028002

ANCR-1184-4

UC-32

RAMOS
FOCAL PROGRAMMING MANUAL

by

W.R. Myers

R.C. Davies

Aerojet Nuclear Company

IDAHO NATIONAL ENGINEERING LABORATORY

Idaho Falls, Idaho — 83401

DATE PUBLISHED—NOVEMBER 1974

PREPARED FOR THE
U. S. ATOMIC ENERGY COMMISSION
IDAHO OPERATIONS OFFICE UNDER CONTRACT AT(10-1)-1375

35

Printed in the United States of America
Available from
National Technical Information Service
U. S. Department of Commerce
5285 Port Royal Road
Springfield, Virginia 22161
Price: Printed Copy $4.00; Microfiche $2.25

ANCR-1184-4

Mathematics and Computers
TID-4500

RAMOS
FOCAL PROGRAMMING MANUAL

W. R. Myers

R. C. Davies

AEROJET NUCLEAR COMPANY

Date Published - November 1974

PREPARED FOR THE U. S. ATOMIC ENERGY COMMISSION

IDAHO OPERATIONS OFFICE

UNDER CONTRACT NO. AT(10-1)-1375

/

ABSTRACT

RAMOS FOCAL is a version of the Digital Equipment Corporation FOCAL (Formulating On-Line Calculations in Algebraic Language) high level interpretive language processor. It has been modified to operate with the RAMOS PDP-15 operating system, and optimized for the PDP-15 instruction set. Several features have been added, including program library operations, program chaining, computed "GOTO", more versatile output formatting, and program selection of input and output devices.

TABLE OF CONTENTS

Page

I. INTRODUCTION TO RAMOS FOCAL*

FOCAL (Formulating On-Line Calculations in Algebraic Language) is an on-line, conversational, high level problem solving language developed by Digital Equipment Corporation for PDP computers. The language consists of short imperative English statements in which mathematical expressions are represented in standard algebraic notation. The FOCAL language processor is interpretive and therefore relatively slow, but it makes the full computational power of the host computer easily available to the user.

There is probably no other computer language that has been modified more than FOCAL, by individual users, to meet the problem-solving requirements peculiar to their own needs. RAMOS FOCAL has evolved in several stages from "standard" DEC PDP-8 FOCAL. The language has been included in the RAMOS system to provide post-processing, formatting, and report generation for data that have been partially processed by fast, complex primary data reduction modules such as 'STRIP' and 'GAUSS'.

FOCAL operates in the RAMOS PDP-15 operating system environment, utilizing monitor input/output facilities and the monitor-resident re-entrant floating point arithmetic facilities. FOCAL operates in any available PDP-15 (4096 word) page; the interpretive language processor occupies approximately 1750 locations, leaving 2340 locations for the FOCAL language program and variables. This space is adequate for a typical program of approximately 150 statement lines. An operating FOCAL program may access any of the RAMOS alphanumeric input/output devices including the paper tape reader and punch, the line printer, and the interprocessor data links. FOCAL also has access to the RAMOS "common" registers through a special function.

Commonly used programs may be kept on paper tape or in an on-line (DECtape #6) library. The on-line program library has a capacity of approximately 70 programs which are accessed through a directory using a three-character program name. The library commands implement the capability of chaining program modules together to perform more complex calculations.

FOCAL may be initiated from any RAMOS command terminal by typing the monitor command 'FOCAL'. When the monitor has loaded and started the program, the system will respond by typing 'FOCAL', followed by an asterisk to indicate that it is ready for user commands or input.

*FOCAL and PDP are trademarks of Digital Equipment Corporation.

II. THE FOCAL LANGUAGE

The FOCAL language consists of simple imperative English-language statements such as: SET A=B; TYPE A; GO 7.32; DO 5, etc. The statements may be either "direct", indicating that the operation is to be performed immediately, or "indirect", indicating that the line should be stored as part of a procedure that will be performed at a later time. The use of direct statements allows FOCAL to be used as a very powerful calculator, and enables on-line editing and modification of indirect programs. The indirect mode is used to write sequential procedures to perform more complex calculations and to provide a means of saving and restoring commonly used procedures. A statement preceded by a line number (of the form GG.SS) is indirect; if the command is entered without the line number, it is direct and is executed immediately.

1. 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	A^B	A↑B
* Multiplication	$A \cdot B$	A*B
/ Division	$A \div B$	A/B
+ Addition	$A+B$	A+B
- Subtraction	$A-B$	A-B

These operations may be combined into expressions. When FOCAL evaluates an expression which may include several arithmetic operations, the order of precedence is: exponentiation, followed by multiplication, division, addition and subtraction. Expressions with the same precedence are evaluated from left to right. Addition and subtraction have the same precedence.

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 $\frac{X}{2Y}$ (multiplication has a higher precedence than division)

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, or angle brackets. Expressions may be nested; FOCAL computes the value of nested expressions by doing the innermost first and working outward.

2. FOCAL SYMBOLS

FOCAL allows variables to be represented by alphanumeric symbols of up to three characters, the first of which must be alphabetic. If a symbol name consisting of more than three characters is used, only the first three are recognized. FOCAL includes several implicitly defined names for function evaluation and numeric manipulation (see Section III-18) which may not be used as variable names.

A	legal
ABC	legal
KA1	legal
1A3	illegal, first character not alphabetic
SAMMY	legal, but equivalent to 'SAM'
COS	illegal, a reserved function name

3. SUBSCRIPTS

FOCAL always allows identifiers, or variable symbols, to be further identified by subscripts (range ± 65535) 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
```

4. SIMPLE COMMANDS

One of the more useful commands in the language is the 'TYPE' command which directs FOCAL to evaluate and print the results of the following expression:

```
If the user enters:    *TYPE 6.4318+8.1346

FOCAL answers:          14.5664
```

Another useful command is 'SET', which causes FOCAL to store the symbol and its numerical value; when the symbol appears in an expression, the numerical value is used. Thus, the user may type,

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

These symbols may now be used to identify the values:

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

These two commands alone allow FOCAL to be used as a very powerful "desk calculator". Both the 'TYPE' and 'SET' commands will be explained more fully in the next section.

Several statements may be entered on the same line if they are separated by semi-colons (;). The FOCAL interpreter treats semi-colons as if they were line terminators in most situations.

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.

5. OUTPUT FORMAT

Numeric output may be printed in fixed-point, floating point (scientific), or integer formats. On initial loading, the default format is E18.06, floating point with six digits to the right of the decimal point and one to the left: X.XXXXXXE±XX, with some leading blanks. The user may set the format by using the % character in a 'TYPE' statement:

```
*TYPE %E18.06,A+B/C
```

The characters E, F or I may follow the % sign to indicate the type of format required; if the characters are not used, defaults to E or F formats are assumed.

Representation	Format	
% (only)	E14.06	
%MM.NN	FMM.NN	(MM field width)
%E	E14.06	
%F	F12.06	
%I	I7	
%EMM.NN	EMM.NN	(NN "fraction" width)
%FMM.NN	FMM.NN	
%IMM.NN	IMM	

FOCAL calculations are accurate to 10 decimal digits, regardless of the format specified; printed results are always rounded in the last displayed digit.

A variable name may be used as a format descriptor if the variable value defines a legitimate format. If the variable 'A' has the value 12.06, then:

```
*TYPE %EA,A,%IA,A
```

causes FOCAL to respond:

```
1.206000E+01        12
```

6. ALPHANUMERIC OUTPUT

Text strings are enclosed in quotation marks ("...") and may include most ASCII 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 a 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, a number sign (#) is used. For a form-feed (useful on the line printer), the ampersand (&) is used.

7. ABBREVIATIONS

All FOCAL commands may be abbreviated to the first letter; the interpreter does not examine past the first character of any command. It is good programming practice, especially when writing large FOCAL programs, to use the one-character representations. The following two procedures give equivalent results, except that the first runs faster and uses less computer memory:

```
1.01   S  ABC=3             1.01   SET   ABC=3
1.02   S  XYZ=7             1.02   SET   XYZ=7
1.03   T  %,ABC+XYZ         1.03   TYPE  %,ABC+XYZ
```

For the same reason, it is good practice to use one-character variable names where practical. Also, since line numbers require essentially two words of memory, while characters require only 1/3 word, it is good practice to combine several statements, separated by semi-colons, in each program line.

8. INDIRECT COMMANDS

If a line is prefixed by a line number, that line is not executed immediately; 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 999.99. The numbers 1.00, 2.00, etc., are illegal line numbers; they are used to indicate an entire group of lines. The number to the left of the point is called the group number; the number to the right is called the step number. Indirect commands are executed by using 'GO' or 'DO' commands which may be either direct or indirect. The 'GO' command followed by a line number causes FOCAL to execute the command at the specified line number. The 'GO' command alone, without a line number, causes FOCAL to go to the lowest numbered line to begin executing the entire program.

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 %I2,A+B
*GO
 STARTING= 3 FINISHED= 7
```

When the 'DO' command at line 1.3 was reached, the 'TYPE' command at 3.2 was performed and then the program returned to line 2.1.

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

```
*1.1 TYPE "A"
*1.2 TYPE "B"
*1.3 TYPE "C"
*1.4 DO 5.0
*1.5 TYPE " END"; GO 6.1
*5.1 TYPE "D"
*5.2 TYPE "E"
*5.3 TYPE "F"
*6.1 TYPE "."
*GO
ABCDEF 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,

```
*4.8 SET A=1; SET B=2
*6.3 TYPE %12.04, B/C+A
*4.9 SET C=1.31*.29
*GO
```

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.

9. ERROR DETECTION AND CORRECTION

FOCAL checks for a variety of errors, and, if an error is detected, prints an error code and the line number (if the error occurred in an indirect statement) in which the error was detected. Most errors are due to incorrectly constructed user statements and will be obvious on examination of the offending line.

The WRITE command without an argument can be used 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 questions marks will be printed. 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 III-17).

If the user types a bad character, or several bad characters, he can use the RUBOUT key (which echoes a backslash) to erase one character to the left each time the key is depressed. The entire input line may be erased by the CNTRL-U key (ASCII erase code) if used before the line is terminated by the RETURN or ALT-MODE key.

A line can be overwritten by repeating the same line number and typing the new 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 2.0
```

Used alone, without an argument, the 'ERASE' command causes FOCAL to erase the user's entire symbol table. The command 'ERASE ALL' erases all user input, program and variables.

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 III-16.

Use of the 'ERASE', 'MODIFY' and 'LIBRARY' commands, and entry of new lines cause all user-defined symbols to be lost because of text rearrangement operations performed by the interpreter.

10. SAVING FOCAL PROGRAMS

The user FOCAL programs may be saved by punching on paper tape using the 'WRITE' command, or in a DECtape (unit #6) library using the 'LIBRARY' commands. The DECtape library has a capacity of approximately 70 programs per tape. Library programs are accessed through a directory index by three-character alphanumeric names.

III. FOCAL COMMANDS

1. TYPE

The 'TYPE' command is used to request that FOCAL type out a text string, the value 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
*4.16 TYPE ABC
```

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

```
*9.19 TYPE %4.01, A1*2, E+2↑5, 2.51*81.1
```

The user may request printing 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:

```
*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. A statement may not be continued on a 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 using an exclamation mark (!); to insert a carriage return without a line feed, the user types a number sign (#); a form-feed is generated by the ampersand (&). Spaces may be inserted by enclosing them in quotation marks. These operations are useful in formatting output.

2. ASK

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

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

When step 11.99 is encountered by FOCAL, it waits for numeric input. The user then types a value in any format for the first identifier, followed by a line terminator. FOCAL then waits until 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.

FOCAL recognizes the value only when a terminator is entered. Therefore, a value can be changed before typing the terminator. This is done by typing a CNTRL-U, and they typing the correct value followed by the terminator.

Text string and format control may be included in an 'ASK' statement 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 'APP' (FOCAL recognizes the first three characters only) now has the value 25.

If CNTRL-D (the EOT character) is entered as the first character in response to an 'ASK', an error is generated which forces FOCAL out of program execution mode and into command mode. This can be useful in forcing the termination of a malfunctioning procedure.

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)
```

4. USE

The 'USE' command permits the FOCAL program to access all of the RAMOS monitor alphanumeric devices for input and output. The command must be followed by one of the modifiers 'INPUT' or 'OUTPUT', and a mnemonic device name:

```
10.1 USE INPUT PT; ASK !"ENTER A",A
10.2 USE OUTPUT LP; TYPE "A = ",A,!
```

The input and output mnemonic device names that may be used are specified in the RAMOS user's manual; they include: PT for paper tape input or output, LP for line printer output, BA for sequence control text input and TT for RML console teletype input or output.

If the device name is not used, the input and output functions default to the command terminal from which FOCAL was requested. This can be useful when FOCAL is called from a batch sequence but dialogue with the initiating terminal is required:

```
1.01 U I; U O
```

The entire indirect program may be listed on the line printer by typing: *U O LP;WRITE.

5. SET

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

```
*ERASE ALL
*3.4 SET A=2.55; SET B=8.05
*3.5 TYPE !,A+B
*GO
 1.06000E+01
```

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

```
*3.6 SET F=A
*3.7 SET G=(A+B)*2.2↑5
```

6. 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
C RAMOS FOCAL
 1.20 SET B=2
 1.30 SET C=4
 1.40 TYPE B+C
```

7. GO

The 'GO' command appearing alone requests that FOCAL execute the program starting 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 999.99, excluding 2.00, 3.00, etc.

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

```
*1.1 TYPE "A"
*1.2 TYPE "B"
*1.3 TYPE "C"
*1.4 TYPE "D"
*GO  1.2
BCD
```

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 specified group or line.

```
*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 entire group sequentially and returns control to the statement following the 'DO' command.

If 'DO' is written without an argument, FOCAL executes the entire indirect program, as with 'GO'.

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

IMPORTANT NOTE

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

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

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

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 indicated line if the expression is less than zero, to the second indicated line if the expression has a value of zero, or to the third indicated line 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 sample, 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.

VARIABLES AS LINE NUMBERS

Indirect Program line numbers appear as explicit parts of the 'GO', 'DO' and 'IF' statements. These line numbers may be represented by FOCAL variable names if it is ensured that the variables will have legitimate line number values when evaluated in these statements.

The statement: *FOR I=1,8,2;DO I will cause FOCAL to execute all lines in group 1, then all lines in groups 3, 5 and 7, in that order. Variables used as line numbers are rounded in the second fractional digit; 1.277 represents line 1.28, for example.

10. FOR

This command is used for 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 D and compared to the value of C. If A is less than or equal to C, the command after the semicolon is executed again. This process is repeated until A is greater than C, 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 the comma and value of D are omitted, it is assumed that the increment is one. If C and D are omitted, 'FOR' 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.

The values of the expressions, B, C and D, the initial value, final value, and increment, are fully determined when the 'FOR' statement is first encountered. Changing the values of B, C, or D in the loop can have no effect on the number of iterations performed.

11. 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.

12. QUIT

A 'QUIT' command causes FOCAL to leave program execution mode and return control to the user. FOCAL types an asterisk and the user may enter another command.

IMPORTANT NOTE

The 'QUIT' command should only be used in special circumstances such as debugging programs in development. After a program is operating satisfactorily, all 'QUIT' commands should be replaced by 'Z' to return control to the RAMOS monitor.

13. Z

The 'Z' command terminates execution of the FOCAL program and returns control to the RAMOS monitor.

14. COMMENT

Beginning a statement 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.

15. LIBRARY

The 'LIBRARY' command group contains six operations to save and retrieve user FOCAL programs, to print a summary of the library contents, and to maintain the library index. The FOCAL library is maintained on DECtape unit #6.

15.1 LIBRARY OUT nam transfers the user FOCAL program to the library and makes a corresponding entry of "nam", a one- to three-character alphanumeric identifier, in the library index. If a program with name "nam" is already in the library, it is replaced.

15.2 LIBRARY IN nam retrieves the program with the name "nam" from the FOCAL library.

15.3 LIBRARY XCT nam retrieves the program with the name "nam" from the library and starts execution at the lowest-numbered line.

15.4 LIBRARY DELETE nam removes the entry "nam" from the library index, freeing the corresponding storage area in the library.

15.5 LIBRARY ERASE clears all entries from the index, ordinarily used only to prepare a new tape for use as a FOCAL library tape.

15.6 LIBRARY LIST causes the names of all programs in the library index to be printed on the current output device.

16. MODIFY

Frequently, only a few characters in a particular line require changes. To facilitate this, and to eliminate the need to replace the entire line, the FOCAL programmer may use the 'MODIFY' command; 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 FOCAL 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 found.

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 TAB (CNTRL-I); this will cause the search to proceed to the next occurrence, if any, of the search character.

c. Type a CNTRL-Z; 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 CNTRL-U 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' and 'MODIFY' commands are generally used only in immediate mode since they return to command mode upon completion.

Notice the errors in line 7.01 below.

```
*7.01 JACK AND BILL W$NT UP THE HALL
*MODIFY 7.01
 JACK AND B/JILL W$/ENT UP THE HA/ILL
*WRITE 7.01
07.01  JACK AND JILL WENT UP THE HILL
```

[*] On full-duplex RAMOS terminals; some terminals are wired half-duplex, and the control characters will be echoed by the hardware. In either case, the FOCAL action is the same.

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 used the CNTRL-Z key 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.

17. USING THE TRACE FEATURE

As stated in Section II-9, 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 three lines are printed:

```
*1.1 SET A=1
*1.2 SET B=5
*1.3 SET C=3
*1.4 TYPE %I3, ?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.

18. FOCAL FUNCTIONS

RAMOS FOCAL supports the six common functions: square root, natural logarithm and exponent, and sine, cosine and arctangent, and three numeric modification operators: integer, absolute, and sign value. Two functions that utilize RAMOS monitor facilities, time-of-day and "common" register access, are also included.

All function names are three-character identifiers which may be used in the same manner as normal variable names in arithmetic expressions, except that they must have parenthesized arguments. The arguments may be simple variable names or complex expressions. With the exception of 'REG', the register access function, function names may not be used as the destination in value assignment commands; they may not appear on the left side of the equal sign (=) in the 'SET' and 'FOR' statements.

18.1 SQUARE ROOT

The square root function 'SQT(X)' returns the square root of the absolute value of the argument:

```
*S A=SQT(B)
```

18.2 NATURAL LOGARITHM

The logarithm function 'LOG(X)' returns the value of the natural logarithm of the absolute value of the argument. If the argument has zero value 'LOG' returns zero.

```
*S A=LOG(B)
```

18.3 NATURAL EXPONENTIAL

The exponentiation function 'EXP(X)' returns the value of e^x, the natural exponent.

```
*S A=EXP(B)
```

18.4 GENERAL EXPONENTIAL

The general exponentiation function is included in FOCAL as an arithmetic operator (↑). This operator uses the RAMOS logarithm and exponential functions:

$$A{\uparrow}B = |A|^B = \exp\,(B*\log(|A|))$$

The arguments, A and B, may have any value, but if the base, A, is negative, its absolute value is used.

18.5 SINE AND COSINE

The trigonometric functions, 'SIN(X)' and 'COS(X)', require arguments in radians.

```
*S TAN=SIN(X)/COS(X)
```

18.6 ARCTANGENT

The arctangent function 'ATN(X)' returns the principle value of the arctangent of the argument, in the range $\pm\pi/2$, in radians.

```
*S X=ATN(Y)
```

18.7 ABSOLUTE VALUE

The absolute value operator 'ABS(X)' evaluates the argument, then forces the sign positive.

18.8 INTEGER VALUE

The integer value operator 'ITR(X)' evaluates the argument and returns the result rounded to the nearest integer value. The integer value range is ± 65535.

18.9 SIGN VALUE

The sign evaluation operator 'SGN(X)' returns a value of +1 for positive arguments (including zero), and -1 for negative arguments.

18.10 TIME-OF-DAY

The time-of-day function 'TIM()' returns the time of day, in seconds, from the RAMOS system clock. The value of the argument is meaningless to the 'TIM' function, but FOCAL requires that the parentheses be used.

18.11 REGISTER ACCESS

The 41 RAMOS "common" registers may be used as FOCAL variables. The name 'REG' is reserved for this purpose. 'REG' is treated as a function name and must have a parenthesized argument whose value is in the range zero through forty. The registers are used to transfer values between different RAMOS program modules, but must be used with caution since they are available to all operating programs.

```
*S REG(3)=REG(2)+REG(1)*A
```

IV. FOCAL PROGRAM EXAMPLES

1. PRINTING TIME-OF-DAY

This example is a simple procedure illustrating the general format for FOCAL programs and the use of FOCAL functions. The time-of-day (in seconds) is read from the RAMOS clock and printed on the user output terminal:

```
C RAMOS FOCAL
   1.10 S T=TIM()
   1.20 S H=ITR(T/3600-.5)
   1.30 S T=T-H*3600
   1.40 S M=ITR(T/60-.5)
   1.50 S T=T-M*60
   1.60 T %I4,!,H,M,T,!!
   1.70 Z
```

Line 1.1 reads the time to the variable 'T'. Line 1.2 extracts the "hours" information; since 'ITR' rounds the operand, 1/2 must be subtracted to simulate truncation. Line 1.6 sets the output format to I4 and prints hours, minutes and seconds on a new line.

2. LOG-LOG INTERPOLATION

Many useful functions appear linear when presented in log-log format. In particular, the efficiency dependence on gamma energy of Ge(Li) detectors exhibits several such segments. The following program is used in preparing efficiency tables by interpolating between end-points of "linear" regions:

```
1.10 T !! "LOG-LOG INTERPOLATION ROUTINE"
1.20 A !"START ENERGY X(1) = ",X(1)
1.21 A !"EFFICIENCY Y(1)   = ",Y(1)
1.23 A !"FINAL ENERGY X(2) = ",X(2)
1.25 A !"EFFICIENCY Y(2)   = ",Y(2)
1.32 A !"ENERGY INCREMENT  = ",DX
1.35 A !"MULTIPLIER  = ",MP
1.37 S M=LOG(Y(1)/Y(2))/LOG(X(1)/X(2))
1.38 S B=LOG(Y(1))-M*LOG(X(1))
1.40 D 2
1.45 T !&;U O;Q
2.10 U O LP;T &
2.15 DO 1.1
2.20 T !!"ENERGY (KEV)    EFFICIENCY"
2.30 D 4
4.10 S X=X(1)
4.40 S Z=M*LOG(X)+B
4.50 S Y=EXP(Z)
4.52 S Y=Y*MP
4.55 T !%F8.02,X,%E18.03,Y
4.60 S X=X+DX
4.70 I (X-X(2))4.4,4.4,4.9
4.90 R
```

Line 1.10 TYPEs the program title on the user's terminal, and lines 1.20 through 1.35 ASK for the initial and final values of the segment, and the energy increment required. Lines 1.37 and 1.38 SET the values of the slope and intercept from the input data.

Line 1.40 makes a "subroutine call", telling FOCAL to DO all the lines in group 2.0. On returning from group 2.0, line 1.45 will be executed to TYPE a form-feed (the output will have been set to the line printer), reset the output device to the user teleprinter, and QUIT. Use of the 'QUIT' command is not ordinarily recommended, but in this case it allows the user to restart (GO) and interpolate another segment of the curve. 'Z' will have to be entered as a direct command to release FOCAL and return to the RAMOS monitor.

Group 2.0, lines 2.10 through 2.30, is executed as a subroutine from line 1.40. Line 2.10 switches the output device to the line printer (USE OUT LP) and starts a new page (T &). Line 2.15 prints the program title by executing line 1.10 as a subroutine, and line 2.20 prints column titles. Line 2.30 executes group 4.0 as a subroutine to calculate and print the pairs of values.

On returning from group 4.0, the command following the 'DO' command at line 2.30 should be executed (see section III-18), i.e., the program should continue at line 4.10. However, group 2.0 is called as a subroutine (from 1.40), is completed when line 2.30 is executed, and control is thus passed back another level to the line following the 'DO 2' statement.

Group 4.0 performs the computations and printing. Line 4.10 SETs the initial value of X, and lines 4.40 through 4.52 calculate the interpolated value at X. Line 4.55 prints the values, X in fixed-point, and Y in floating-point format.

Line 4.60 SETs X to its next value, and the new value is tested in line 4.70. IF the value is less than or equal to the limit, control is returned to line 4.40. IF it is greater than the limit, line 4.90 is executed to RETURN from the group 4.0 subroutine.

2.1 A VARIATION

In the group 4.0 subroutine, the value of X is incremented by explicitly adding the value of DX in line 4.60, and the test for completion is made using an 'IF' statement in line 4.70; the initial value is set in line 4.10. All of these operations may be combined using the 'FOR' statement. For example, group 4.0, above, can be replaced by:

```
4.40 F X=X(1),X(2),DX;D 5
5.10 S Y=MP*(EXP(M*LOG(X)+B))
5.20 T !%8.02,X,%E18.03,Y
```

Line 4.40 in this example sets the initial value of X, then executes group 5.0 as a subroutine (note that the "subroutines" are now nested three deep). X is incremented by DX, executing group 5.0 each time until the limit value X(2) is exceeded.

Note that the computations in lines 4.40 through 4.52 of the first example are performed in this variation in one line, using a more complex expression. Note also that since group 4.0 consists in only one line, the 'FOR' statement, line 4.40, should replace the 'DO 4' statement, line 2.30. This would save program and stack space but complicate the example.

2.2 ANOTHER VARIATION

Group 5.0 of the second example can be merged into the 'FOR' statement in line 4.40, immediately above:

```
4.40 F X=X(1),X(2),DX;T !%8.02,X,%E18.03,MP*EXP(M*LOG(X)+B)
```

The 'DO 5' statement is replaced by a 'TYPE' command which evaluates (but does not save) Y as a part of the procedure required to print it. Again, a "better" procedure would result from replacing line 2.30 with this line 4.40.

3. NEUTRON FLUX DETERMINATION

In many RAMOS applications, data are pre-processed by primary data reduction modules such as 'STRIP' and 'GAUSS', and the "raw" results are passed to conversational post-processor modules for further operations and report formatting.

The following FOCAL program example is used, as the third in a sequence of data reduction operations, to calculate the neutron fast flux measured by a nickel wire irradiated in a nuclear reactor. It is assumed that a previous FOCAL program has calculated the irradiation history and decay correction term and has left the result in "common" register #40. It is also assumed that the gamma spectrum from the wire has been operated on by the 'STRIP' module to determine the area and standard deviation of the area in the ^{58}Co peak; the 'STRIP' results are passed in registers #0 and #10.

```
1.12 U I,S W1=REG(10)*100,S S=REG(40)
1.14 S N=7.05E18;S XS=.137E-24
1.17 T !"          NRL DATA REDUCTION CODE
1.20 A !"DATE: MTH = ",MM," DAY = ",DD," YR = ",YY
1.30 A !"SAMPLE ID = ",Y
1.32 A !"RUN TIME (SEC) = ",T
1.34 A !"NEUTRON FLUX MULTIPLIER = ",D
1.36 A !"WEIGHT OF WIRE (MG) = ",WT
1.40 S A=D*1E7*REG(0)/T;S FL=A/(N*S*XS*WT)
1.50 D 3
1.60 T !&;Z
3.02 U 0 LP;T !;D 1.17
3.10 T !!,%I3,"          DATE = ",MM," -",DD," -",YY
3.15 T %I5,!"      SAMPLE ID = ",Y
3.20 T !" RUN TIME (SEC) = ",T
3.25 T %8.03,!"FLUX MULTIPLIER = ",D
3.30 T !" WEIGHT OF WIRE = ",WT
3.35 T !!"ISOTOPE     DIS/SEC     % STD DEV       FAST FLUX"
3.40 T !"CO-58",%,A,%F10.02,W1,"     ",%,FL
```

The program is ordinarily executed in the batch mode with the 'STRIP' procedure, so that the 'U I' command (USE INPUT) in line 1.12 is required to switch the input device from the sequence control text to the user terminal for dialogue. The 'Z' command (line 1.60) is required, instead of 'QUIT', to return control to the monitor on completion of the procedure.

Line 1.12 sets the input device to the (default) user terminal and extracts two values from the "common" registers. Note that the 'REG(X)' function is used exactly as if it were a FOCAL variable. Line 1.14 SETs the values of some constants required in the calculation.

Line 1.17 prints the problem title, and lines 1.20 through 1.36 request input data from the user. The simple calculations are performed in line 1.40, the results are printed by calling a "subroutine" in line 1.50; line 1.60 ejects the (line printer) page and returns control to RAMOS.

Group 3.0 is used as an output routine, called from line 1.50. Line 3.02 selects the line printer as the output device and prints the problem title by re-executing line 1.17. Lines 3.10 through 3.30 summarize the input data on the printer, line 3.35 prints column headings, and line 3.40 prints the numeric data with several format changes. On completion of group 3.0, control returns to line 1.60 where the procedure is terminated.

APPENDIX A

FOCAL COMMAND SUMMARY

Command		Example of Form	Explanation
ASK	A	ASK X,Y,Z A P,Q,R	Requests FOCAL to wait for numeric input from current input device.
COMMENT	C	COMMENT	If a line begins with the letter C, the remainder of the line will be ignored.
DO	D	DO 4.1	Execute line 4.1; return to command following DO command.
		DO 4.0	Execute all group 4 lines; return to command following DO command, or when a RETURN is encountered.
ERASE	E	ERASE E	Erases the symbol table.
		ERASE 2.0 E 2	Erases all group 2 lines.
		ERASE 2.1 E 2.1	Deletes line 2.1.
		ERASE ALL E A	Deletes all user input.
FOR	F	FOR i=x,z,y;(commands)	Where the command following is executed at each new value.
		FOR i=x,z;(commands)	x = initial value of i y = value added to i until i is greater than z (default is 1)
GO	G	GO	Starts indirect program at lowest line number.
		GO 3.4 G 3.4	Starts indirect program at line 3.4.
GO?	G?	GO?	Starts at lowest numbered line and traces entire indirect program until another ? is encountered, until an error is encountered, or until completion of program.

Command		Example of Form	Explanation

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 transferred to the second line number.
			
			If X is greater than zero, control is transferred to the third line number.

LIBRARY	L	L E	Erase library directory.
		L L	List library directory contents.
		L O nam	Output current indirect program to library with name "nam".
		L D nam	Delete directory entry "nam".
		L I nam	Input program with name "nam" from program library.
		L X nam	Load and execute program with name "nam".
MODIFY	M	MODIFY 1.15 M 1.15	Enables editing of any character on line 1.15 (see below).
QUIT	Q	QUIT	Returns control to the user.
RETURN	R	RETURN	Terminates DO subroutines, returning to the original sequence.
SET	S	SET A=5/B*C; S A=3	Defines identifiers in the symbol table.

TYPE	T	TYPE A+B-C;
		T A+B-C;
			Evaluates expression and types out result in current output format.

		TYPE A-B, C/E;	Computes and types each expression separated by commas.
		TYPE "TEXT STRING" T !"MORE TEXT"	Types text. May be followed or preceded by ! to generate carriage return-line feed, # to generate carriage return, & to generate form-feed.

Command		Example of Form	Explanation
USE	U	U I	Sets input device to user command terminal.
		U I inp	Sets input device to monitor input device "inp".
		U O	Sets output device to user command terminal.
		U O out	Sets output device to monitor output device "out".
WRITE	W	WRITE	FOCAL types out the entire indirect program.
		WRITE 1.0	FOCAL types out all group 1 lines.
		WRITE 1.1 W 1.1	FOCAL types out line 1.1
Z	Z	Z	Returns control to the RAMOS monitor.

SUMMARY OF FUNCTIONS

Square Root	SQT(x)	Root of absolute value of x.
Absolute Value	ABS(x)	Absolute value of x.
Sign Part	SGN(x)	-1 if x negative, +1 if positive or zero.
Integer Part	ITR(x)	x is rounded to nearest integer.
Exponent	EXP(x)	Natural exponential, e^x.
Sine	SIN(x)	Sine of x; x in radians.
Cosine	COS(x)	Cosine of x.
Arctangent	ATN(x)	Principal value arctangent.
Logarithm	LOG(x)	Natural logarithm of absolute value of x.
Register Access	REG(x)	RAMOS "common" registers, x in range zero through 40.
Time-of-day	TIM()	Time-of-day, in seconds, from RAMOS clock.

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 TAB. FOCAL will proceed to the next occurrence of the search character.

c. Type a CNTRL-Z. After this, the user will enter a new search character.

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

e. Type CNTRL-U. 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.

ASK/TYPE CONTROL CHARACTERS

%	per-cent	Format delimiter (section II.5)
"	double quote	Text delimiter (section II.6)
!	exclamation	Carriage return and line feed
#	number sign	Carriage return only
&	ampersand	Form feed
$	dollar	Type the symbol table contents
SPACE		Terminator for names
,	comma	Terminator for expressions
;	semi-colon	Terminator for commands
RETURN		Terminator for lines
ALT-MODE		Terminator for lines

APPENDIX B

PROGRAM AND VARIABLES STORAGE FORMAT

1. NUMERIC STORAGE FORMAT

All numeric values in RAMOS FOCAL are stored and evaluated in three-word floating mode format, allowing a precision greater than ten decimal digits and a mágnitude range spanning 78000 decades.

Five memory words are required for each variable generated by FOCAL, one each for the variable name and the subscript, and three for the variable value:

name	three sixbit characters, right adjusted, blank filled.
subscript	two's-complement integer.
low mantissa	
high mantissa	and sign of mantissa.
two's exponent	two's-complement integer.

2. PROGRAM STORAGE FORMAT

Following the line number and leading blanks, each line of an indirect program is stored exactly as it is entered, three characters per word in sixbit ASCII format. In addition to the character storage, one word is required for the line number and one word to indicate the location of the next line in numeric sequence. The line numbers are stored with the group number in the high eleven bits of the word, and the step number in the low seven bits:

word #1	address of next line		
#2	group no.		step no.
#3	1st	2nd	3rd
#4	4th	5th	
etc.			

etc.

3. ESTIMATING PROGRAM SIZE

The size of a FOCAL program with all variables can be calculated from:

$$5*(\text{no. of var.}) + \sum_{\substack{\text{all}\\ \text{lines}}} \left(2 + \frac{\text{char. in line}}{3}\right)$$

The amount of core available at any stage in program entry can be determined by typing the direct statement:

```
*F I=1,1E5;S X(I)=0
```

An error code will be given when the available memory has been exhausted; one then types:

```
*T !"REMAINING =",5*I
```

and FOCAL will print the number of memory words remaining for program expansion or variable storage.

An operating program requires several words for the interpreter to use for stack operations, so not all of the memory reported by this operation is really available to the user.

APPENDIX C

COMMENT ON ERROR MESSAGES

FOCAL prints error messages in one of two forms: for direct commands, only an octal error code is printed; if an error is detected while executing an indirect program, the octal error code is printed with the number of the line in which the error is detected.

The error code is a six-digit octal number of which only the low order four digits are significant. The error message: *432736 18.03, indicates that error #2736 occurs in line 18.03. The octal error number represents the memory address, in the FOCAL interpreter program, where the error is detected.

The FOCAL error codes are not summarized for several reasons:

a. The errors commonly made by FOCAL users are usually easily recognized on examination of the offending line; an error summary is not ordinarily required.

b. The error codes represent memory addresses in the interpreter. RAMOS FOCAL is subject to internal change and upgrading, and the error codes are therefore not fixed. There are several ways each in which several error types are recognized, and the task of updating an error table is, in view of (a), not worth the effort.

c. In situations where errors are not recognized, tracing the error code to the assembly listing of the interpreter will define the error more completely than a reasonably succinct error code table.

Some of the more common user errors are:

Bad line number format
Illegal command used
Nonexistent line referenced by DO, GO or IF
Nonexistent group referenced by DO
Bad argument for MODIFY
Command input buffer exceeded
Bad argument for ERASE
Variable storage exceeded
Bad argument in IF command
Missing operator in an expression
Bad argument in FOR, SET, or ASK
Operator missing before parenthesis
Error to left of equal sign
Parentheses do not match
Function not followed immediately by parens
Double operators in an expression
Error in FOR command format