 02.05 of 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 allows accuracy of six significant digits, which makes possible the use of very small loop increments (in this example, .00001); this should eliminate the need to interpolate between table values of trigonometry functions in most cases. With modifications, FOCAL can give results having an accuracy of up to ten significant digits.

```
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   =+0.977508E-05   =+0.271831E+01

=+1.000020   =+0.841481   =+0.540285   =+0.195501E-04   =+0.271834E+01

=+1.000030   =+0.841487   =+0.540278   =+0.293250E-04   =+0.271836E+01

=+1.000040   =+0.841492   =+0.540269   =+0.390998E-04   =+0.271839E+01

=+1.000050   =+0.841497   =+0.540261   =+0.488744E-04   =+0.271841E+01

=+1.000060   =+0.841502   =+0.540252   =+0.586491E-04   =+0.271844E+01

=+1.000070   =+0.841508   =+0.540244   =+0.684236E-04   =+0.271847E+01

=+1.000080   =+0.841513   =+0.540236   =+0.781980E-04   =+0.271849E+01

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

=+1.000100   =+0.841524   =+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.

```
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:

$$T^{0}C = 5/9(T^{0}F - 32)$$

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 parameters 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.

```
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:
     =-  40.0000FAHR. DEG.....=-  40.0000 CENTIGRADE DEG.
     =-  20.0000FAHR. DEG.....=-  28.8889 CENTIGRADE DEG.
     =+   0.0000FAHR. DEG.....=-  17.7778 CENTIGRADE DEG.
     =+  20.0000FAHR. DEG.....=-   6.6667 CENTIGRADE DEG.
     =+  40.0000FAHR. DEG.....=+   4.4444 CENTIGRADE DEG.
     =+  60.0000FAHR. DEG.....=+  15.5556 CENTIGRADE DEG.
     =+  80.0000FAHR. 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.

```
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).

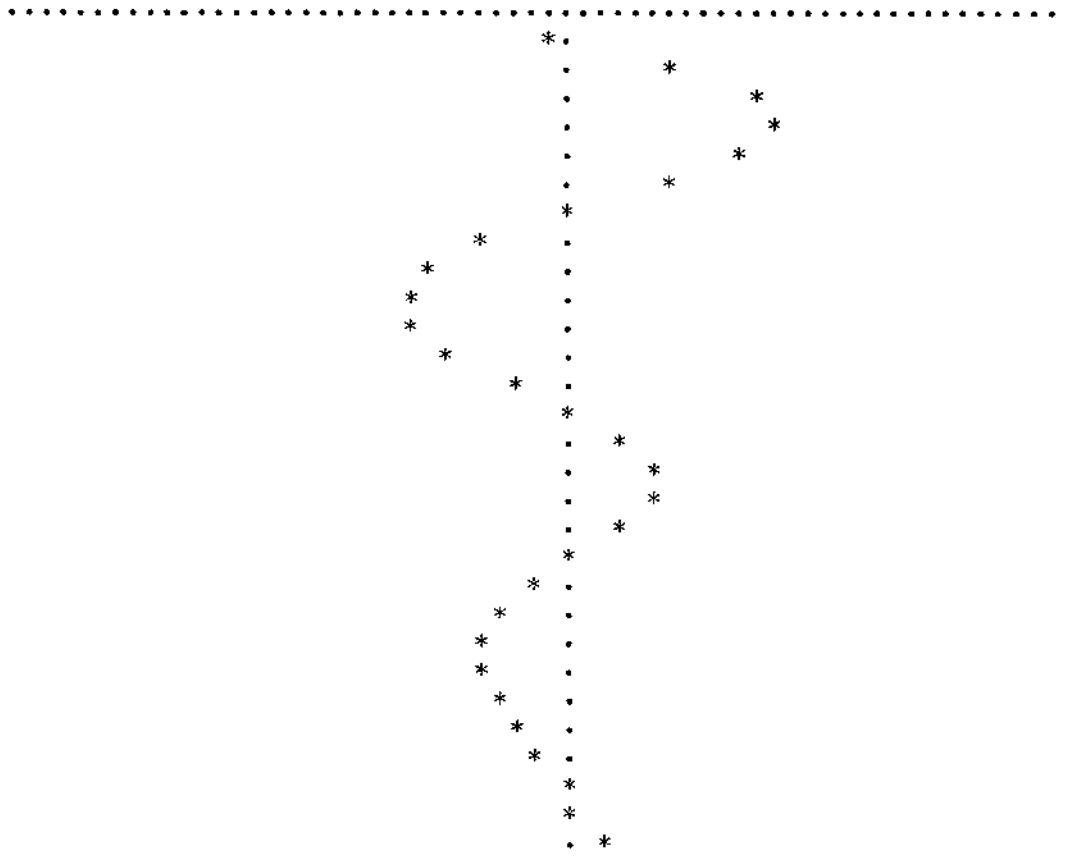
It is often useful to superimpose one function plot upon another. This can be accomplished in FOCAL by replacing the exclamation point (representing a carriage return, line feed) with the number sign (representing a carrige return only) in certain TYPE commands. A very large number of functions using 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.

```
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
```



4.6 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 (remember 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 comments 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.

```
*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;I (1000-A) 3.1
*01.22 I (A-1)3.4,1.26,1.26
*01.26 IF (A-FITR(A))3.5,1.3,3.5
*01.30 ASK M;DO 2;SET D=C;DO 2;T " ";SET 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)1.4,3.2,1.4
*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 SET C=FITR(10*FABS(FRAN()));IF (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,!!;G 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
**
**C-ONCE YOU PLACE A BET, FOLLOW THE OTHER COLONS WITH A
**C   CARRIAGE RETURN TO INDICATE THE COMMAND: "ROLL THE DICE".
**
*GO


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

YOUR BET IS:.50 MINIMUM BET IS $1

YOUR BET IS:15 :7
  =+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:3 :
  =+6  =+5
YOU WIN. YOUR WINNINGS ARE      =+     13

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.

In one dimension:

ARRAY( )

A
B
C
D
E

Element D could be represented as ARRAY(3); any element in this array can be represented by a subscript in the range 0 through 4. The first element in an array always has a subscript of 0.

In two dimensions:

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

This must be reduced to the form A(G). Since subscripts are linear, 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

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

IMAX = 5;
JMAX = 3.

To generate the number (G) in any box, using the corresponding values of I and J, the formula

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

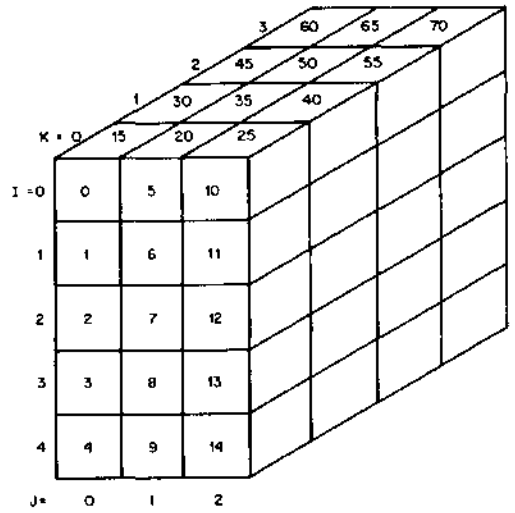
can be used. Each element in a 2-dimensional array represents an area. The example for solving simultaneous equations, above, uses this algorithm for subscripts, merely by replacing I, IMAX, and J with J, L, and K, respectively, so as to form the equation

A(J+L*K).

In three dimensions:

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

Three dimensions can be illustrated as a cube.



This cube has dimensions of five rows, three columns, and four planes; thus, IMAX = 5, JMAX = 3, and KMAX = 4. 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. 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)

In four dimensions:

$$\text{ARRAY(row, column, plane, cube)} = A(I,J,K,L) = A(G)$$

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 theoretically be extended indefinitely to n-dimensions.

```
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

*
```

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 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.
GOTO	G	G 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.

MODIFY	M	MODIFY 1.15	Enables editing of any character on line 1.15 (see below).
QUIT	Q	QUIT	Returns control to the user.

Command	Abbreviation	Example of Form	Explanation
RETURN	R	RETURN	Terminates DO subroutines, returning to 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 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.

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.

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.

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 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.

Scope Functions

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

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
RETURN	Terminator for lines

*These are known as extended functions.

APPENDIX B

ERROR DIAGNOSTICS

Code	Meaning
*?00.00	Manual start from console
*?01.00	Interrupt from keyboard via CTRL/C
*?02.07	Bad line number format
*?02.24	Keyboard input buffer overflow
*?02.28	Group number or literal too large
*?02.29	Illegal command used
*?02.44	Line number too large
*?02.46	Imaginary square roots, or nonexistent line referenced by DO
*?02.61	Nonexistent group referenced by DO
*?02.67	Bad argument for MODIFY
*?02.80	Division by zero
*?02.87	Command input buffer exceeded
*?02.;0	Illegal step number
*?02.;3	Number too large to be made an integer
*?02.;7	Illegal or misspelled function name
*?03.10	Bad argument for ERASE
*?03.42	Log of zero requested
*?03.50	Improper step number
*?03.79	Variable storage exceeded, or exponent not a positive integer
*?04.12	Bad argument in IF command
*?04.13	Missing operator in an expression, or illegal E format on input or literal
*?04.18	Bad argument in FOR, SET, or ASK
*?04.33	Operator missing before parenthesis
*?04.39	Error to left of equal sign
*?04.45	Parentheses do not match
*?04.53	Excess right parenthesis
*?04.61	Illegal character in FOR
*?04.93	Double periods in a line number
*?04.;0	Function not followed immediately by parens
*?04.;2	Multiple periods in a line number
*?04.;9	Double operators in an expression
*?05.11	No argument in IF command
*?05.28	Command not available
*?05.60	Error in FOR command format
*?05.;6	Function not loaded into core

NOTE

The above diagnostics apply only to the version of FOCAL, 1968 issued on tape DEC-08-AJAB-D.

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 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 program area of symbol table area becomes too large, FOCAL types the error message

?03.79

FOCAL occupies core locations 1-3300_8 and 4600_8-7576_8. This leaves approximately 1000_{10} locations for the user's program (indirect program, identifiers, and push-down list). The extended functions occupy locations 4600-5277. If the user decides not to retain the extended functions at load-time, there will be space left for approximately 1800_{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
?03.79                                  (disregard error code)
*TYPE %4, I*5, " LOCATIONS LEFT "
=+ 705 LOCATIONS LEFT *
```

APPENDIX D

CALCULATING TRIGONOMETRIC FUNCTIONS IN FOCAL

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

APPENDIX E
OPERATING HINTS[1]

OVERLOAD RECOVERY

When the program and symbol table areas become too large the error diagnostic ?03.79 will be typed out. The user should then do one of the following.

a. Restart at location 0200.

b. Restart at location 2216, if ?03.79 follows a legitimate command. This erases all variables.

c. As a last resort, restart at 2213. This erases the text.

LOADING PROGRAM TAPES

When loading a long program tape into FOCAL the user can suppress the echo (printing) feature by changing the content of location 2475 to 7000. This will cause only asterisks to be typed as the tape is being read; there will not be a carriage return-line feed at the end of the line.

Entries from the keyboard will not echo unless each entry is preceded by a TYPE command. Output will be typed in the normal manner.

To restore the echo feature, depress the STOP key on the computer console and deposit 4277 into location 2475.

[1] These hints apply only to FOCAL, 1968, issued on tape DEC-08-AJAB-D.

APPENDIX F
LOADERS

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	ASR33 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	5356
7776	0000	0000

b. If the instruction in any location does not agree with the above list, deposit the correct instruction into that location (see User's Handbook, F-85, for details).

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. A detailed description of the BIN Loader is in the User's Handbook, F-85.

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.

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

PRINTED IN U.S.A.