FOCAL USER'S MANUAL

FOCAL Version 3D

FOCAL MINI-MANUAL

This manual is designed to be a brief introduction to FOCAL commands, functions and the language as a whole.

"FOCAL" is a registered trademark of Digital Equipment Corp.

!!!Note!!! Before running FOCAL on your system make sure that the zero page memory location "PDLIST" is set correctly for your memory size. It should contain the address of the highest memory byte minus FF.

Also, check the section of the listing called FOCDEF. This area contains definitions of all of the special definition used by FOCAL. Make sure they match your system configuration.

Loading FOCAL on your KIM system

FOCAL is supplied as an audio cassette containing two blocks. The ID of both blocks is Ø1. One block contains page zero (0000 - 0100), the other block contains the FOCAL program (2000 - 36B0). Both must be loaded for FOCAL to operate correctly. After loading both blocks, enter address 2ØØØ and G (to GO). FOCAL will type "-17@" and an asterisk. You are now in FOCAL command mode.

If you ordered FOCAL on paper tape, load the tape using the "L" command or whatever command loads KIM/TIM format tapes on your particular system.

Part One
Getting Started in Programming

FOCAL is a programming language which allows the human user to communicate with a computer in higher-level mathematical terms without having to be concerned with electronic particulars of the given computer. FOCAL is an interactive language, in that you communicate directly with it through the use of a keyboard-input system, and it reacts immediately to the commands you give it. This allows for quick and easy creation, testing, and execution of programs. This reduces the amount of human effort involved in programming which, in many cases, represents the major expenditure in solving problems with computers.

Once the FOCAL system has been loaded and initialized on the computer that you are using (procedures may vary with computer type), then you are ready to interact with the computer through commands expressed in the "FOCAL" language. Much of the rest of this document will be concerned with the structure and rules of the "FOCAL" language.

Some Fundamental Ideas

The user communicates with the computer through the use of "commands" which are recognized by the "FOCAL" system and cause it to perform certain specific actions. A FOCAL command must always begin with a command "name"--which may be abbreviated. The command "name" is then followed by other parameters upon which the command will operate. Several commands may be placed on one line by separating them with the semicolon character (;). The entire line is ended with the "RETURN" character ("CR" on some keyboards). The general format may be illustrated as:

```
COMMAND1 PARAMETER1;COMMAND2 PARAMETER2;COMMAND3 PARAMETER3  RETURN
```

When the "FOCAL" system requires a line to be input by the user, it outputs a "prompting" character (usually an asterisk "*").

High Speed Calculations Using the "TYPE" Command

You only need to learn one FOCAL command, "TYPE" (abbreviated "T") in order to do calculations such as the following:

$$10^3 \times 3/10 + 21 - 2$$

To do this with FOCAL, you type:

`*T 10↑3 *3/10 + 21-2`
`= 319.0000`
	(And after you press the return key FOCAL computes this result. Every line must end with a return.)

This example shows the arithmetic operations performed by FOCAL. These are done from left to right except that exponentiation (↑) is done first, then multiplication (*), then division (/), then addition (+) or subtraction (-).

This means that

`*T 6/6*2`	is .5000 and not 2.0000 because multiplication is done before division in FOCAL.

Enclosures

To make sure that the computer performs these operations in the order you want, you can place them inside parenthesis marks. When the computer sees an expression enclosed in parentheses, it does that first. If the statement includes parentheses within parentheses (nesting), it computes the innermost first.

```
7+(6/3) - (5+2) *3
```

In this example the computer first computes the values of the expressions enclosed in parentheses (6/3) is 2, and (5*2) is 25. Then 7+2 - 25*3 = -66.ØØØØ.

Another Command: SET

This useful command tells FOCAL, "Store this symbol and its numerical value. When I use this symbol in an expression, insert the numerical value."

```
*SET PI=3.14159; SET E=2.71828; SET R=9.12739
```

Symbols may consist of one or two (depends on the implementation you are using) alphanumeric (letters and numbers) characters. The first character must be a letter, but cannot be the letter "F" which has special meaning, as we will see later.

Just for practice, let's use FOCAL to calculate the volume of a sphere which has a radius of 9.12739. (We're going to use two of the symbols we have just defined in the SET commands above.)

The formula is V = (4/3)*PI*R^3 whcih we can type like this:

```
*T R↑3*PI*4/3
```

You might be interested in running a timing test to show how long it takes to do such calculations by hand, with a calculator, and with FOCAL.

The Talking Typewriter

To make the output of your program absolutely clear to other people, it is sometimes useful to give FOCAL certain messages or column headings. We call these "character strings." These messages are enclosed in quotation marks.

```
*SET E=2.71828
*SET PI=3.14159
*TYPE "PI TIMES E=",PI*E
```

and FOCAL types out:

```
PI TIMES E=  8.5397*
```

You are not allowed to use the carriage return, line feed or leader-trailer characters in these character strings, but you can tell FOCAL to do a carriage return/line feed by inserting an exclamation mark (!). You can get just a carriage return by inserting a number sign (#).

```
*TYPE  "JOHN"!,  "BILL"!,  "FRED"!,  "SAM"!,  "RICHARD"!
JOHN
BILL
FRED
SAM
RICHARD
*
```

Spaces can be used in character strings as needed.

Keeping Track of the Decimal Point

FOCAL results are accurate to 9 significant digits (depends again on the implementation you are using).

As we have shown in the examples so far, FOCAL assumes at the start that you want to see your results with 4 digits (or spaces) to the left of the decimal point and 4 digits to the right of the decimal point. This is called fixed-point notation.

You can change the output format within a type statement by typing "%X,Y" where X is the total number of digits to be output, and Y is the number of digits to the right of the decimal point. Both X and Y are positive integers equal to or less than 31. If Y is a single digit, it must be preceded by Ø. For example, %6.Ø2 indicates six digits to the left and two to the right of the decimal point.

Correcting Mistakes

If you should strike the wrong key, you can delete it by striking the rubout key. Each time you strike rubout, another previously typed character will be deleted. When you strike rubout, FOCAL echoes back a backslash "\" (may vary with implementations) to inform you that it has deleted the character.

You may erase everything back to the left margin by striking the "backarrow" key.

Summary of Part One

In this first part you have learned how one FOCAL command type is used to evaluate expressions, to type out character strings enclosed in quotes, and to use symbols (defined in SET commands) in expressions.

In the second part you will learn the other commands and the use of line numbers to write a sequence of FOCAL statements. As you learn these techniques you will be advancing rapidly in the art of computer programming.

Part Two

Sequential Commands

Indirect Commands

Up to this point, all of our commands have been executed immediately upon hitting the "return" key. If a line is prefixed by a line number, that line is not executed immediately; instead, it is stored in the computer's memory for later execution, usually as part of a sequence of commands.

Line numbers must be in the range from 1.01 to 99.99.

The numbers 1.00, 2.00, etc., are illegal line numbers; they are used to identify 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.

```
*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.

GO Command

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 in the example above, FOCAL will carry out the command at line 1.1, and then the others, sequentially.

```
*GO
 3*
```

GOTO Command

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 specific line, FOCAL continues to the next higher line. The GOTO causes a program branch; we have jumped from one sequence of lines to another. Sometimes we merely jump back and repeat a sequence of commands. This technique of repeating sequences is called iteration and it is often used by experienced computer programmers.

```
GOTO 1.3
 2*           (Since line 1.1 has not been executed, FOCAL gives
              the symbol A the value zero.)
```

For a line number in the GOTO command, FOCAL will allow an arithmetic expression which begins with a variable (e.g., X*2), for example, a statement in a program may look like the following:

```
*1.7 GOTO X*2
```

Note that the command GOTO 2*X, is illegal since the arithmetic expression begins with a number rather than a variable.

Other commands in which line number computation is provided for are the "DO," IF," "MODIFY," and "ON" commands.

DO Command

The DO command is used to transfer control to a specific step, or group of steps, and then return automatically to the command following the DO command. The step, or group of steps, may be given as an arithmetic expression beginning with a variable (as in the GOTO command).

```
*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*
```

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

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

The following example causes a programming loop which could be terminated by inserting line 1.5 QUIT, see below.

```
*1.1 SET A=1
*1.1 TYPE A
*1.3 DO 2.Ø
*1.4 TYPE "FINISHED"

*2.1 SET A=A-1
*2.2 TYPE A
```

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, explained below.

RETURN Command

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.

IF Command

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

The program below transfers control to line number 2.1Ø, 2.3Ø, or 2.5Ø, 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.

```
*2.2Ø IF (X) 1.8; TYPE "Q"
*
```

In the above example, when line 2.2Ø 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
*3.2Ø 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 3.2Ø and the value of B is typed out.

If a GOTO or an 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 command 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 "C"
*2.2 IF (X)2.5, 2.6, 2.7
*2.5 TYPE "H"
*2.6 TYPE "I"
*2.7 TYPE "J"
*2.8 TYPE "K"
*2.9 TYPE %2.Ø1, X; TYPE " "; SET X=X+1
*
*3.1 TYPE "B"; GOTO 5.1; TYPE "F"
*
*5.1 TYPE "C"
*5.2 TYPE "#"
*5.3 TYPE "L"
*GO
```

(FOCAL types the answer)

```
ABCDGHIJK-1.Ø GIJK 1.Ø BCEL*
```

<u>ON Command</u>

The ON command is a three-branch conditional "DO." It functions similarly to the IF command (actually uses some of IF's coding) except that the line number (or group) is executed as in a "DO" command rather than as in a "GOTO" command.

Example:

```
ON (X)2.1,5,3.4;D 6; CONTINUE
```

In this case,

If X < Ø, line 2.1 is executed and then group 6 is executed; control continues onto the next line.

If X = Ø, group 5 is executed, then group 6.

If X > Ø, line 3.4 is executed, then group 6.

In the ON command, the line number may be an arithmetic expression, but the arithmetic expression must begin with a variable (as in the GOTO command).

More about Symbols

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

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

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

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

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

The user's symbol table is typed out like this:

```
A (ØØ)=  1111.11ØØØ
B (ØØ)=.3Ø6Ø51
C (ØØ)=    3Ø1.ØØØØØ
*
```

Subscripted Variables

FOCAL allows variables to be further identified by subscripts (max range may depend on implementation), positive and negative, which are enclosed in parentheses immediately following the variable name. A subscript may also be an expression:

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

The ability of FOCAL to compute subscripts is especially useful in generating arrays for complex programming problems.

An important difference between the "FOCAL" language and most other languages is that only the elements in an array that are actually used by the program are set aside in the computer's memory. Thus no definition of array size is needed.

QUIT Command

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.

COMMENT Command

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.

FOR Command

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

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

The identifier A is initialized to the value B, then the commands following the semicolon are executed. When the commands have 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 commands after the semicolon are executed again. This process is repeated until A is greater than D, at which time 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 the 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=1ØØ
*1.2 FOR B=1,1,5; TYPE %5.Ø2, "N IS="B+A,!
*GO
N IS=1Ø1,ØØ
N IS=1Ø2.ØØ
N IS=1Ø3.ØØ
N IS=1Ø4.ØØ
N IS=1Ø5.ØØ
*
```

Example 2:

```
*1.1 FOR X=1,1,5;DO 2.Ø
*1.2 GOTO 3.1
*
*2.1 TYPE !"      "%3,"X"X
*2.2 SET A=X+1ØØ.ØØØ
*2.3 TYPE !"     "%5.Ø2, "A"A

*
*3.1 QUIT
*GO

        X  1
        A  1Ø1.ØØ
        X  2
        A  1Ø2.ØØ
        X  3
        A  1Ø3.ØØ
        X  4
        A  1Ø4.ØØ
        X  5
        A  1Ø5.ØØ*
```

ASK Command

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

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

When line 11.99 is encountered by FOCAL the user then types a value in any format for the first identifier, followed by a terminator. (Terminators are space, comma, alt mode and return keys.) FOCAL then asks for another value (no visible indication on the paper), 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. The user must end the last number in an ASK sequence with the return key.

```
*11.99 ASK X,Y,Z
*DO 11.99
5 4 3  (RETURN KEY WAS TYPED HERE)
*
```

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, just as in the TYPE command.

```
*1.10 ASK "HOW MANY APPLES DO YOU HAVE?"  APPLES
*DO  1.10
HOW MANY APPLES DO YOU HAVE? 25
*
```

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

Error Correction Commands and Procedures

One of the advantages in the FOCAL language is the ability to find errors and to make the appropriate corrections with a minimum of effort. Several commands and procedures are available to the programmer for quick and easy debugging.

WRITE Command

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

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

```
*WRITE 2.0                (FOCAL types all group 2 lines)
*WRITE 2.1                (FOCAL types line 2.1)
*WRITE                    (FOCAL types all numbered lines)
*
```

ERASE Command

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 earase the user's symbols and control remains in the same mode as the one in which the command was issued (either the stored mode or the immediate mode). In the stored mode, the ERASE command must be the last command in the line. Note that the command, ERASE with an argument, causes control to go to the immediate mode.

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

Corrections

If the user types the wrong character, or several wrong characters, he can use the rubout key as we explained in Part One, 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                (Erases program text and variables)
*1.1 P\TYPE X-Y
*1.2 SET $=13\\\\X=13
*WRITE

Ø1.Ø TYPE X-Y
Ø1.2Ø SET X=13
*
```

A line can be corrected by retyping the 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
*
```

MODIFY Command

Frequently, only a few characters in a particular line require changing. To facilitate this job and to eliminate the need to retype 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 an altmode character and then a search character. FOCAL then proceeds to the first occurrence of that character. Characters can now be inserted or deleted from that point. If the user wishes to search for another character in another position in the line, he types another altmode and the search character. If the user types a line feed, the remainder of the line will be accepted as is. If the user types a carriage return, all of the text from the current position to the end of the line will be deleted.

The ERASE ALL and MODIFY commands are generally used only in immediate mode because they return to command mode upon completion.

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.Ø1 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.Ø1, 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 alt mode 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.

Error Detection in Indirect Statements

When an error occurs in executing an indirect statement, the error message is typed out when the statement is encountered during program execution. In addition to the error code, FOCAL types the line number containing the error, as shown in the following example.

```
*1.1Ø SET A=2; TYPE "A", A,!
*1.2Ø SET B=4; TYPE "B", B,!
*1.3Ø GOTO 1.Ø1
*1.4Ø TYPE "A+B", A+B
*GO
A   2.ØØØØ
```

```
B   4.0000
?-3 @ 1.30

*
```

FOCAL executes lines 1.1 and 1.2 and then recognizes that line 1.3 is an illegal command. Therefore it issued the error message to show you that an illegal command was used. An error code table is available at the beginning of each source listing.

To pinpoint an error in line 3.3, for example, type "DO 3.3 ?" and the program will be traced until the error is found.

Using the Trace Feature

The trace feature is useful in checking a program's operation. 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 three lines are printed.

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

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

```
* ERASE ALL
* 1.1 ?SET A=0; TYPE %3,A!
* 1.2 for B=1,1,4; TYPE B+A!
*GO
SET A=0 ; TYPE %3,A!
= 1 TYPE B+A!
= 2 TYPE B+A!
= 3 TYPE B+A!
= 4*
```

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

FOCAL Functions

The functions are provided to improve and simplify arithmetic capabilities and to give potential for expansion to additional input/output devices. A standard function call consists of four (or fewer) letters beginning with the letter F and followed by a parenthetical expression:

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

There are basic mathematical functions provided as part of the FOCAL system, and extended functions which allow the user to do special operations (like reading data from external storage devices). Also, FOCAL provides the mechanism for user written functions, programmed in either "FOCAL" or machine language. Some FOCAL functions are specific to the machine you are using, or the implementation of the FOCAL system on that machine.

FOCAL functions are executed using the SET or the TYPE command

```
*T FINT(Y)
*S X=FODV(1)
```

All FOCAL functions return a value, and the value can either be placed in a variable or typed out. If the function is a mathematic operation the value returned is the result of the operation. With some non-mathematic function, such as (FODV), the value returned is superfluous and can be ignored.

Absolute Value

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*
```

Integer

Integer part function (FINT) outputs the integer part of a number.

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

Integer Rounding

The integer rounding function (FINR), keeps the integer part and round. [illegible] 0.5 or larger to the next higher integer.

```
*TYPE FINR(5.5)
5.ØØØØ*
*TYPE FINR(5.6)
6.ØØØØ*
```

Random Number Generator

A random number generator, FRAN ( ), generates repeatable random numbers between Ø and Ø.999999.

```
*TYPE %, FRAN ( )
=Ø.6Ø7295
*TYPE FRAN ( )
*=Ø737615
```

The random number generator must be initialized before it is used. Generally, this is done at the beginning of a program. It can be initialized using the SET command.

```
S X=FRAN(-1)
S X=FRAN(1)
```

If FRAN is [illegible] a positive number, the random sequence is repeatable. If it is set to a negative value, the sequence is entirely random.

```
S X=FRAN(Ø) OR S X=FRAN( )
```

FRAN with a zero subscript returns with X set to a random value.

Input and Output Device Functions

Input device function (FIDV) and output device function (FODV) allow FOCAL to send and receive information through more than one device. For example,

```
*S X=FIDV(1)
```

will set FOCAL's input device to device number one. Generally, device number Ø is the console device. Other devices such as printers or cassette recorders can be added by placing the addresses of the I/O drivers in the I/O dispatch table at the end of the listing. (Note: The symbols IDEVM and ODEVM set the maximum number of input and output devices allowed. They may be altered by patching their value in the routines labeled CHKODV and CHKIDV.)

Character Input and Output Functions

The functions FOUT and FCHR allow decimal numbers to be converted to ASCII equivalent characters and vice versa. Here are some examples:

```
*S X=FOUT(65)
A*

*S X=FCHR()
A*
*T X
65.0000*
```

In the first example, 65 is converted to the character "A" and printed. Any printable character, control character, or binary value can be printed or sent to the output device. In the second example, X is set to the decimal value of the character typed or sent to FOCAL. Using these two functions the numeric values of characters may be tested and operated upon mathematically

Echo Control Function

The echo control function (FECH) allows echo to the console device to be suppressed.

```
*S X=FECH(1)
*S X=FECH(0)
```

Setting the control to one suppresses the echo, setting it to zero restores echo.

Memory Examine and Deposit Function

This function (FMEM) allows machine level memory locations to be examined or deposited by FOCAL. If a memory location is examined, the function carries two arguments: the decimal equivalent of the high and the low byte of the address. If a value is to be deposited a third argument carries the decimal value to be loaded into memory.

```
*T FMEM(20,01)
25.0000*
*S X=FMEM(20,01,00)
*T FMEM(20,01)
0.00000*
```

FOCAL Subroutine Function

The subroutine function (FSBR) allows the user to create functions within a program. FSBR calls a line or a group of

FOCAL code and returns a value. The first argument in the function is the line or group number of the code, and the second argument is the value carried to the subroutine. The variable "&" replaces the second argument and carries its value.

```
1.1 ASK X;TYPE FSBR(99,X);Q

99.1 S &1=&;S &=2;S &3=.ØØØØØ1
99.2 S &2=&1/&;1 (FABS(&2-&)=&*&3)99.3;S &=(&+&2)/2;G 99.2
99.3 R
```

Initialize Input-Output Device Function

These functions (FINI, FINO) allow input and output devices requiring specialized initialization to be set up through FOCAL. The address of the initialization routines are placed in the dispatch tables and the function is called with the device number as an argument.

```
*S X=FINI(2)
```

Cursor Address and TV Memory Functions

These functions (FCUR, FTVM) allow direct cursor addressing and direct TV memory examination. Currently, drivers exist only for the Digital Group system TV board.

Part Three

FOCAL Version 3D

FOCAL Version 3D has a number of new features and optimizations. These features include increased speed, string functions, more powerful DO and SET commands, explicit error messages, improved random number generator and priority interrupt control.

SET

SET now has the ability to evaluate more complex expressions. For example, it is now possible to execute the following statements:

```
SET X=Y=Z=9

SET X=1, Y=2

SET X=(Y=A+B) + 4
```

This new feature enables the user to generate much more compact and efficient statements. The following examples illustrate the same concepts written in both old and new FOCAL:

```
Old:
S A=4;S B=D/3;S G=A+B

New:
S G=(A=4)+(B=D/3)
```

The ability to evaluate complex expressions is also available in other FOCAL commands. For example:

```
IF ((Y=D/3)-5)2.1,2.2,2.3

TYPE I=X+U

GOTO Y=G+.1

DO Z=(Y=R+.1)*X=A/B

For X=1, Y=4/R;T X
```

The SET command can now be used to execute a function without setting a variable. For example, both of these expressions are correct:

```
SET X=FECH(1)

SET FECH(1)
```

Strings

String functions allow up to 25Ø characters to be stored in a single variable. Generally, these are ASCII characters, but they can also be any 8-bit number (Ø - 255).

The format of string variable names is the same as regular variables except that they are followed by a dollar sign:

```
A$            B5$            C4$(23)
```

Each position in a string is numbered starting with zero. For example, "B3$(23)" designates the twenty-third character in string B3. Each character in a string can be treated as a numeric value, and as such can be operated upon by all normal FOCAL commands. In addition, a number of new functions have been created to deal with strings.

FISL

This function sets aside a certain amount of space for a string. For example, "S FISL(1ØØ,A$)" sets aside 100 characters for "A" string. Several strings can be initialized at once by including several arguments with the function.

```
S FISL(1ØØ,A$,25Ø,B7$)
```

When a string is initialized it fills with spaces. If a string is called by a routine without having been initialized with the FISL function, the default string length sets the string to 72 characters. Once a string has been initialized, its length cannot be changed.

FSTI

This function allows characters read from the input device to be inserted into a string. The function requires three arguments: the number of characters to be loaded, the name of the string and string position, and a terminating character.

```
S FSTI(1ØØ,A$(4),13)
```

This statement instructs FOCAL to read characters into "A" string starting at character position number 4. The function will read characters into the string until either 100 characters are loaded or the character represented by the decimal ASCII value 13 is encountered. In this case, 13 is the decimal value for a carriage return. If the keyboard is the input device, the user could type characters into "A" string until either 100 characters are loaded or a carriage return is typed. Since FOCAL can deal with strings either as characters or as numeric values, a simple method has been provided to convert characters into their numeric values. Preceding a character with a single quote (') returns the decimal value of that character.

```
*T 'A
*65.0000
```

```
*S FSTI(100,A$,'@)
```

```
*F X=0,25;S Y='A+X
```

FSTO

This function allows part or all of a string to be sent to the output device. It has the same format as the FSTI function. For example, "S FSTO(100,A$(4),13)" will send characters from "A" string starting with position 4. It will continue to send characters until either the 100th character is sent or a carriage return is encountered in the string.

FSLK

Allows all or part of one string to be compared with all or part of another string. The function requires four arguments to define the start and end of the first string and the start and end of the second string.

```
*S Z=FSLK(A$(1),A$(5),B$(0),B$(50))
```

In this example, FOCAL will try to find the characters of the first string in the second string. If a match is found, Z will be set to the position number where the match began. If no match is found, Z is set to -1.

One of the most powerful string functions in FOCAL allows strings to be set as input and output devices. This enables text or data generated during program execution to be stored directly into a string. It also enables data or text to be read directly into an executing program. We'll go through an example step by step:

*S FODV(A$)	This sets "A" string as the output device. Now all information that would normally go to the CRT or teletype is loaded into "A" string.
*T "HELLO"	Executing a TYPE command loads the text into the string. This is the usual technique used to preload a string.
*S FSTO(,B$,)	Executing this function copies "B" string into "A" string.

Similar kinds of operations can be performed by setting a string as the input device. For example:

```
*1.1 S FODV(A$);T "3,4";R O
*1.2 S FIDV(A$);A X,Y;R I
```

This routine loads the string with the text "3.4" and then loads them into variables X and Y.

Notice that the input and output devices must be restored to the console once the operation is complete. This is accomplished by using the "R O" and "R I" commands. The former restores the output device and the latter restores the input device.

One powerful and unusual feature available in FOCAL is the ability to execute FOCAL code contained within a string. If the text contained within a string is a series of FOCAL commands, the text can be executed using the "DO" command.

FOR Loops

FOCAL now has the ability to exit from a FOR loop before the loop is completed. This is accomplished using the "RETURN" command. For example:

```
*1.1 F X=1,10;I (X-6),,1.3
*1.2 G 6.1
*1.3 T "DONE";R
```

The FOR loop will execute until the "IF" statement sees that X has exceeded 6. The program then goes to line 1.3, where it encounters the "RETURN." The RETURN forces the FOR loop to finish and sends the program to line 1.2.

FOR loops can now be stepped in both a positive and negative direction. For example:

```
*1.1 F X=10,-1,1;T X,!

10
 9
 8
 7
 6
 5
 4
 3
 2
 1
*
```

Exponents

FOCAL now understands negative exponents. For example:

```
*T 34.67↑(-12)
```

Priority Interrupts

FOCAL now has the ability to deal with hardware interrupts through the FPIC function. The interrupting external device can be given a priority from 1 to 7, with 7 having the highest priority. The priority levels correspond to bits in a special byte; the most significant bit corresponds with the lowest level of priority. The user must provide machine language routines to set the bits that correspond with a specific external device.

FOCAL deals with interrupts by executing a "DO" of a FOCAL line number that is assigned to the interrupting device. FOCAL will ignore the interrupt bits unless the specific bits are enabled with the FPIC function. Example:

```
S FPIC(1,99,3,22.1,7,3Ø)
```

The arguments are arranged in pairs, with the first argument being the device priority bit and the second argument being the line to be executed if that device interrupts.

FOCAL looks for interrupts before each command FETCH, so that in some routines the interrupts would not get serviced very quickly. Whenever FOCAL returns from program execution to command mode, all interrupts are disabled and must be re-enabled during program execution.

Accuracy

FOCAL now calculates floating point numbers to about nine digits of accuracy. This means that FOCAL can deal with numbers as large as 999999999.

Saving FOCAL Programs

There are several methods of saving programs generated in FOCAL for later use. Since they depend heavily on the type of mass storage device available the exact routines have been left to the user.

(1) Complete Memory Image

The simplest method of saving a program is to copy the complete core image of FOCAL and the text area onto the mass storage device. If the mass storage device is fast, this is probably the easiest way to store a program. If the mass storage device is slow, this method is tedious and inefficient.

A slightly more efficient method involves saving a memory image of the text and the text pointers only. Before the image is saved the variable list should be deleted using the "ERASE" command. The text pointers are on zero page. All of the memory image between the labels PC: and VAREND: should be saved. The text area follows the FOCAL interpreter. The pointer on zero page labeled VARBEG: indicates the end of the current FOCAL program. The text is stored between the label PBEG: (at the end of the interpreter) and the address pointed to by the label VARBEG:

One disadvantage of these two methods is that a memory image cannot be loaded into a different version of FOCAL, or even into the same version of FOCAL assembled at different addresses.

(2) <u>Listing Save</u>

The most obvious method of saving a program is to have FOCAL send a listing to the mass storage device. FOCAL can then read the text back in as though it were being typed into the keyboard. This method has the advantage of being able to transfer the program between assemblies and versions of FOCAL. The problem with this method is that after FOCAL sees the carriage return at the end of the text line it must do quite a bit of processing to store the line in the text area. If one line of text immediately follows another, characters will be lost while FOCAL stores the text.

One solution involves having the mass storage device under start stop control. This way FOCAL can read text as it is needed. Another solution is to insert delays or fill characters after every carriage return. The delay should be between 25Ø and 5ØØ msec. If this method is used, the echo flag must be turned off while the text is loaded.

The dilemma for us is to find a program save routine that will work on all possible devices. For example, delays work fine on cassette mass storage, but are worthless on paper tape. The lowest common denominator is the memory image method, since every system can save programs this way.

FOCAL Memory Map

```
ØØØØ    ==========================================
        I                                        I
        I               ZERO PAGE                I
        I                                        I
        I----------------------------------------I
        I                                        I
        I             TEXT POINTERS              I
        I                                        I
        I----------------------------------------I
        I                                        I
        I               ZERO PAGE                I
        I                                        I
        I========================================I
Ø1ØØ    I                                        I
        I              LINE BUFFER               I
        I                                        I
        I                                        I
        I              STACK PAGE                I
        I                                        I
        I                                        I
        I========================================I  2ØØØ
BEGIN   I                                        I
MAIN    I                                        I
MEM     I                                        I
        I          MAIN BODY OF FOCAL            I
        I                                        I
        I                                        I
        I                                        I
        I========================================I
        I                                        I
        I               TEXT AREA                I
        I                                        I
        IVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVI
        I                                        I
        I           AVAILABLE MEMORY             I
        I                                        I
        I↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑↑I
        I                                        I
        I            SOFTWARE STACK              I
        I                                        I
        I========================================I

                      END OF MEMORY
```

Summary of FOCAL Commands

Command	Example	Explanation
ASK	ASK X;	Input value of variable
	ASK "TEXT",X;	Type text, then input X
COMMENT	C TEST PROGRAM	Ignore rest of line
CONTINUE	1.1 C	Null target of branch
DO	DO 23.73;	Call line 23.73 as a subroutine
	DO 1Ø;	Call all of group 1Ø as a subroutine
ERASE	E;	Erase variables
	E 12.1	Delete program line 12.1Ø
	E A	Erase both text and variables
FOR	FOR I=1,1Ø;<TEXT>	In Fortran, DO 99 I=1,1Ø <TEXT> 99 CONTINUE
	FOR J=1,3,25;<TEXT>	In Algol, for J:= 1 STEP 3 UNTIL 25
GO	GO	Start program
	GO 1.23	Unconditional branch to 1.23
IF	IF(X)1.1,1.2,1.3	Conditional branch (as in Fortran IF)
	IF(Y),,2.3;<TEXT>	If Y > Ø go to 2.3, else DO <TEXT>
	I(Z-X)3.4;<TEXT>	If Z < X go to 3.4, else DO <TEXT>
MODIFY	MOD 1.53	Edit line 1.53
	M 1.34=5.43	Edit line 5.43 and put the result in line 1.34. Leave 5.43 alone.
ON	ON(X)1,2,3;	Conditional subroutine call (-,Ø,+)
	O(Y),2.3;<TEXT>	If Y = Ø do line 2.3; in any case DO 2.3
QUIT	Q	Stop program execution. Go to "*" mode
RETURN	R	Return from subroutine

Command	Example	Explanation
SET	SET X=23+3*2;	Assign the value of 23+3*2 to X
TYPE	TYPE X,X*X,X↑3	Type the values of X, X↑2, and X↑3
	T "<TEXT>"!;	Type <TEXT>, then carriage return/line feed
WRITE	WRITE;	List the program
	W 3;	List group 3