FOCAL-11
User's Manual

DEC-11-LFOCA-F-D

pdp11

digital

FOCAL-11
User's Manual

DEC-11-LFOCA-F-D

digital equipment corporation · maynard. massachusetts

First Printing, September 1975

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

CONTENTS

PREFACE

While this manual covers the language and usage of FOCAL-11, it would be helpful to the reader to be familiar with the RT-11 system software as described in the RT-11 System Reference Manual (DEC-11-ORUGA-C-D). Paper tape users should become familiar with the PDP-11 Paper Tape Software Programming Manual (DEC-11-XPTSA-A-D).

For readers who may not be familiar with terms such as subscript, variable, etc., there is a glossary of terms in Appendix J. Appendix K summarizes the major differences between this version of FOCAL-11 and previous versions.

CHAPTER 1

INTRODUCTION TO FOCAL-11

FOCAL-11 (FOrmula CALculator) is an easily learned high-powered programming language. This dynamic combination of strength and simplicity makes FOCAL-11 an ideal language for students, managers, scientists, and technicians who do not have time to learn complex languages but require problem-solving capabilities.

1.1 THE PHILOSOPHY OF FOCAL-11

In general there are two main classes of problems which are usually programmed for computer solution. The first of these are problems which require numerous operations to be performed. These problems are usually programmed in a high level language such as FORTRAN. The other main class of program generally solved by computers are those which are capable of solution by hand, but are used often enough to warrant the time spent in their programming. It is this latter class of programs to which FOCAL addresses itself. In addition, due to FOCAL's ease of use, those problems which have been done in the past by hand because of the time required to program them are now candidates for solution by a computer. With FOCAL, the user is able to write a program, correct it when necessary, and obtain results in usually a single on-line session with FOCAL.

FOCAL is the actual program which is running on the PDP-11 computer. The program functionally consists of three major parts. The first of these is the command interpreter. This section reads the FOCAL command and actually performs the indicated operation, hence the term interpreter. The second part of FOCAL, is the program storage area. This is the area used by FOCAL to remember the program written by the user. The third part of the system is the variable storage area. This area shares the same space in the computer's memory as the program storage area. When FOCAL is loaded, all memory not occupied by the command interpreter is set aside for program and variable storage.

Appendix E describes the method used for loading FOCAL into memory. Whenever FOCAL displays an asterisk (*) upon the terminal, FOCAL is indicating that it is ready to respond to a user's directive. The user's response can either be a FOCAL COMMAND (described in the next section) or a directive to save a set of FOCAL COMMANDs into the program storage area for later use (also explained further in the next section).

In general, whenever the user wishes to return to the point where a directive is expected, the user can strike CNTRL/C (depress the CNTRL key while striking the C) twice. In the RT-11 version, the user must also type RE followed by the RETURN key. Appendix E further describes methods of restarting FOCAL.

1.2 FOCAL PROGRAM STRUCTURE

A FOCAL PROGRAM is a collection of commands which have been organized to perform a given task. The method by which these commands have been organized is the key to FOCAL programming.

A FOCAL STATEMENT is a set of one or more FOCAL COMMANDs placed on a line. The FOCAL STATEMENT is the smallest section of a program which may be referenced by FOCAL. For example, the following is a FOCAL statement.

```
SET A=15;TYPE A;QUIT
```

The FOCAL COMMANDs: SET, TYPE, and QUIT would be executed in that order. Note that when more than one FOCAL COMMAND is placed on a single line, they are separated by semi-colons (;).

When a user wishes to store a FOCAL STATEMENT for later use, a statement number is assigned by the user as a means of referring to it. The statement number may be any value from 1.01 to 99.99 in .01 increments, with the exception of those numbers ending in ".00". Line numbers do not require that two digits be specified after the decimal point. The line number 2.1 is equivalent to 2.10. When a user wishes to store a FOCAL statement, he or she simply types the line prefixed by the appropriate statement number. For example:

```
1.1 SET A=1.5;TYPE A;QUIT
```

This FOCAL STATEMENT would be entered into the FOCAL PROGRAM storage area, and could be referenced by FOCAL COMMANDs using the number 1.1. If another FOCAL STATEMENT with this line number currently exists in the program storage area, it will be replaced when the new line is entered.

All FOCAL STATEMENTs which have the same integer portion of their line numbers are collectively referred to as a FOCAL GROUP. This allows certain FOCAL COMMANDs to refer to these FOCAL STATEMENTs as if they were all a single long FOCAL STATEMENT. FOCAL GROUPS are referenced by using the integer portion of the line number. For example:

```
1.1 SET A=1.5;TYPE A
1.2 SET B=2.4;TYPE B
1.3 SET C=A*B;TYPE C
2.1 SET D=A+B;TYPE D
2.2 TYPE "THE END"
2.3 QUIT
```

The FOCAL COMMAND: "ERASE 1.2" would remove just statement 1.2, while "ERASE 1" would delete lines 1.1, 1.2, and 1.3.

```
                  1.05 COMMENT PROGRAM TO CALCULATE C=FSQT(A*A+B*B)
          GROUP   1.10 TYPE !"ENTER A,B"
                  1.20 DO 2
                  1.30 TYPE !"C",C,!; QUIT
PROGRAM
                  2.10 COMMENT PERFORM CALCULATIONS
          GROUP   2.20 SET C=FSQT(A*A+B*B)
                  2.30 RETURN
```

1.3 FOCAL COMMANDS

Each FOCAL STATEMENT consists of one or more FOCAL COMMANDs with their associated arguments, if any. Every FOCAL COMMAND may be abbreviated by a single character and requires at least one space between the command and its arguments. This will cause memory to be conserved, as one word of memory is required for each two characters in the user's program. The following table lists the commands available in FOCAL-11.

Command	Abbreviation	Example	Function
ASK	A	ASK X	Used to assign values to variables from the keyboard.
COMMENT	C	C EXAMPLE	Used for comments or non-executable program steps. Allows the user to document his program.
DO	D	DO 2	Used to direct program execution to a statement or group of statements which will be executed until either a "RETURN" command is executed or the end of a statement or group is reached. When this occurs, the next statement executed is the statement following the last DO command encountered.
ERASE	E	ERASE 4.1	Used to erase part of a program or an entire program.
FOR	F	FOR I=1,5;SET X=X+I	Execute the remaining commands on the line while incrementing the value of the variable specified in the FOR command over the defined limits. This is used for loop control.
GO	G	GO 2.4	Begin executing commands at the statement specified. Used to direct program control to the lowest line number, or to a specific group or statement.

Command	Abbreviation	Example	Function
IF	I	IF(A)1.1,1.2,1.3	Conditionally directs program execution using the results of testing the sign of A. (-,0, or +)
KILL	K	KILL	Stops the program and I/O activity.
LIBRARY	L	LIBRARY RUN TEST	Allows the program to access file structured devices.(RT-11 version only)
MODIFY	M	MODIFY 1.1	Used to alter words or characters in a stored program statement.
OPERATE	O	OPERATE TK	Used to select non-file structured I/O devices for I/O.
QUIT	Q	QUIT	Used to terminate program execution and return control to user.
RETURN	R	RETURN	Used to return program execution to the statement following the last "DO" command executed.
SET	S	SET A=1.531	Assigns the value of the result of an expression to the variable.
TYPE	T	TYPE "HI"!,A	Send output to the currently selected output device. Used to print text, results of calculations, and values of variables.
WRITE	W	WRITE 1	Used to list part or all of a program.
XECUTE	X	XECUTE FPRM(3,1)	Used to call and execute functions.

For the convenience of the user, a detailed FOCAL-11 Command Summary is included in Appendix C.

Several commands may be put together on the same FOCAL STATEMENT as long as the total statement does not exceed 79 characters in length. Semi-colons are used to separate FOCAL-11 commands within a statement.

All FOCAL STATEMENTs must be terminated by the carriage-return character. A FOCAL COMMAND may be terminated by either a semi-colon (;) or a carriage-return. For example:

```
1.10 FOR I=1,5;DO 2;TYPE "I",I,!
```

There are a few exceptions to this rule, viz., the COMMENT, ERASE, and certain LIBRARY commands can only be terminated by a carriage-return.

1.4 FOCAL ERROR DIAGNOSTICS

FOCAL has the ability to detect many user errors. Whenever one of these errors is detected, FOCAL displays an error message of the form: ?NN AT LL.LL and follows this with an asterisk (*). This message informs the user of two facts. The first is the portion of the message which describes the nature of the error which occurred. A complete list of these error numbers (?NN) will be found in Appendix A along with a short description of their meaning. The other portion of the error message (AT LL.LL) informs the user as to where in the program the error was detected. If a line number (LL.LL) is zero (0.00), this means that a directive issued by the user when an asterisk (*) appeared was in error.

The special error message ?00 AT LL.LL is used to inform the user that FOCAL has been restarted.

CHAPTER 2

FOCAL-11 ARITHMETIC

2.1 NUMBERS

A FOCAL-11 number may be any decimal number between 10↑(-38) and 10↑(+38). Numbers may be optionally signed (+ or -), contain a decimal point with a fractional part or be in exponential format with a mantissa and exponent (see Data Formats, paragraph 6.1). In single-precision FOCAL-11, all numbers are to approximately seven significant digits. The following numbers are identical in single-precision FOCAL-11.

```
60
60.00
6E+01
600E-01
60.00003
6E1
```

Additionally, FOCAL-11 allows for the specification of octal constants. An octal constant is specified by the character "@" followed by up to six (6) digits in the range 0 to 7. Octal numbers are treated as a signed 16 bit integer, and can be used in any arithmetic expression.

```
*ERASE
*SET A=@400
*SET B=@177777
*SET C=-@400
*TYPE $
 S A=  256.0000
 S B=   -1.0000
 S C= -256.0000
 S &=    0.0000
*
```

When an octal constant is to be input by the ASK command, a leading plus or minus sign is required.

```
*ERASE ALL
*1.1 ASK A
*1.2 TYPE A; QUIT
*GO
:+@400
=  256.0000*
```

*** WARNING ***

> Octal numbers are used to represent integer values within the range of -32,768 to 32,767. Therefore, the expression @077777+@1 does not equal @100000. @100000 represents -32,768, while @077777 represents +32,767. If a variable has the value of +32,767 (@077777), and has 1.0 added to it, the resulting value (+32,768) can not be represented as an integer, and if used in the FX function, or as an array subscript, for example, an integer overflow (?38) will result.

2.2 VARIABLE NAMES

When programming by computer, it is usually necessary to refer to the result of a calculation by a symbolic name. These names are called variables because the same name may be assigned different values at different times in a user's program. FOCAL-11 variable names may consist of either one or two characters. The first character must always be alphabetic; however, it cannot be an F because FOCAL-11 reserves that character for functions (see FOCAL-11 Functions, Chapter 4). The second character may be either alphabetic or numeric. The user may write variable names consisting of more than two characters, but FOCAL-11 uses only the first two characters to identify the variable. Therefore, the first two characters must be unique.

A variable is represented internally as a binary fraction with an exponent.

```
*SET A=546789
*SET B=123456
*SET C1=15
*SET C2=30
*SET DEPTH=10
*SET DISTANCE=C1+C2
```

Variables not starting with the letters A or F may be used in place of a line number. The reason for this restriction is that the letter "A" is used in the DO ALL command and the letter "F" is used in functions.

```
*2.01 SET Z=5

*2.02 DO Z
```

The above commands are the same as writing

```
*2.02 DO 5.0
```

The ampersand character (&) is a special variable which is used by the FSBR function. See Chapter 4 for details of its use. (This variable is not cleared by the ERASE command.)

2.3 SUBSCRIPTED VARIABLES

At times, it is convenient to refer to a table of values by a single name, and refer to an element of this list by a number corresponding to the position of this element. This is the function of subscripted variables or arrays. The position of a value in an array is defined by the use of subscripts which are placed immediately after the variable name used to identify the array. These subscripts are enclosed in parentheses. FOCAL-11 permits the use of either singly or doubly subscripted variables.

```
*SET AR(0)=5
*SET AR(1)=10
*SET AR(2)=15
*SET AR(3)=20
*SET AR(4)=25
*SET AR(5)=30
*FOR X=0,5; TYPE AR(X),!
=   5.0000
=  10.0000
=  15.0000
=  20.0000
=  25.0000
=  30.0000
*
```

In the above example, subscripts are used to set up an array called AR. Any element in the array (AR) can be represented by a subscript in the range 0 to 5. The first element in an array always has a subscript of 0.

It is not necessary that all elements of an array be used. FOCAL will only reserve space for the elements of an array which are actually used.

A subscript may be a number, another variable, or an expression. If it is a number, it must be in range +127 to -128 for doubly subscripted variables and +32,767 to -32,768 for singly subscripted variables. For an explanation of the symbol table typeout for subscripted variables using the TYPE $ command, see Appendix E.

2.4 LITERAL CONSTANTS

Some programs may require an alpha-numeric keyboard response to a question asked during program execution. The answer typed to the question determines the next command to be executed (for example, in an initial dialogue). For this purpose, alphanumeric constants may be used in an IF statement to direct program execution as shown below:

```
*1.10 TYPE "DO YOU WANT A LINE?",!
*1.20 ASK "TYPE YES OR NO",ANS,!
*1.30 IF (ANS-0YES)2.1,2.2,2.1
*2.10 QUIT
*2.20 TYPE "----------",!
*2.30 GO 1.1
*GO
DO YOU WANT A LINE?
TYPE YES OR NO:YES
----------
DO YOU WANT A LINE?
TYPE YES OR NO:NO

*
```

The 0 in 0YES indicates to FOCAL-11 that the characters following the 0 should be interpreted as numbers, not as a variable name.

The ASCII characters A through Z have the values of 1 through 26, except for the character E which is interpreted by FOCAL-11 as an indication of floating-point format (see Data Formats). Therefore, in the above example, 0YES will be interpreted as 25E+19, where Y=25 and S=19

2.5 ARITHMETIC OPERATIONS

To print the results of arithmetic calculations, the user issues the FOCAL-11 command TYPE. This is followed by a space and the data to be calculated followed by the RETURN key. For example:

```
*TYPE 6+10-3-1        User presses RETURN key.
=   12.0000*          FOCAL-11 prints the answer.
```

The above example shows two of the arithmetic operations (+ and -) performed by FOCAL-11.

Unless otherwise indicated, FOCAL-11 mathematical computations retain an accuracy of approximately seven significant digits. FOCAL-11 is also available in double precision. This version provides the user with approximately seventeen digits of accuracy.

2.5.1 Priority of Arithmetic Operations

Arithmetic operations are performed from left to right except when the operation to the right has priority or when parentheses are used.

The priorities of FOCAL-11 arithmetic operations are:

First priority	-	exponentiation (↑)
Second priority	-	multiplication (*)
Third priority	-	division (/)
Last priority	-	addition (+)
	-	subtraction (-)

When FOCAL-11 evaluates an expression which includes several arithmetic operations, the above order of priority is followed. Therefore, FOCAL-11 evaluates

```
25+5*2+5 as 25+(5*2)+5

     or

=   40.0000*
```

because multiplication (*) has a higher priority than addition (+).

A negative number may only be raised to a positive integer power.

2.5.2 Use of Parentheses

The order of arithmetic operations may be altered by using parentheses: () or [] or <>. On some terminals, [and] are formed by SHIFT/K and SHIFT/M, respectively. FOCAL always computes the expression within parentheses first.

If the expression contains parentheses within parentheses (called "nesting"), FOCAL-11 executes the contents of the innermost parentheses first and works outward.

```
(5*<2+3>-5)↑2
```

is the same as

```
(5*5-5)↑2
```

and as

```
(25-5)↑2
```

and as

```
20↑2
```

and as

```
400.00
```

CHAPTER 3

FOCAL-11 COMMANDS

3.1 INPUT/OUTPUT COMMANDS

3.1.1 TYPE Command

The TYPE command is used to print results of calculations, values of variables, text or character strings, and variable tables. TYPE may also be used to print combinations of text and variables.

The format of this command is the FOCAL COMMAND: TYPE followed by a series of expressions, variables, constants, or text strings. Any variable, constant, or expression, except text, in a TYPE or ASK command must be followed by either a comma, semicolon, or carriage return. In addition, the exclamation mark (!) causes a carriage return (CR) and line feed (LF) to be output to the terminal.

The user may command FOCAL-11 to print all of the user-defined variables (variable table) by using the TYPE command and a dollar sign ($). This feature may be used as a debugging aid to determine the value of all stored variables in the user's program.

Result of a Calculation

```
*TYPE 1+1
=   2.0000*
```

Value of Variable or Variables

```
*SET N=5*5;  SET M=30
*TYPE N,M
=   25.0000=   30.0000*
```

Text

```
*TYPE "THIS IS A LINE OF TEXT",!
THIS IS A LINE OF TEXT
*
```

Variable tables

```
*TYPE $
S M=   30.0000
```

```
S G(27)=    3.5000
S N=   25.0000
S A(3)=   14.0000
S &=    0.0000 *
```

Combination of Variable and Text

```
*SET N=25
*TYPE "N IS",N
N IS=   25.0000*
```

3.1.2 ASK Command

The ASK command is normally used in stored statements to enable the user to input numerical data during the execution of the program.

The ASK command is used with a single variable or, more commonly, with a text string and a variable. When an ASK command is executed, FOCAL-11 prints a colon (:) or a text string and a colon, and the user types the value to be input. Each input value is followed by a terminator character. Terminators are SPACE, COMMA, ALTMODE, LINE-FEED and RETURN keys.

The value is assigned to the variable when the terminator is typed. At any time before the terminator is typed, the value can be changed by typing a CNTRL/U immediately after the value and then retyping the correct value and a terminator. The RUBOUT key can be used to delete one character at a time.

More than one value can be input with a single ASK command, and a colon (:) is printed for each variable specified.

Carriage return and line spacing in an ASK command with text are controlled the same as with the TYPE command (see Data Formats, paragraph 6.1).

The ALTMODE key is a special non-spacing terminator which enables the user to leave a previously assigned value unchanged.

The LINE-FEED key is used to perform the same function as the ALTMODE key. In addition this character also performs a carriage return to start a new line.

Alphabetic characters may be entered in response to an ASK command. See section 2.4 for details.

Input to the ASK command may be an expression instead of a value following the colon (:). This expression will be evaluated using the current variables in the symbol table. If an expression is used, a leading plus or minus sign is required.

The form of the ASK command is:

```
ASK variable

    or

ASK "text"  variable
```

For example;

```
          *ASK "HOW MANY ENTRIES?",E     When this line is executed,
          HOW MANY ENTRIES?:5            FOCAL prints HOW MANY
                                         ENTRIES?: and the user types
                                         a value and a terminator.
```

Using more than one variable in a single ASK command:

```
          *ASK "HI LIMIT",HI,"LO LIMIT",LO
          HI LIMIT:125
          LO LIMIT:50
```

The values 125 and 50 are assigned to the variables HI and LO respectively.

Using the ALTMODE

```
          *ASK "HI LIMIT",HI,"LO LIMIT",LO
          HI LIMIT:  LO LIMIT:0     User typed ALTMODE to keep
                                    the old value.
          *TYPE HI,! LO
          =   125.0000
          =   0
```

Inputting expressions:

```
          *1.1 SET A=1
          *1.2 SET B=2
          *1.3 SET X=3
          *1.4 ASK Y
          *1.5 TYPE Y
          *GO
          :+A+X↑2+B
          =   12.0000*
```

3.1.3 OPERATE Command

The OPERATE command is used to select the input and/or output device for the TYPE and ASK commands. FOCAL-11 selects the terminal keyboard and printer by default unless the user specifies, by the OPERATE command, another I/O device or devices. Selectable I/O devices are:

P for the high speed paper tape punch

R for the high speed paper tape reader

T for the terminal printer

K for the terminal keyboard

L for the line printer

The command OPERATE RP selects both the high-speed paper tape reader and punch. OPERATE TK selects both the low speed terminal keyboard and printer.

The following example program prints ABC on the line printer and then on the console terminal.

```
*1.10 OPERATE L
*1.20 TYPE "ABC"!
*1.30 OPERATE T
*1.40 TYPE "ABC"
```

3.1.4 KILL Command

The KILL command is used to reinitialize all peripheral devices such as clocks and A/D converters by doing a RESET instruction. Error code ?09 is issued when the KILL command is executed. (CTRL/C halts the program but does not halt all I/O devices.)

NOTE

> RESET is a PDP-11 machine language instruction, not a FOCAL-11 Command. In the RT-11 version of FOCAL (using the single job monitor), the KILL command performs a ".HRESET" monitor call. This will perform a similar function to the PDP-11 RESET instruction, but allows the RT-11 monitor to continue to function normally. The Foreground/Background version of RT-11 FOCAL performs the same function as if a ↑C and REenter sequence had been performed.
>
> Although the ↑C does not reinitialize all peripheral devices in RT-11, it does purge all library channels which were open at that time.

3.2 COMPUTATIONAL COMMAND (SET)

The SET statement is used to set a variable equal to the result of an expression. All of the usual arithmetic operations, including exponentiation, may be used.

The SET command consists of the command SET followed by a single variable. This is then followed by an equal sign (=) and an expression to be evaluated.

For example:

```
*SET B=2
*SET D=4
*SET A=B*(6+8/D)
*TYPE A
=   16.0000*
```

In the RT-11 version, it is possible to display the results of all SET commands. For details, see section 5.1.2 of this manual.

3.3 CONTROL COMMANDS

3.3.1 GO Command

The GO command is used to instruct FOCAL to continue executing commands from memory.

This command may have a line number, a group number, or no argument associated with it.

The GO command, without an argument, is a command which causes FOCAL-11 to transfer control to the lowest numbered line in the program and begin execution.

The GO command, with a line number as an argument, causes FOCAL-11 to transfer control to a specific line in the program and begin execution of the commands in ascending line number order.

FOCAL-11 also allows a group number to be given in place of a line number. In this instance, program control would transfer to the first FOCAL statement in the group.

In the above cases where an argument is used, this may be either a constant, a variable (not beginning with the letter A), or an expression.

Example of a GO used with no argument:

```
*1.1 SET A=1
*1.3 SET B=2
*1.5 TYPE A,B
*GO
=   1.0000=   2.0000*
```

In the above example the GO command caused execution to begin at line 1.1.

Transfer to a specific line:

```
*SET A=0
*SET B=0
*1.1 SET A=1
*1.3 SET B=2
*1.5 TYPE A,B;QUIT
*GO 1.3
=   0.0000=   2.0000*
```

In the above example, A and B are set equal to zero before the start of the program. Since the GO causes program execution to begin at line 1.3, line 1.1 is never executed and A is not set to 1.

Example of group transfers:

```
*SET A=0
*SET B=0
*1.1 SET A=1
*1.2 SET B=2
*1.5 TYPE A,B;QUIT
*GO 1
*=   1.0000=  2.0000*
```

3.3.2 IF Command

The IF command lets the program make decisions to transfer program control after a comparison.

The normal IF command format is:

```
        IF (expression) line1,line2,line3
                             or
        IF (expression) group1,group2,group3
```

The expression or variable is evaluated, and program control is transferred to "line1" if the value of the expression is less than zero, "line2" if the value is zero, or to "line3" if the value is greater than zero. In addition FOCAL-11 allows the user to specify a group number in place of any line number in the IF command. This will direct program control to transfer to the first statement in the specified group.

The IF command format can be altered to transfer program control to one of two lines: i.e., if a semicolon or a carriage return is immediately after the first line number, control goes to the first line number if the value of the expression is less than zero. If the value is greater than or equal to zero, control goes to the next sequential command.

If a semicolon or a carriage return follows the second line number, control goes to the first or second line number, depending upon whether the value of the expression is less than or equal to zero. If the value is greater than zero, control goes to the next sequential command.

The program below transfers control to line number 2.1, 2.3 or 2.5 according to the value of the expression in the IF command.

```
*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*
```

Example of an IF with less than 3 line numbers:

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

When line 2.20 is executed, program control goes to line 1.8 if X is less than zero. If X is greater than or equal to zero, Q is typed.

Another example using less than three line numbers:

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

If B is less than zero, control goes to line 1.8; if B equals zero, control goes to 1.9; and if it is greater than zero, control goes to the next sequential command, in this case line 3.20, and the value of B is typed.

CAUTION

When using non-integer arithmetic in the IF command, a test for zero may not always be appropriate due to the nature of the floating-point arithmetic used by the computer. To avoid this problem, the programmer should either avoid using non-integer arithmetic in the IF command, or test for fractional values less than the tolerance desired and set the value to 0.

IF statements that test the running variable in FOR loops which use non-integer increments, are particularly sensitive to this problem. For example:

```
1.10 FOR A=-5,.1,5;DO 2
1.20 QUIT

2.10 IF (A)2.20,2.30,2.40
2.20 RETURN
2.30 TYPE "EQUAL TO ZERO";QUIT
2.40 RETURN
```

The above program will never go to statement 2.30 because the value 0.1 can not be represented exactly by FOCAL. This causes the value of A to be slightly less than zero after fifty iterations. If statement 2.10 were written as:

```
2.10 IF (FABS(A)-0.01)2.30,2.20
```

then statement 2.30 would be executed after fifty iterations of the FOR in line 1.10. This is because A was now being tested for the case where its magnitude was less than 0.01. Notice that the third number is not present. This is to further demonstrate that if a number is not present, the IF command will transfer to the next sequential line (2.20) in the user's program. In this case, only the first number (2.30) was needed, since if the expression is zero, and the line number is not given, then statement 2.20 would be executed.

3.3.3 DO command

At times, it is beneficial to have a section of a program perform a general function which is used in several other portions of a program. For this reason, the DO command is provided in FOCAL. This command causes transfer to a specified point in a user's program, and then automatic return to the point in the program from which it was called. The DO command is used to cause execution of single lines or groups of lines. Control is returned to the command following the DO command after the subroutine is executed.

If the user types a command such as DO 3, the DO command treats the group of program lines beginning with 3 as a subroutine. Control proceeds in ascending order through the group numbers until the end of the group is reached.

The DO ALL command executes the entire program as a subroutine.

DO commands may be nested, i.e., while executing a DO command, another DO command may be encountered and executed. The number of nested DO commands is limited only by the amount of memory available for preserving information as to where to return.

If this memory space is exhausted, and the user attempts to use more than is available, the user will get an error. If the RT-11 monitor is used, a ?M-TRAP TO 4? message will be given. A RE (REenter) command may be given to restart FOCAL. Paper-tape users will be notified by a ?09 error, followed by an asterisk (*). This amount of memory space may be altered by using the /B switch when linking FOCAL under RT-11 (see Appendix I).

In most instances, the user need not be concerned by this, as sufficient memory has been reserved for a moderate amount of nesting. In order to ensure that this problem is not encountered, nesting of DO commands should be limited to three levels.

The following is an example of the DO command:

```
*1.1 SET A=1;  SET B=2
*1.2 TYPE "STARTING"
*1.3 DO 3.3;  TYPE "FINISHED"
*3.1 SET A=3;  SET B=4
*3.3 TYPE A+B
*GO
STARTING=   3.0000FINISHED=   7.0000*
```

The following is an example of nested DO's:

```
*1.1 TYPE "BEGIN",!
*1.2 DO 2
*1.3 TYPE "END",!;QUIT
*2.1 DO 5.1; TYPE A,!
*2.2 DO 5.2; TYPE A,!
*2.3 DO 7.5; TYPE A,!
*5.1 SET A=1
*5.2 SET A=2
*7.5 SET A=3
*GO
BEGIN
=   1.0000
=   2.0000
=   3.0000
END
*
```

3.3.4 RETURN Command

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

```
3.10 ASK Y; DO 10
3.11 TYPE X; QUIT

10.10 IF (Y)10.20,10.30,10.40
10.20 SET X=-1
10.25 RETURN
10.30 SET X=0
10.35 RETURN
10.40 SET X=1
10.45 RETURN
```

3.3.5 QUIT Command

The QUIT command is used to stop the execution of a program under program control. FOCAL-11 then returns to command mode and prints an asterisk. The QUIT command does not affect the current operation of the I/O devices.

3.3.6 FOR Command

The FOR command is used to set up program iterations. The general command format is:

```
*FOR A=B,C,D;commands
*FOR A=B,D;commands
```

The variable A is initialized to the value B, then the command or commands on that line following the semicolon are executed. When all the commands on the line have been executed, the value of A is incremented by C and compared to the value of D. If A has not been incremented past the limit (D) the commands after the semicolon are executed again. This process is repeated until A is incremented past D, at which time FOCAL-11 goes to the next sequential statement. The command or commands will always be executed at least once.

A must be a single variable. B,C, and D may be expressions, variables or numbers. If the value C is omitted, it is assumed that the increment is one. If C and D are omitted, the FOR command is handled like a SET command (i.e., A is set to the value of B) and the program will continue following the semicolon. The values of B,C, and D may be positive or negative.

The DO command may be used in conjunction with a FOR command in order to access subroutines during the iterative process of the FOR command.

```
*ERASE ALL
*1.1 FOR X=1,1,5;DO 2
*1.2 QUIT
*
*2.1 TYPE !"    X",X
*2.2 SET A=X+100
*2.3 TYPE !"    A",A
*GO
    X=     1.0000
    A=   101.0000
    X=     2.0000
    A=   102.0000
```

```
X=    3.0000
A=  103.0000
X=    4.0000
A=  104.0000
X=    5.0000
A=  105.0000*
```

It is often useful to have one or more loops within a loop as in the following example.

```
    *1.10 FOR Z=1,3; TYPE "   A  B  C "
    *1.20 TYPE !
    *1.50 FOR A=1,3; DO 3
    *1.70 QUIT

    *3.10 FOR B=1,3; DO 4

    *4.10 FOR C=1,3; TYPE %1,A,B,C," "
    *4.30 TYPE !
  *
  *GO
    A  B  C   A  B  C   A  B  C
  = 1= 1= 1 = 1= 1= 2 = 1= 1= 3
  = 1= 2= 1 = 1= 2= 2 = 1= 2= 3
  = 1= 3= 1 = 1= 3= 2 = 1= 3= 3
  = 2= 1= 1 = 2= 1= 2 = 2= 1= 3
  = 2= 2= 1 = 2= 2= 2 = 2= 2= 3
  = 2= 3= 1 = 2= 3= 2 = 2= 3= 3
  = 3= 1= 1 = 3= 1= 2 = 3= 1= 3
  = 3= 2= 1 = 3= 2= 2 = 3= 2= 3
  = 3= 3= 1 = 3= 3= 2 = 3= 3= 3
  *
```

Another way of writing the same program is:

```
*1.1  FOR Z=1,3;  TYPE " A B C "
*1.2  FOR A=1,3;  FOR B=1,3;  TYPE !;  FOR C=1,3;TYPE  %1,A,B,C,"
"
*GO
```

3.3.7 COMMENT Command

The COMMENT command (abbreviated as C) causes the program line to be ignored by FOCAL-11. The user may use the C command to insert explanatory comments into the program. Program lines beginning with C are skipped when the program is executed. However, comments are printed in response to a WRITE command. (Refer to paragraph 3.4.1.)

```
*ERASE ALL
*1.1 COMMENT INITIALIZE VARIABLES
*1.2 SET A=5
*1.3 SET B=6
*1.4 SET C=7
*2.1 COMMENT PERFORM CALCULATION
*2.2 TYPE A+B+C
*GO
=   18.0000*
```

NOTE

A comment command may be terminated only by a carriage return.

```
*ERASE ALL
*1.05 SET A=1;SET B=2;SET C=0
*1.10 COMMENT DEFINE C=A+B;SET C=A+B
*1.20 TYPE C
*GO
=    0.0000*
```

In this case, the SET command following the comment will never be seen by FOCAL.

3.3.8 XECUTE Command

Some routines or functions do not return a value (i.e., they perform an operation rather than a calculation). To call such a routine requires a dummy SET command (i.e., SET X=FCHR(48)) or the more efficient XECUTE command. The XECUTE command performs a subroutine call for these functions (such as FCHR or FSBR). Any expression may follow the XECUTE command.

*XECUTE FCHR(48)	Outputs the character 0. (48(10) = 7 bit ASCII code for 0.)
*XECUTE FSBR(10,5)	Execute the lines in group 10 with variable & (ampersand) equal to 5.

The FOCAL-11 functions, including the special variable &, are described in Chapter 4.

3.4 EDIT COMMANDS

3.4.1 WRITE or WRITE ALL Command

The WRITE or WRITE ALL command causes FOCAL-11 to print a program line, a group of lines, or an entire program on the console terminal. Using the WRITE ALL command after an editing session is a practical method of producing a clean listing of the program.

*WRITE 2.1	Print a line
*WRITE 2	Print a group
*WRITE	Print entire program
*WRITE ALL	Print entire program

The write command can be used to create paper-tapes on-line. Once his program has completed and in command mode, the user may save it by putting it on paper tape. It is also possible to save and restore programs using RT-11 files. This is described in section 3.5 of this manual. The procedure for saving a FOCAL-11 program on-line is as follows:

1. Make sure FOCAL-11 is in command mode(*).
2. Type WRITE.
3. Set low-speed punch to ON position.
4. Type RETURN key.

FOCAL-11 will punch the entire program onto the low-speed paper tape punch and simultaneously print it on the terminal. Once the program has been punched, the paper tape punch should be immediately turned off.

To load and run a program previously saved on paper tape:

1. Make sure FOCAL-11 is in command mode(*).
2. Put the program tape in the low-speed reader.
3. Switch the low-speed reader to START.

The program will be put into memory the same as if the user were typing it on the terminal keyboard. When the entire program has been read into memory, the user should type CTRL/C since the asterisk printed when the WRITE command is finished is also punched and may be interpreted as a command.

To load programs into the computer via the high-speed paper-tape reader:

1. Make sure FOCAL-11 is in command mode(*).
2. Put the program tape in the high-speed reader.
3. Type O R (or OPERATE R) and the RETURN key.
4. Either a. Paper-Tape FOCAL types CTRL/C when tape reading is completed or,
   b. RT-11 FOCAL automatically outputs ?00 AT 00.00 error message.

To save programs on paper tape using the high-speed paper-tape punch:

1. Make sure FOCAL-11 is in command mode(*).
2. Type O P;W A;O T and the RETURN key.

3.4.2 ERASE Commands

The ERASE command erases a single line or a group of lines.

ERASE used alone has the function of merely removing the variables without affecting the program text. The special character "&" (ampersand) is not cleared by this command in RT-11 FOCAL. This is so that a value can be passed from one program to another, while freeing all other variable area. The Paper-tape version of FOCAL does clear this variable with the ERASE command. This may also be thought of as initializing the values of the variables to zero. Modifications to the text using the MODIFY command may or may not cause the variables to be deleted as if an ERASE command was given. It is suggested that lines be retyped instead of using the MODIFY command if it is required that variables be saved. It is good practice to use an ERASE at the beginning of a program.

The ERASE TEXT command leaves the variables intact but removes all program text in preparation for using another program of the same or smaller size with the same variables. This command allows the same data to be used for many programs.

The ERASE ALL command deletes the entire program. It is good programming practice to type ERASE ALL before starting to enter a new program.

The ERASE ALL and ERASE TEXT commands should never be used as part of a FOCAL program. The ERASE<line or group number> may be used inside of a program under the following conditions.

1. It is not part of a FOR command.
2. It is not used as part of a DO.
3. It is the last command on that line.

Examples of the ERASE command:

*ERASE 2	Deletes all lines that begin with 2.
*ERASE 2.2	Deletes line 2.2
*ERASE	Initialize variables to zero.
*ERASE TEXT	Delete text, leave variables intact.
*ERASE ALL	Delete both text and variables.

3.4.3 MODIFY Command

The MODIFY command is used to change, insert, or delete characters within a line without the need to retype the entire line. The format for MODIFY is:

```
MODIFY line number RETURN key
Search character
```

The search character is not printed. After the user has typed the line number, RETURN key, and search character (which is not echoed). FOCAL-11 prints the contents of the specified line up to and including the search character. When printing stops, the user has the following options:

1. Type new characters in addition to those already printed. The new characters are inserted at that point.

2. Type a form feed (CTRL/L). This causes the search to proceed to the next occurrence, if any, of the search character.

3. Type CTRL/G(bell). The user can then change the search character he specified in the MODIFY command by typing another search character (again not echoed).

4. Type the RUBOUT key. This causes FOCAL-11 to delete a character, starting with the last character printed and moving one character to the left each time RUBOUT is typed. FOCAL-11 echoes each character deleted. Groups of deleted character are separated from the rest of the line by enclosure in back-slashes (\).

5. Type CTRL/U or back arrow (←). This causes FOCAL-11 to delete everything between the current character and the line number.

6. Type the RETURN key. This causes FOCAL-11 to terminate the line at that point, deleting everything to the right.

7. Type the LINE FEED key. This is normally done after the user has exercised one or more of the above options. After the user has modified the line, he may type LINE FEED and cause the remainder of the line from the search character to the end to be printed and saved.

```
*7.01 FOR C=2,3;  TYPD %1,A,B,C," "
*MODIFY 7.01
FOR C=2\2\1,3;  TYPD\D\E %1,A,B,C," "
```

In the above example, 2 was typed as the search character for line 7.01 (Note that the search character did not print.) FOCAL-11 stopped typing when it encountered the search character (2), and the user typed the RUBOUT key to delete the 2. Then he typed the correct character 1. Next he typed CTRL/G(bell) and the D key to change the search character. FOCAL-11 continued to print the line until the new search character was encountered. The user typed RUBOUT to delete the D and then typed the correct character E. He then typed the LINE FEED key and the remainder of the line was printed.

```
*WRITE 7.01
 7.01 FOR C=1,3;  TYPE %1,A,B,C," "
```

The WRITE command can be used to display the corrected line.

The MODIFY command cannot be used to change the line number itself, or to divide the line into two lines. Once the MODIFY command has been started, the best way to undo a mistake (for instance an unintentional CTRL/U) is to type a CTRL/C. If the RT-11 version of FOCAL is used, a REENTER command is also required. FOCAL will print ?00 AT 00.00 and return with an asterisk.

3.4.4 Replacing Lines

It is sometimes easier to replace a line in a program rather than attempt to make a modification to it. To replace a line, retype the line number followed by the new command or commands. The old line is automatically erased.

3.5 LIBRARY COMMAND

THE LIBRARY command, provided in the RT-11 version of FOCAL, allows the user to access any RT-11 file structured device.

A file is a collection of information which can be stored on a mass-storage device such as a disk or a DECtape, and assigned a name which can be later used to reference this information. A file structured device is any device which is capable of storing one or more files.

One of the features of the RT-11 operating system is to maintain files in a manner which allows them to be easily accessed by the user. The RT-11 version of FOCAL makes use of this feature through the FOCAL LIBRARY commands.

The user is allowed to store information (programs and data) on a RT-11 file-structured device and assign to it a unique name which can be used to reference the material.

This name (which will be refered to in the examples as "file-name") is composed of two separate parts. The first part of the file name consists of up to six (6) characters. These characters may be either letters, numbers, or both. No other characters may be used in a file name. The second portion of the file name is optional. This portion is called the file extension because it qualifies the first part of the file name. This section may be composed of up to three characters having the same restriction on special characters as the first part. If this part of the file name is specified, a period (.) must be placed between the first and second parts. If this section is not specified, FOCAL will automatically supply ".FCL" as an extension.

Some examples of legal file names appear below:

```
FILE.EXT
PROG01
DATA.001
123456.789
```

This section should provide enough material for the user to effectivly utilize the LIBRARY command. Due to the versitility of this command, Chapter 7 has been devoted to providing the user with a full explaination the features available.

The material presented in this section pertains to all RT-11 versions of FOCAL-11.

3.5.1 Saving a Program Under RT-11

In order to save a program on an RT-11 file for later use, the LIBRARY SAVE command is used.

After a program has been entered into memory, a command of the form:

```
LIBRARY SAVE file-name
```

is given to FOCAL. This command will cause FOCAL to save the user's program in the specified file. If an older version of this file existed, it would be deleted when this version of the program was saved.

3.5.2 Running a Program Under RT-11

Once a program has been saved, it can later be recalled from the RT-11 file and started by giving the command:

```
LIBRARY RUN file-name
```

This will cause the specified program to be brought into memory, the variables erased, and the program started as if a GO command was issued.

3.5.3 Modifying a Program Under RT-11

In order to modify an existing program using FOCAL under RT-11, three distinct steps should be taken. These are:

1. Retrieve the program into memory.
2. Modify the program as described in Section 3.4
3. Save the modified program.

Both steps 2 and 3 have been discussed previously. There remains only the method of loading a program into memory without starting it. This is done by using the LIBRARY GET command.

The function of this command is to input a file under RT-11 just as if the user had entered the contents of the file on the terminal. This means that if a user already had a program in memory at the time that the LIBRARY GET command was issued, the new program which was being entered would combine with the program already in memory, replacing lines which have the same line number, and inserting lines where no line had existed before. This is useful if the user wishes to include a general subroutine into a program.

The LIBRARY GET command should be combined with an ERASE ALL command when using this command to load a program for modification. This will ensure that only the desired program will be in memory at that time.

For example:

```
*ERASE ALL           (The user clears memory)
*LIBRARY GET file-name (The program is loaded)
*
```

3.5.4 Virtual Files

In addition to saving and restoring programs under RT-11, it is possible to use RT-11 files for storing data in the form of arrays similar to normal arrays. Files used in this way are called VIRTUAL FILES since it looks to the programmer as if the variables reside in memory instead of on the file.

Preparing a Virtual File for use

In order to access a file on an RT-11 file structured device, the LIBRARY OPEN command is used. The form of this command is:

```
LIBRARY OPEN file#,file-name[size]/Z/V:variable
```

All files must have a number associated with them so that the user is able to refer to them within his program. The file# can range from zero (0) to seven (7).

If no file by the specified name is found (i.e. the first time a file is used), the LIBRARY OPEN command will create one according to the parameters specified in the command.

If a file by the specified name already exists, then the LIBRARY OPEN command will use the data placed in it.

If it is desired to either alway create a new file, or to always use an existing file, the LIBRARY MAKE and LIBRARY INPUT commands may be used. These commands have the same format as the LIBRARY OPEN command.

For example:

```
LIBRARY MAKE file#,file-name[size]/Z/V:variable
```

or

```
LIBRARY INPUT file#,file-name/V:variable
```

The LIBRARY INPUT command does not need all of the information specified for the LIBRARY MAKE or LIBRARY OPEN commands. This is because LIBRARY INPUT command may only use existing files, and can not create them. Therefore, the information which is only used for creating files is not needed, and may be eliminated as in the above example.

The "[size]" parameter informs FOCAL as to the number of data blocks to be reserved for the file if it is to be created. The value which is placed inside of the brackets ([]) must be a numerical constant. Variables and expressions are not allowed. There is a means of altering this value under program control. This is described in Section 7.2.4. In general, the user should make this value equal to the result of following expression:

(maximum subscript)/64+1   (truncate result)

or for doubly subscripted variables:

(maximum second subscript)*4+1

More information concerning the "[size]" parameter, and its relation to Virtual Files can be found in Section 7.4.

The /Z indicates that the array should be initialized to zero if it is created. If a file already exists, the data stored in it will not be destroyed.

Accessing the data in a Virtual File

The variable in the LIBRARY OPEN, MAKE, or INPUT commands specified after the /V: should be the name by which the data is to be referenced in the user's program. For example, if "/V:X(0)" was specified in the LIBRARY OPEN command, then whenever the program referenced the variable "X(n)", the information will be stored on the RT-11 file rather than in memory. There is no restriction on the use of the variable.

Subscripts used in a Virtual File should be positive integer values within the range of 0 to 32,767 for singly subscripted variables, and from 0 to 127 for doubly subscripted variables. It is possible to use negative subscripts, but due to the extra considerations required for their use, information can be found in section 7.4.

Virtual Files created by the single and double precision versions of FOCAL are not compatable. Once data has been placed in a Virtual File, only the version of FOCAL which created it should access the data. Provisions are available to use Virtual Files in a manner which can be compatable with other versions of FOCAL. This is described in Section 7.2.5.

One fact which the user should be aware of is that the variables are stored on a device much slower than memory. In order to prevent the necessity of accessing the mass storage device each time that data is requested by the user, FOCAL reserves a small amount of memory for each Virtual File in use. When a variable is needed by the program, FOCAL checks to see if the data is contained in the region of memory reserved for that file. If it is, the data is immediately made available to the program. If the requested data is not in memory, FOCAL must first check to see if the user program has altered the contents of any of the data currently in memory. If any data had been altered, this data is rewritten to the device. Then FOCAL reads the data requested from the mass storage device. When this data is read, several adjacent (subscript-wise) variables are also read from the device into memory. If a program then references one of these adjacent variables, FOCAL will be able to return to the user with the results immediately.

Closing a Virtual File

When processing of a virtual file is completed, FOCAL must be informed of this fact so that it is able to ensure that all of the data is recorded properly in the RT-11 file. This is done by using the LIBRARY CLOSE command. There are two major forms of this command. These are:

```
LIBRARY CLOSE file#
```

and

```
LIBRARY CLOSE
```

The first form of this command is used to terminate activity on a specific file specified by the file#. The other form is used to close all open files. It is usually a safe practice to perform the latter form of LIBRARY CLOSE command before issuing a QUIT command in a user's program. Files will not be closed by the QUIT command itself. This is so that data files can be left open for several programs to use.

Example

For example, suppose a user wished to use a virtual file. The following program will open a file, set the elements equal to the value of the subscript squared.

```
1.10 LIBRARY MAKE 0,DATA[2]/Z/V:X(0)
1.20 FOR I=0,100;SET X(I)=I↑2
1.30 LIBRARY CLOSE 0
```

The data is now placed in the file: "DATA.FCL". Since 100 was the highest subscript value, the file size was calculated to be 100/64+1=2 (when truncated).

A second program can now be written to use the file "DATA.FCL". This program will use the old data and output the sum of the squares:

```
1.10 LIBRARY INPUT 3,DATA/V:Y(0)
1.20 SET SUM=0
1.30 FOR I=0,100; SET SUM=SUM+Y(I)
1.40 LIBRARY CLOSE 3
1.50 TYPE !"THE ANSWER ",SUM,!;QUIT
```

It should be noted that data can be read from a file by using a different variable from which it was created. This was illustrated by the above example, since the data was created using X(n) and was later used by the variable Y(n).

Statement 1.10 in both programs could have been replaced by a LIBRARY OPEN command. The form of this command would have been:

```
     1.10 LIBRARY OPEN (0 or 3),DATA[2]/Z/V:(X(0) or Y(0))
```

CHAPTER 4

FOCAL-11 FUNCTIONS

The FOCAL-11 functions, which are subprograms internal to FOCAL-11, improve and simplify arithmetic capabilities and give the potential for expansion to additional input/output devices.

In general, the FOCAL-11 functions may be used anywhere a number or a variable is legal in a mathematical expression. A standard function call consists of two or more letters beginning with the letter F and followed by an argument expression in parenthesis.

Fxxx(expression)

The following standard functions are available:

FSIN(R)	Sine function (radians)
FCOS(R)	Cosine function (radians)
FEXP(arg)*	Exponential function
FLOG(arg)*	Logarithm to the base 10.
FLN(arg)*	Natural logarithm
FX(func,addr,data)	Access to UNIBUS
FCHR(arg)	Print and accept ASCII codes
FRAN( )	Random number function
FADC(channel)	Analog to digital converter function
FCLK( )	Clock function
FABS(arg)	Absolute value function
FSGN(arg)	Sign function
FITR(arg)	Integer part function
FSQT(arg)	Square root function
FSBR(group,arg)	User programmed function
FPRM(parameter,value)	Alter FOCAL internal parameters
FERR(group/line)*	Define error handling routine
FINT(vector,group,pri,CSR addr,mask)*	Establish a routine to be executed on the detection of a specific hardware interrupt.
FQUE(count,group,interval,delay,priority)*	Schedule a group or line number to be run "count" times, once every "interval" seconds, starting "delay" seconds from now. The routine will have a priority of "priority".

The above functions may be used where any legal FOCAL-11 expression is allowed. For example:

```
*SET Z=A+FSQT(FSIN(X))
*XECUTE FCHR(48)
```

NOTE

The starred (*) functions are not available in either the 4K Paper-tape or the 8K RT-11 versions of FOCAL-11. Some of these functions can be obtained by using the FSBR function as indicated in Appendix D.

The FERR, FINT, and the FQUE functions are discussed in Chapter 6.

4.1 TRIGONOMETRIC FUNCTIONS

4.1.1 Sine Function (FSIN)

The sine function (FSIN) is used to calculate the sine of a user-specified angle in radians. The format for FSIN is:

FSIN(angle)

For example:

```
*TYPE FSIN(3.14159/4)
= 0.7071*
```

The format for calculating the sine of an angle in degrees is:

FSIN(degrees*3.14159/180)

For example:

```
*TYPE FSIN(30*3.14159/180)
= 0.5000*
```

4.1.2 Cosine Function (FCOS)

The cosine function (FCOS) is used to calculate the cosine of a user specified angle in radians. The format for FCOS is:

FCOS(angle)

For example:

```
*TYPE FCOS(2*3.14159)
=      1.0000*
*TYPE FCOS(.5000)
=    0.8776*
*TYPE FCOS(45*3.14159/180);  C COS OF 45 DEG.
=    0.7071*
```

4.2 LOGARITHM FUNCTIONS (EXTENDED FOCAL ONLY)

4.2.1 Exponential Function (FEXP)

The exponential function is used to calculate powers of "e". For example:

```
                    SET X=FEXP(-A*T)
```

will set the value of X to

$$e^{-A*T}$$

This function is the inverse of the natural logarithm funtion (FLN).

```
     i.e.
                              FLN(FEXP(X))
     is equal to
                              FEXP(FLN(X))
     which is equal to
                                   X
```

4.2.2 Natural Logarithm Function (FLN)

The natural logarithm function is used to find the power to which "e" must be raised in order to be equal to the specified argument. This function is complementary to the exponential function (FEXP). See the section above on FEXP for an example of this.

4.2.3 Logarithm Base Ten (FLOG)

This function is used to determine the power to which ten must be raised in order to be equal to the specified argument. For example:

```
                              10↑FLOG(X)
     is equal to:
                              FLOG(10↑X)
     which is equal to:
                                   X
```

4.3 UNIBUS FUNCTION (FX)

The UNIBUS control function FX is used to control additional device options, nonstandard peripherals and references to core storage. The first argument "func" can have a value that is negative (-2,-1), zero (0), or positive (+1,+2) to select the function that is to be performed. The functions are respectively read (+1,+2), logical "AND" (0), and load (-1,-2) onto the UNIBUS. The reason that two functions are supplied for the read and load operations, is that one is for byte transfers (+1,-1), and the other is for word operations (+2,-2). The second argument "UNIBUS-address" must be either an octal number (16 bits maximum) or a variable name (maximum integer value of 15 bits). The function selected will be performed on the UNIBUS address specified with the data given, if any, in the third argument ("data"). The format for FX is:

```
     FX(func, UNIBUS-address, data)

          XECUTE FX(+2,@177570)          One word is taken from the
                                         UNIBUS from address 177570,
```

	and is returned as a signed integer value. If the address does not exist, or is odd, error ?09 will result.
XECUTE FX(+1,@177570)	Data is taken through the UNIBUS from address 177570 (where 177570 is an octal byte address) and the value of the function is the UNIBUS reading.
XECUTE FX(-2,@1630,X)	One word is put into the UNIBUS to octal address 1630, where x is the data stored. If the address does not exist, or is odd, error ?09 will result.
XECUTE FX(-1,@35773,x)	Data is put into the UNIBUS to octal byte location 35773 (octal), where X is the data. If the address does not exist, error ?09 will be given.
XECUTE FX(0,@43210,ALPHA)	Bit information is selectively taken from the UNIBUS by "ANDing" the contents of UNIBUS address 43210 with the value of the bits of ALPHA. The value of the function call is the result of the intersection of the argument and the data found in the UNIBUS address. The address used must be even numbered, or an error ?09 will result.

NOTE

The "@" used as a prefix in the above examples is used to inform FOCAL that the digits immediately following are octal. See section 2.1 for further information.

4.4 CHARACTER I/O FUNCTION (FCHR)

The FCHR function is used to accept and/or print ASCII codes. Its principal use is to convert characters from ASCII to decimal or from decimal to ASCII. The function manipulates the value of a single character from the currently selected input or output device (see the OPERATE command for how to select devices). If the argument is negative then the function will read the next available eight bit character from the input device. If the argument is zero or positive then that (decimal) value will be converted to an eight bit integer and transmitted to the currently selected output device. (The value of the function is the integer value of the argument.) Multiple arguments and literals are accepted. This function makes it possible to print control characters to the line printer when selected.

The format for FCHR is:

     FCHR(args)

```
        *COMMENT 77.=115(8)=ASCII"M"=0M+64
        *XECUTE FCHR(77,@12,0B+64,-1)
        M

         B*
```

This command prints three 8-bit ASCII characters: "M", <line-feed>, and "B". The function then inputs one character from the input device. This last character becomes the (decimal) value of the function. It is not echoed.

The function may also be used recursively.

```
        *SET Z=FCHR(FCHR(-1))
```

This command accepts a character from the input device, outputs the same 8-bits, and leaves the value in the variable Z.

The FCHR function is useful in analyzing a character or string of characters. For example, FCHR can be used to check the answer given in a teaching program which uses questions and answers. In a multiple choice quiz, the answer to the following question

```
        THE PAPER-TAPE VERSION FOCAL-11 RESTART ADDRESS IS :
        a) 111111 b) 000000 c) 000111
        YOUR ANSWER? B
```

could be analyzed with the following code:

```
        1.45 SET RE=FCHR(-1)
        1.50 IF (RE-65)3.10,4.10,3.10

        3.10 TYPE "SORRY, THAT'S INCORRECT",!;G 1.45

        4.10 TYPE "THAT'S CORRECT",!
```

4.5 RANDOM FUNCTION (FRAN)

The random number function (FRAN) is used to generate a value between -1 and 1. This is formed by generating a sixteen bit random integer within the range of -32,768 and +32,767. This value is then scaled down to the normalized range of -1 to +1. The format for FRAN is:

```
        FRAN()

        *TYPE FRAN()
        =- 0.3916*
        *TYPE FRAN()
        = 0.1659*
```

FRAN(1) restarts the sequence

```
        *XECUTE FRAN(1)
```

FRAN(1) executed once at the beginning of a program is useful in debugging that program because the same sequence of random values is produced each time the program is restarted. When the RT-11 version is first loaded, the lowest portion of the time of day is used for the first random number. This insures randomness for the first program run.

4.6 ANALOG TO DIGITAL CONVERTER FUNCTION (FADC)

The analog to digital converter function FADC allows easily programmed access to as many as 16 A/D channels on the AR11. The argument is the channel number. The function returns a value in the range 0 to 4095, which corresponds to the digital value returned by the device.

The format for FADC is:

```
     FADC(channel)

          *SET A=FADC(3)          Channel 3 is read and it's value
                                  stored in the variable A.
```

For other types of A/D converters, use the FX(func, addr, data) function.

4.7 CLOCK FUNCTION (FCLK)

The clock function FCLK has a value of the time elapsed in 60ths (or 50ths) of a second since the clock was started. In RT-11, the system real-time clock (KW11-L) runs at all times. The RT-11 TIME command is used to set the time of day. When the FCLK function is used in RT-11 FOCAL, the elapsed time since midnight (00:00) is returned.

The paper tape version of FOCAL-11 attempts to start the user's clock when it is loaded. This version supports either the KW11-L or the KW11-P clocks. The KW11-L line clock is tried first. If this clock is not available, then the KW11-P programmable clock will be tried.

In the Paper-Tape version the clock can be stopped by the KILL command, the statement FX (-1,@177546,0) for the KW11-L, FX(-1,@172540,0) for the KW11-P, power-fail, or manual-restart, but not by CTRL/C.

The format for FCLK is:

```
          *SET X=FCLK()
```

If an argument is given, this value will be subtracted from the present value of the clock.

```
     *1.1 SET X=FCLK()        Get initial clock value
     *1.2 DO 7                Execute group 7
     *1.3 TYPE FCLK(X)/60     Print the elapsed time
                              in seconds
     =14.5000*                14.5 seconds were spent in group 7
```

4.8 ABSOLUTE VALUE FUNCTION (FABS)

The absolute value function (FABS) is used to obtain the absolute (positive) value of an expression. The format for FABS is:

```
FABS(expression)

     *TYPE FABS(-66)
     = 66.0000*
     *TYPE FABS(+23)
     = 23.0000*
     *TYPE FABS(-99.05)
     = 99.0501*
```

4.9 SIGN FUNCTION (FSGN)

The sign function (FSGN) is used to obtain the sign of a number. If the argument is <0, FSGN returns -1; if -0, FSGN returns 0, and if >0, +1.

```
FSGN(expression)

     *TYPE FSGN(6-4)
     = 1.0000*
     *TYPE FSGN(0)
     = 0.0000*
     *TYPE FSGN(-7)
     =- 1.0000*
```

4.10 INTEGER PART FUNCTION (FITR)

The integer part function (FITR) is used to obtain the integer part of a number. The format for FITR is:

```
FITR(expression)
```

NOTE

The FITR function returns the integer part of a number by truncating the practional portion. It is not equivalent to the greatest integer function, as for negative numbers the truncation causes the result to be larger than the argument. i.e.: FITR(-3.5)=-3 and -3>-3.5

```
*TYPE FITR(5.2)
= 5.0000*
*TYPE FITR(55.66)
= 55.0000*
*TYPE FITR(77.434)
= 77.0000*
*TYPE FITR(-4.1)
=- 4.0000*
```

4.11 SQUARE ROOT FUNCTION (FSQT)

The square root function (FSQT) is used to compute the square root of an expression. The format for FSQT is:

```
     FSQT(expression)

          *TYPE FSQT(4)
          = 2.0000*
          *TYPE FSQT(9)
          = 3.0000*
          *SET Z=FSQT(144);TYPE Z
          = 12.0000*
```

It is illegal to take the square root of a negative number in FOCAL.

4.12 USER PROGRAMMED FUNCTION (FSBR)

The user programmed function FSBR (group,arg) is similar to the DO command in that it is used to call the indicated group of lines (or single line) as a subroutine. The function call transmits the second argument to the variable & (ampersand) and when the subroutine returns, the last value of & (ampersand) is taken as the function return (i.e., the final value of the function FSBR (group,arg)).

```
          *1.2 SET Y=FSBR(5,A+B+C)
```

In a group 5 subroutine, the argument & (ampersand) is automatically set to A+B+C then & is used in the subroutine. The final value of & is returned as the final value of the FSBR function. Notice that the command which includes the FSBR function cannot be in the same group which is called by the FSBR function.

This technique is analogous to the more cumbersome:

```
          *SET &=A+B+C;DO 5;  SET Y=&
```

The FSBR function can also be used to present the same argument to a sequence of functions, and can even be used recursively.

```
          *FOR J=5,1,14;  SET Y(J-5)=FSBR(J,FSQT(BETA))
```

Refer to Appendix D for examples of extended functions which make use of the FSBR function.

4.13 FOCAL PARAMETER FUNCTION (FPRM)

It is often of use to the FOCAL-11 user to be able to alter some of the operations of FOCAL-11 to suit one's needs. In the past, the user was required to make physical patches (i.e. alter locations in FOCAL) to produce the desired changes. The FPRM function now performs many of the more common alterations for the user under program control. There are two forms of the FPRM function

```
                    FPRM(prm)
                       or
                  FPRM(prm,value)
```

If the FPRM function contains only a single argument, the function will return as a value the current value associated with the parameter number specified. The value of this parameter will not be altered. If two arguments are specified, the first will designate the parameter number, and the second will specify a new value for that parameter. The function will return the old value of the parameter. The permissible parameters, and their functions are shown in the table below.

<u>FPRM Parameters</u>

Param #	Default	Function
0	0	Determine the top of FOCAL (RT-11 only) 0 - Full area (USR swapping) 1 - Top is below user area (USR non-swapping) (see RT-11 Programming Manual for information on USR swapping)
1	0	Extended symbol table 0 - Normal FOCAL symbols 1 - "BASIC" type symbols (RT-11) "Sequential" symbol storage (Paper Tape)
2	73	Terminal width for output of TYPE
3	0	Switch for ":" and "=" in TYPE/ASK 0 - Characters are printed 1 - ":" is not printed on ASK 2 - "=" is not printed on TYPE 3 - Neither ":" or "=" are printed on TYPE/ASK
4	@60	FOCAL input expressions from ASK 0 - Input of "YES" is the variable YES @60- Input of "YES" is constant 0YES xx - This character is prefixed to all ASK input
5	0	Extended debug mode (Section 5.1.2) 0 - Normal mode 1 - Line numbers are printed as well as variable values (using current type format) -1 - Same as above but format is %8.04
6	0	Output rounding switch 0 - Rounding is performed on TYPE output 1 - Rounding is not performed on TYPE output
7	x	Holds the current output position (should not be changed by user)
8	0	Switch for leading zeroes on TYPE output conversions (TYPE ↑O, ↑B, and ↑I) (Sections 6.1.3 - 6.1.5) 0 - Only significant digits are output 1 - Leading zeroes are output

9	1	Floating point underflow error disposition 0 - Report error (?12) 1 - Ignore underflow errors
10	0	Scientific notation switch (Section 6.1.2) 0 - Use standard floating point notation 1 - Use scientific notation
11	0	Time scale for FQUE function (Section 6.6) 0 - Time is in units of seconds 1 - Time is in units of "ticks" (either 1/60 or 1/50 of a second depending upon the power line frequency.)

CHAPTER 5

IMPLEMENTATION NOTES

5.1 DEBUGGING

5.1.1 Using the Error Diagnostics

Whenever FOCAL-11 detects an illegal command or improbable condition within a user's program, the execution of the program stops and an error message is printed in the form ?XX AT GG.ss, where ?XX is the error message and GG.ss is the line at which the error occurred. (See FOCAL-11 Error Diagnostics, Appendix B, for the complete list of error messages.)

For example, if the user types CTRL/C (and RE in RT-11) to terminate a loop, the error message ?00 is printed and program control goes to command mode. In this case, the user ignores the message and types his next command.

For example:

```
*ERASE ALL
*1.10 SET A=2;  TYPE "A",A,!
*1.20 SET B=4;  TYPE "B",B,!
*1.30 GO 1.01
*1.40 TYPE "A+B",A+B
*GO
A=    2.0000
B=    4.0000

?05 AT 1.30
*
```

?05 AT 1.30 indicates to the user that line 1.30 referenced a nonexistent line number, 1.01.

5.1.2 Using the Trace Feature

The user may want to check the logic in sections of his program. The trace feature is used for that purpose. To implement the trace feature, the user inserts a question mark (?) into a command string at any point. FOCAL-11 prints each succeeding character in the program as it is executed until another question mark is encountered or until the program returns to command mode.

For example, the trace feature is used to print parts of 3 lines in the following program:

```
          *ERASE ALL
          *1.1 SET A=1
          *1.2 SET B=5
          *1.3 SET C=3
          *1.4 TYPE ?A+B-C?,!
          *1.5 TYPE ?B+A/C?,!
          *1.6 TYPE ?B-C/A?,!
          *GO
          A+B-C= 3.0000
          B+A/C= 5.3333
          B-C/A= 2.0000
```

Extended Debug features are also available in FOCAL-11. To enable this feature, set FOCAL parameter 5 to a non-zero value, e.g. FPRM(5,1) (see Chapter 4 for information on the FPRM function). This mode, in addition to the normal debug output, displays the line number of the FOCAL statement to be executed. In the RT-11 version, the results of each SET and FOR command are also displayed.

```
          *ERASE ALL
          *1.1 TYPE %8.04;SET A=1
          *1.2 FOR I=1,3;DO 2
          *1.3 QUIT
          *2.1 SET B=(B+A)*I
          *2.2 RETURN
          *X FPRM(5,1)                (enables extended debug mode)
          *GO?                        (starts extended debug mode)

          C:FOCAL-11S V1  (RT-11) 14-AUG-74
            1.10 TYPE %8.04;SET A=1
          [A=   1.0000]
           1.20 FOR I=1,[I=    1.0000]
          3;DO2
            2.10 SET B=(B+A)*I
          [B= 106.0000]
            2.20 RETURN
          [I=   2.0000]            (this is from the FOR in line 1.2)
          DO 2
            2.10 SET B=(B+A)*I
          [B= 214.0000]
            2.20 RETURN
          [I=   3.0000]            (this is from the FOR in line 1.2)
          DO 2
            2.10 SET B=(B+A)*I
          [B= 645.0000]
            2.20 RETURN
          [I=   4.0000]            (this is from the FOR in line 1.2)
            1.30 QUIT
          *             (Debug mode is now turned off.)
```

5.2 CREATING A PAPER TAPE FOCAL-11 PROGRAM OFF-LINE

To create a FOCAL-11 program off-line, the procedure is:

1. Turn the terminal knob to LOCAL.
2. Set the low-speed punch to ON.
3. Type the off-line program. Type the RETURN key and the LINE FEED key at the end of each line. Every time RETURN and LINE FEED are typed, type CTRL/@ (CTRL/SHIFT/P) three times.

NOTE

If the user types an incorrect character while he is preparing an off-line tape, he can correct the error by pressing RUBOUT. The user then types the correct character.

To run a program created off-line, the procedure is:

1. Load FOCAL-11.
2. Put off-line program tape in the low-speed reader.
3. Switch the low speed reader to START.

If a high-speed paper-tape reader is available:

1. Load FOCAL-11.
2. Put the off-line program tape in the high-speed reader.
3. Type OPERATE R.
4. When the tape is finished being read, type CTRL/C to return to command mode.

Either way, the program will be put into core the same as it would if the user were typing it on the terminal keyboard.

5.3 ESTIMATING PROGRAM LENGTH

5.3.1 PAPER TAPE VERSION

In a 4K PDP-11, FOCAL-11 permits approximately 450 (decimal) word locations for program text and variables. Space not used by text may be used for variables. Since FOCAL-11 uses four words for each variable stored in the variable table, and one word for each two characters of stored program, the approximate length of a program may be determined by the formula

Length of program=4S+C/2+2L

where:

S=number of variables
C=number of characters in program
L=number of lines.

If the total program area or variable table area becomes too large, FOCAL-11 types an error message (?10 or ?11).

The following technique allows the user to find out how many memory locations are left for his use:

```
          *FOR I=1,1000; SET A(I)=1
                                              disregard error code
          ?10 AT 00.00
          *TYPE %4,I*4,"LOCATIONS LEFT"
          =910 LOCATIONS LEFT*
```

The number 1000 in the FOR command is assumed to be large enough for a 4K machine. A larger number would be required for a machine with more memory. At the end of this routine, use ERASE to clear all the variables A(I) from the symbol table.

5.3.2 RT-11 VERSION

Estimating the program size is slightly more complex in RT-11 than in Paper-Tape FOCAL. In Single precision:

LENGTH = 4*SV+3*NV+C/2+2*L+O

where:

SV	=	Number of subscripted variables
NV	=	Number of nonsubscripted variables
C	=	Number of characters in the program (excluding line numbers)
L	=	Number of lines
O	=	Overhead: 256 words per open file plus the maximum size required to contain all of the non-system device handlers required at one time by the user's FOCAL program. Each handler requires at least 256 words of memory. If a handler is larger than 256 words, memory is reserved in multiples of 256 words. Virtual files are not counted into the number of variables used in the above expression.

For double precision users, the coefficients for SV and NV should be changed to 6 and 5 respectively.

LENGTH(double precision) = 6*SV+5*NV+C/2+2*L+O

CHAPTER 6

ADVANCED FOCAL-11

6.1 DATA FORMATS

6.1.1 Fixed-Point

Unless it is instructed to do otherwise, FOCAL-11 produces numerical results showing up to eight digits, normally four digits to the left of the decimal and four digits to the right. This is called fixed-point format. Leading zeros are suppressed, and trailing zeros are printed. For example:

```
*TYPE 7777.7777,1111.1111,6666.6666
= 7777.7771= 1111.1110= 6666.6669
```

Although eight digits are shown in this format, only approximately seven digits are significant in single precision FOCAL. This is demonstrated in the above example by the number 1111.1111 which is printed as 1111.1110. The user may alter the fixed-point format by typing

```
*TYPE %w.dd,
```

where % is the FOCAL-11 format symbol, w is the total number of digits to be printed and dd is the number of digits to the right of the decimal point. Formats remain in effect until changed by the user.

The numbers w and dd, called the "format specification", must always be positive integers. If dd is less than ten, a leading zero must be inserted. For example:

```
*TYPE %4.02
*
```

indicates that the total number of digits is four and the number of digits to the right of the decimal is two.

Notice that the example below has a comma immediately after the last number of the format specification. The comma separates the format specification from the data it is to format.

```
*TYPE %4.02, 12.22+2.37
= 14.59*
```

Once a format is requested by the user, all subsequent evaluations appear in that format until the user changes it. For example, FOCAL-11 is designed to initially evaluate numbers with four digits to the left of the decimal and four to the right (TYPE %8.04,).

The TYPE format specification can also be an expression:

```
*SET V=3.02
TYPE %V,A,B,C
= 0.00= 0.00= 0.00*
```

6.1.2 Floating-Point

If the user wants to use numbers that are larger or smaller than possible with the fixed-point format, he may request a standard floating-point format by typing

```
*TYPE %,x
```

where X is the number to be evaluated.

For example:

```
*TYPE %,678
=  0.678000E+03*
```

Where E represents "times ten to the", and +03 is the power of ten (10↑3) The value of the expression is interpreted as .678 times 10↑3 or 678.

After floating-point format is requested, all subsequent evaluations appear in floating-point format until the user specifies another format.

The floating-point format enables FOCAL-11 to handle numbers as large as 1 times 10↑38 and as small as 1 times 10↑(-38). If the format specification is too small to print an evaluated number, FOCAL-11 will use floating-point format.

In double precision, the default size of the output field is extended to 12 digits after the decimal point. This allows for display of the extended precision. In addition, if a format of the form %w.dd is used with "w" being set to zero, "dd" digits will be displayed to the right of the decimal point.

The various numerical data formats available with FOCAL-11 are shown below:

Format	Number	Printed Result
TYPE %,	.123456E02	= 0.123456E+02
TYPE %8.04,	.123456E02	= 12.3456
TYPE %4.02,	.123456E02	= 12.35
TYPE %4,	.123456E02	= 12
TYPE %0.04	.123456E02	= 0.1235E+02

In addition, another mode of floating point notation is available with FOCAL-11. If the user alters FOCAL parameter 10 (see Section 4.13) to a non-zero value, scientific notation will be used in place of the normal floating point format discussed above. The difference between scientific notation and normal floating point representation is that the normal representation uses a fraction (a number between zero and one) and a power of ten. Scientific notation uses a value within the range of one and ten along with its power of ten.

For example:

Standard Notation	Scientific Notation
0.123456E+02	1.234560E+01
0.000000E+00	0.000000E+00
0.500000E+01	5.000000E+00

The FPRM function is used to enable and disable this mode:

```
XECUTE FPRM(10,1)
```

will enable scientific notation in place of normal floating point representation, while:

```
XECUTE FPRM(10,0)
```

will return FOCAL to normal floating point representation.

6.1.3 Text Output

The user may request FOCAL-11 to print text by typing

```
*TYPE "text"
```

where the text is a single character or a group of characters. The text is a string of characters which is printed exactly as it was typed on the user's terminal.

```
*TYPE "THIS IS AN EXAMPLE OF TEXT OUTPUT",!
THIS IS AN EXAMPLE OF TEXT OUTPUT
*
```

The carriage return and line spacing on the terminal are controlled with an exclamation mark (!). For example:

```
*TYPE !,"LINE 1",!,"LINE 2",!,"LINE 3",!
LINE 1
LINE 2
LINE 3
*
```

If the user wants to return the carriage without line spacing, he may use the number sign (#). For example:

```
*TYPE "L N  1",#," I E",!
LINE 1
*
```

NOTE

After typing 72 characters on a line, FOCAL-11 automatically types a CR-LF and continues on the next line. The value 72 may be changed using the FPRM function. See Chapter 4 for details.

6.1.4 Octal Output (↑O)

In addition to outputting a number in decimal notation, FOCAL-11 has the capability to display a constant, variable, or expression in the form of an octal number. This is performed by the special output sequence of

```
↑O(constant,variable, or expression)
```

used in place of a normal value in a TYPE command. This feature may only be used within the value range of -32,768 to 32,767. Attempts to use this feature outside of this range will result in an error.

This output mode does not normally output any leading zeros. For example:

```
*TYPE "ABCD",↑O(256),"EFGH",!
ABCD400EFGH
*TYPE ↑O(@256),!
256
*TYPE ↑O(-1),!
177777
*
```

Leading zeros may be obtained by using the FPRM function, described in Chapter 4, set parameter number 8 to a non-zero value.

NOTE

Throughout this discussion ↑CHAR is used to describe the sequence (↑ key) followed by (CHAR). This is distinguished from the normal connotation of CTRL/CHAR.

6.1.5 Binary Output (↑B)

The user may display output in the binary radix in a similar manner to the octal method described above. This is accomplished by using the form of

```
↑B(constant,variable, or expression)
```

in place of a value in the TYPE command. Leading zeros are not normally printed, but will be if FOCAL parameter 8 is set to a non-zero value. For example:

```
*TYPE ↑B(4)
100*X FPRM(8,1)
*TYPE ↑B(4)
```

```
          0000000000000100*SET A=5.5
          *TYPE ↑B(A+1)
          0000000000000110*X FPRM(8,0)
          *TYPE ↑B(A),"  ",↑O(A),"  ",A,!
          101  5 =    5.0000
          *
```

6.1.6 Integer Output (↑I)

It is at times convenient to place an integer value within a text string, and not have excess spaces placed in front of the number due to formatting. For this reason, a method similar to the ↑O and ↑B modes above is available for integer values. As an example, suppose a user desired to TYPE output of the form: THERE WERE xx ITERATIONS. and it is desired to not leave extra spaces within the line. This could be accomplished by the following statement:

```
          *SET ITER=348
          *TYPE "THERE WERE ",↑I(ITER)," ITERATIONS."!
          THERE WERE 348 ITERATIONS.
          *
```

The range of possible values which may be output in this manner is -32,768 to 32,767. Any attempt to output a number outside of this range will result in an integer overflow error (?38).

6.1.7 Current Date Output (↑D) (RT-11 only)

RT-11 maintains a current date in the format of DD-MMM-YY where DD is the day of the month, MM is the month (abbreviated as the first three characters), and YY as the last two digits of the current year. FOCAL-11 allows the user to take advantage of this and request the nine character date to be placed into the user's output line. This is accomplished by placing ↑D at the point in the TYPE command where the date is desired. For example:

```
          *TYPE "TODAY IS ",↑D
          TODAY IS 14-SEP-74*
```

6.1.8 Output Positioning (↑T)

FOCAL-11 also allows the user to tab to a specific location on an output line. This is provided by the ↑T special function. An expression, constant, or variable may be used in conjunction with this feature to enable the user to place the next character to be output in the specified column of the output line. The format of the ↑T feature is:

```
                    ↑T(constant, variable, or expression)
```

This feature may be used with the TYPE, ASK, or LIBRARY TYPE commands. The value of the associated argument must be within the range of one (1) to the current value of the terminal width parameter FPRM(2) (See Section 4.13).

If the expression is larger than the allowed value, no action is taken. If the expression designates a column which is lower in value than the current output position, a single space is output. However, if the argument specifies a column within the legal range of values which is higher in value than the current output position, sufficient spaces (blank characters) are output to position the next character in the specified column.

For example:

```
*1.10 TYPE !"SINE - COSINE TABLE"!!!
*1.20 TYPE ↑T(5),"X",↑T(15),"SIN(X)",↑T(30),"COS(X)"!
*1.30 FOR I=0,5,45; DO 2
*1.40 TYPE !!;QUIT

*2.10 XECUTE FPRM(3,3);TYPE !↑T(3),%2,I
*2.20 TYPE ↑T(14),%6.04,FSIN(3.14159*I/180)
*2.30 TYPE ↑T(29),FCOS(3.14159*I/180)

*G

SINE - COSINE TABLE


    X         SIN(X)         COS(X)

    0         0.0000         1.0000
    5         0.0872         0.9962
   10         0.1736         0.9848
   15         0.2588         0.9659
   20         0.3420         0.9397
   25         0.4226         0.9063
   30         0.5000         0.8660
   35         0.5736         0.8192
   40         0.6428         0.7660
   45         0.7071         0.7071
```

6.2 PROGRAMMING TECHNIQUES AND COMMENTS

To decrease program length and maximize available storage area, the programmer may use the following suggestions:

A. All commands may be abbreviated to one or two letters (see FOCAL-11 Command Summary, Appendix C).

B. A string of commands, except RETURN, MODIFY, QUIT, COMMENT, LIBRARY, and ERASE, can be combined on any one line (up to 72 characters), with each command separated by a semi-colon.

C. When a lengthy program is being written it is a good programming practice to leave free line numbers scattered throughout the body of the program. This will permit insertion of additional lines without complicated referencing routines. Remember that programs are executed in sequence by line numbers, therefore the order in which the lines are typed in is of no consequence. Line numbers must be in the range 1.01 to 99.99.

D. To avoid filling storage and have pushdown list errors occur during long routines, it may be helpful to limit the number of levels of nesting within arithmetic expressions. Using abbreviations and limiting the number of variable names will also maximize the use of storage space. Non-subscripted variables should be used wherever possible, as they will use one less word per variable than a subscripted variable.

E. Virtual files should be used wherever possible for large arrays. Besides saving memory space, virtual files can tend to be much faster than arrays stored in memory. This is due to the fact that arrays stored in memory must be size efficient, whereas virtual files need only be speed efficient.

F. When defining new variables, it is faster if they are defined in reverse alphabetical order if using the RT-11 FOCAL. This is due to the method by which new variables are stored in memory.

6.3 ADDING FUNCTIONS TO FOCAL-11 (FNEW)

FOCAL-11 has the capacity for one or more user defined and written assembly language functions. These user functions (called FNEW for purposes of this discussion) may use as many arguments as desired, and when interfaced to FOCAL-11, they can be used as any other FOCAL-11 function.

To write a function for FOCAL-11, the user should have a knowledge of programming the PDP-11 at the machine language level. The information in this secton describes the data formats and internal routines used by FOCAL-11, both of which may be used in the implementation of the FNEW function. This section also describes the procedure necessary to interface the function to FOCAL-11, so that it is callable in the FOCAL language. In addition to the information described in this section, a source listing for the version of FOCAL-11 in use would be most helpful to the programmer.

6.3.1 The FOCAL-11 Floating Point Package

Most FNEW functions will require the use of some floating-point operations through the FOCAL-11 floating-point package. The following is a discussion of the floating-point package and how to use it.

6.3.1.1 The Floating-Point Accumulator

One of the operands of all floating-point operations is the floating-point accumulator (FLAC), and the result of all floating-point operations is always stored in the floating-point accumulator.

```
FLAC:       exp + high mantissa

                low mantissa
```

The double precision package expands the mantissa field by an additional 2 words.

NOTE

Capitalized names in this discussion refer to specific symbols in the FOCAL-11 listing. To find their value or memory location for the version of FOCAL-11 that you are using, refer to the symbol table (Appendix G) at the end of this manual.

6.3.1.2 The FOCAL-11 Floating-Point Routines

The operations performed by the floating-point package are as follows:

FGET	to load an operand into the FLAC
FPOW	to raise the FLAC to an integer power (especially useful for evaluating series)
FMUL	to multiply the FLAC by the operand
FDIV	to divide the FLAC by the operand
FADD	to add the operand to the FLAC
FSUB	to subtract the operand from the FLAC
FPUT	to store the FLAC at the address specified

With each of these operations there is a choice of seven addressing modes for specifying the address of the operand.

In addition to the above, the floating-point package can be used to perform certain elementary functions:

FABS	to take the absolute value of the FLAC
FSGN	to leave the FLAC=-1 if the FLAC<0, 0 if the FLAC=0, +1 if the FLAC>0.
FINT	to convert the FLAC into an integer and leave the result in R1.
FLOAT	to convert the integer in R1 into a floating-point number and leave the result in the FLAC.
FNEG	to negate the FLAC
FCODE	to execute the floating point instruction in R1

The floating point operations are identified by adding the code for the operation plus the addressing mode to the octal value of 007000, thus making an illegal instruction. This is preceded by the FPMP trap call (104626). When the FPMP call is invoked, floating point operations defined in the manner above are performed by the floating point processor routines. Successive operations may be placed after a single FPMP call. Return from the floating point routines will be to the first word encountered by the routines which does not have as its base the illegal instruction 007000. As all user registers are preserved when entering the floating point routines. Placing several operations ofter a single FPMP trap call will save the time required for repetatively saving and restoring the registers. The codes for the floating point operations are:

Operation	Code	Complete instruction
FGET	00	7000+00+address code
FADD	10	7000+10+address code
FSUB	20	7000+20+address code
FDIV	30	7000+30+address code
FMUL	40	7000+40+address code
FPOW	50	7000+50+address code
FPUT	60	7000+60+address code
FINT	71	7000+71
FSGN	72	7000+72
FABS	73	7000+73
FNEG	74	7000+74
FLOAT	75	7000+75
(unused)	76	
FZER	77	7000+77
FCODE	200	7000+200

The addressing mode available to the floating point package are described below:

Mode	Code	Meaning
DIRECT	0	The address of the operand is in the next word.
IMMED	5	The next four words are the operand.
IPTR	1	The address of the operand is contained in register 2 (@R2).
XPTR	2	Register 2 is used as a pointer to the operand. After use, register 2 is updated by the addition of either 4 bytes (single precision), or 8 bytes (double precision).
THROUGH+STACK	3	The stack currently holds the pointer to the operand to be used.
FROM+STACK	3	The operand is contained on the stack. The single precision version only uses the top two words. The double precision version uses the top four words. For this reason, it is good

		practice to reserve four words of stack space for each variable on the stack. This allows the same code to be used by both the single and double precision versons without modification.
REL	6	The next word contains the offset from that location to the operand (address-.).

A complete instruction to add the constant 1.0 to the floating point accumulator (FLAC) would consist of:

```
104626              ;FPMP CALL
7000+10+5           ;BASE+FADD+IMMED
040200              ;FLOATING CONSTANT 1.0
000000              ;MUST BE 4 WORDS IN LENGTH
000000
000000              ;END OF THE CONSTANT
                    ;IF ANOTHER BASE+OPER+ADDR IS
                    ; USED HERE, IT WOULD BE PERFORMED,
                    ; IF NOT, THEN THE INSTRUCTION
                    ; PLACED HERE WOULD BE EXECUTED.
```

Addressing modes 3 and 4 (THROUGH+STACK and FROM+STACK which use the stack) do not perform the necessary stack maintenance. The user's floating point program must ensure that the necessary amount of area be reserved on the stack prior to using these operations. An example is shown below:

```
SUB     #10,SP      ;OPEN FOUR WORDS ON THE STACK
FPMP                ;INVOKE FLOATING POINT CALL
FPUT+3              ;SAVE FIRST OPERAND ON THE STACK
 .
 .
 .
SUB     #10,SP      ;OPEN A SECOND AREA
FPMP                ;CALL THE FLOATING POINT PACKAGE
FPUT+3              ;SAVE SECOND OPERAND
 .
 .
 .
FPMP                ;FLOATING POINT CALL
FGET+3              ;RECOVER THE SECOND VALUE
ADD     #10,SP      ;RELEASE THE AREA
 .
 .
 .
FPMP                ;CALL FLOATING POINT
FGET+3              ;RECOVER FIRST VALUE
ADD     #10,SP      ;RELEASE THE SAVE AREA
 .
 .
 .
```

Addressing modes 0 and 6 (DIRECT and REL) are two word operations. An example of these is shown below:

```
          FPMP                ;CALL FLOATING POINT
          FPUT+DIRECT         ;INDICATE A DIRECT CALL
          A                   ;LOCATION OF OPERAND.
           .
           .
           .
          FPMP                ;CALL FLOATING POINT
          FGET+REL            ;USE RELATIVE MODE
          A-.                 ;OFFSET TO OPERAND
           .
           .
           .
```

Following are some examples of routines which use the FOCAL-11 floating point routines:

EXAMPLE 1

A routine is required to add successive numbers in an ASCII string which is pointed to by register 2 until a zero value is encountered. The result is to then be output to the terminal.

```
          FREAD=104664        ;READ ROUTINE
          FPRINT=104666       ;PRINT ROUTINE
          FPMP=104626         ;FLOATING POINT CALL
          FPP=007000          ;FLOATING POINT BASE
          GETC=104614         ;READ A CHARACTER (R2)+
;
START:    SUB     #10,SP      ;OPEN AN AREA
          FPMP                ;FLOATING POINT CALL
          FPP+77              ;ZERO THE FLAC
LOOP:     FPMP                ;NEW ENTRY POINT (FPMP!)
          FPP+60+3            ;STORE THE FLAC ON THE STACK
          GETC                ;EXTRACT A CHARACTER (FIRST ONE)
          FREAD               ;INTERPRET AS A NUMBER
                              ; THE RESULT IS IN THE FLAC
          TST     FLAC        ;SEE IF ZERO
          BEQ     DONE        ;EXIT IF SO
          FPMP                ;ELSE ADD TO THE TOTAL
          FPP+10+3            ;ADD THE SUBTOTAL TO THE FLAC
                              ;LEAVING THE RESULT IN THE FLAC
          BR      LOOP        ;CONTINUE UNTIL A ZERO IS FOUND
;
DONE:     FPMP                ;ENTER FLOATING POINT ROUTINES
          FPP+00+3            ;RECOVER THE RESULT
          ADD     #10,SP      ;RELEASE THE WORK AREA
          CLR     R3          ;SET FOR FLAOTING POINT FORMAT
          FPRINT              ;OUTPUT THE VALUE TO THE
                              ; TERMINAL
          RTS     PC          ;POSSIBLE RETURN THIS WAY...
```

EXAMPLE 2

The following section of code evaluates X↑2+2X+1 assuming that R2 initially points to X.

```
START:    SUB      #10,SP          ;OPEN A WORK AREA
                                   ; ON THE STACK
          FPMP                     ;CALL THE FLOATING POINT
                                   ; PACKAGE
          FPP+77                   ;ZERO THE FLAC
          FPP+00+1                 ;GET X (R2)
          FPP+40+1                 ;*X=X↑2
          FPP+60+3                 ;SAVE X↑2 ON THE STACK
          FPP+00+1                 ;GET X AGAIN
          FPP+10+1                 ;+X=2X
          FPP+10+5                 ;ADD ONE
          040200                   ;FLOATING POINT
          000000                   ; VALUE OF
          000000                   ; 1.0 MUST BE
          000000                   ; FOUR WORDS LONG!
          FPP+10+3                 ;ADD X↑2 FROM THE STACK
          ADD      #10,SP          ;RELEASE THE WORK AREA
                                   ;AT THIS POINT, X↑2+2X+1 IS
                                   ; LEFT IN THE FLAC FOR FURTHER
                                   ; USE.
```

It should be noted that use of the stack for temporary storage can reduce the core requirements for the floating point routines, as single word operations can be used. In addition, there is no need for special memory locations to be set aside for storage of these temporary values. The above example should make this quite clear.

6.3.1.3 Using Standard Functions

Besides those arithmetic operations available through the floating-point package, the user may wish to call the standard functions provided with FOCAL-11. These functions may then be called by setting up an argument and then executing a

```
JSR PC,STARTING ADDRESS
```

The starting addresses of the various functions can be found from the symbol table, and are named as shown in the table below:

Function	Entry Point	Argument Address
FSIN	FSIN	FLAC
FCOS	FCOS	FLAC
FRAN	XRAN	FLAC
FCLK	XFCLK	FLAC
FSQT	XSQT	FLAC

NOTE

Only functions requiring a single argument may be called in this fashion.

For example:

1. To call the SIN function

```
FPMP
FGET+DIRECT,ARG                         ;LOAD ARG INTO FLAC
JSR PC,FSIN                             ;CALCULATE SINE
FPMP
FPUT+DIRECT,RESULT                      ;ON RETURN, RESULT
                                        ;IS IN FLAC.
```

2. To call the FSQT function

```
FPMP
FGET+DIRECT,ARG                         ;ARG INTO FLAC
JSR PC,XSQT
FPMP
FPUT+DIRECT,RESULT                      ;SAVE RESULT
```

3. To call the Random Number Generator

```
      .
      .
      .
FPMP                                    ;FLOATING CALL
FZER                                    ;ZERO FLAC
JSR PC,XRAN                             ;NO NEED FOR ARG
FPMP
FPUT+DIRECT,RESULT                      ;SAVE RESULT
```

6.3.2 FOCAL-11 Subroutines

FOCAL-11 has many subroutines that can prove quite useful when writing an FNEW. These subroutines are permanent parts of the FOCAL-11 interpreter, and are called via the TRAP instruction.

FORMAT: TRAP(104400)+TRAP CODE(8 bits)

NOTE

In the following text, CHAR is equal to R4. In the discussion of SORTC and TESTC, the entries labeled "RETURN" represent the addresses of routines to be entered if the condition is met, not instructions that are actually executed.

Mnemonic	Trap Code (Octal)	Description
GETC	214	Get the next character from the text; character is returned in CHAR.
SORTC	202	Compare contents of CHAR against list. Calling sequence:

```
SORTC      ;CALL
LIST       ;ADDRESS OF LIST
           ;OF BYTES.
RETURN     ;RETURN IF IN LIST
   .       ;RETURNS HERE IF NOT
   .       ;IN LIST.
   .
   .
   .
```

NOTE

Lists are terminated by a zero byte.

PRINTC	204	Print the contents of CHAR (7 bit ASCII) or internal code for terminators.
OUTCH	210	Print 8 bit ASCII in CHAR.
READC	206	Read and echo a character from the keyboard and put it into CHAR in FOCAL internal code.
INCH	212	Read 8 bit ASCII character into CHAR.
SPNOR	234	Ignore spaces in text; exit with the first character that is not a space in CHAR.
ERROR	201+error no.*2	Transfer control to the error routine and terminate execution; print error message. (error no. is the octal equivalent of the appropriate error diagnostic message from Appendix B)
TESTC	220	This subroutine is a series of SORTC's with various returns:

```
CALL: TESTC      ;CALL

      RETURN1    ;TERMINATOR RETURN

      RETURN2    ;NUMBER RETURN

      RETURN3    ;FUNCTION RETURN
                 ;CHARACTER ="F".

         .       ;RETURN HERE IF
         .       ;ALPHABETIC
         .       ;CHARACTER.
```

SORTJ	200	This subroutine is used as a multiple sort and branch subroutine. CHAR is compared to a list. If it is in the

LIST AN ADDRESS IS LOOKED UP AND AN EFFECTIVE JMP address is executed. If a match is not in the list, then return is to CALL+6.

```
CALL: SORTJ
      LISTCHAR  ;ADDRESS OF
                ;CHARACTER LIST
                ;TO BE SEARCHED.
      LIST ADDR ;ADDRESS OF JUMP
         .      ;    LIST.
         .      ;RETURNS HERE
         .      ;IF NOT IN
         .      ;LIST ADDR
```

EVAL.X	260	This subroutine evaluates an arithmetic expression. The subroutine return is to CALL+2 with the floating-point value of the expression it evaluated in the FLAC.
GETLN	222	This routine accepts a character string pointed to by R3 and returns a value in LINENO in special line number notation (integer portion of the line number * 256; i.e.2.5 would return in LINENO the octal value of 001200 = 640 decimal = 2.5*256.0).
FINDLN	224	This routine is used in conjunction with the GETLN routine explained above. The line number stored in "LINENO" is used to look up the desired line. If the line is now found, return is to the routine pointed to by the word following the call. In this case, R2 points to the line prior to the desired line, and R3 points to the next line (or zero if no more lines in the program). If the line is found, return is to the instruction following the not-found pointer with R3 set up for GETC calls to extract the ASCII data (start of line plus 8).
PRNTLN	226	This routine outputs a line number which has been stored in R1 prior to call. This line number is in the same format as when stored in LINENO.

ERASEV	236	This routine erases all variables.
PRINT2	242	This routine prints the characters stored in the word following the call. The routine returns to the instruction following this word.
DIGTST	244	This routine performs an integer division by the value stored in the location following the call. The number to be divided is stored in R2 prior to the call. R4 is initially set to the ASCII character "0" by the routine, and is incremented each time the divisor is subtracted from R2. This leavs a printable result in R4 and R2 has the remainder (always less than the divisor). Return is to the instruction following the divisor value.
SKPNON	254	This routine examines the contents of R4 and returns either to the routine pointed to by the word following the call if the character in R4 is between "0" and "9", or to the instruction following the pointer if it is not within that range.
FPMP	262	This routine is used to invoke the floating point interpreter. All registers are saved, and words following the call are interpreted as floating point instructions. Return is to the first instruction following the pseudo floating point instructions (Floating point instructions are based upon the illegal instruction 007XXX. As soon as an instruction not of this form is detected, the registers are restored and control is passed to that instruction.).
FREAD	264	This routine reads characters pointed to by R3 and evaluates them as a floating point value. The result is returned in FLAC.

FPRINT	266	This routine prints a value stored in the FLAC according to the format code stored in R2. The format code is in the form of a line number. (see GETLN)

6.3.2.1 Passing Arguments to FNEW

The above FOCAL subroutines are most often used when passing arguments to FNEW from the main program. When control is passed to the FNEW function, the first argument in the argument list is in the FLAC. If this is the only argument desired, the FNEW need only perform its function and place the result in the FLAC prior to returning (via RTS PC). More than one argument may be passed, however, using the following techniques:

Assume an FNEW that uses three arguments: FNEW (X,Y,Z). When control is passed to the function, the value of X is in the FLAC, with the contents of CHAR (Register 4) containing the "," following the X in the argument list.

```
SET X1 = FNEW(X,Y,Z)



         FLAC  CHAR
```

When FNEW requires the value for Y, it executes a call to EVAL to evaluate the expression for the second argument into the FLAC, and repeats the process for Z. For example:

```
;PORTION OF FNEW TO SAVE THE THREE ARGUMENTS
;ON THE STACK

         FPMP=104662           ;FLOATING POINT CALL
         FPP=007000            ;FLOATING POINT TRAP BASE
         EVAL.X=104660         ;CODE FOR EVAL.X
ARGGET:  CMP -(SP),-(SP)       ;OPEN TWO LOCATIONS ON STACK
         FPMP                  ;INVOKE FLOATING POINT MODE
         FPP+60+3              ;SAVE X ON THE STACK
         EVAL.X                ;EVALUATE Y INTO THE FLAC
         CMP -(SP),-(SP)       ;OPEN 2 MORE STACK LOCATIONS
         FPMP                  ;FLOATING POINT
         FPP+60+3              ;SAVE Y ON THE STACK
         EVAL.X                ;EVALUATE Z INTO THE FLAC
         CMP -(SP),-(SP)       ;OPEN LAST 2 STACK LOCATIONS
         FPMP                  ;FLOATING POINT
         FPP+60+3              ;SAVE Z ON THE STACK
           .
           .
           .
```

A variable number of arguments may be passed by testing the value of CHAR after each use of EVAL.X for the right parenthesis in the function call (using TEST or SORTC). When the right parenthesis is detected, the end of the argument list has been reached.

For example, we can create an FNEW that sums all the arguments and returns that sum. Sample calls to this FNEW might appear as follows:

```
          SET X = FNEW(1,2,3,4,5,6)
```

OR

```
          SET X = FNEW(A,B,C/D)
```

The code to implement this FNEW would look like:

```
        FADD=007000+10     ;DEFINE FLOATING POINT
        FPUT=007000+60     ;CALLS
        STACK=3            ;ADDRESSING MODE
        EVAL.X=104660      ;TRAP CALL TO EVAL
        FPMP=104662        ;FLOATING POINT CALL
        SORTC=104602       ;TRAP CALL TO SORTC

LIST:   .BYTE 211          ;INTERNAL CODE FOR ")"
                           ;OBTAINED FROM TABLE
                           ;IN APPENDIX H.
        .BYTE 0            ;A ZERO BYTE
                           ;  TO TERMINATE LIST

FNEW:   CMP -(SP),-(SP)    ;OPEN 2 WORDS ON STACK
        FPMP               ;CALL FLOATING POINT
        FPUT+STACK         ;SAVE FIRST ARG ON STACK
        SORTC              ;TEST CHAR FOR RIGHT PARENTHESIS
        LIST               ;ADDRESS OF LIST
        DONE               ;JUMPS TO "DONE" IF NEXT CHAR ")"
        EVAL.X             ;MORE ARGS TO COME - GET
                           ;NEXT ONE INTO FLAC
        FPMP               ;FLOATING POINT CALL
        FADD+STACK         ;ADD TO CUMULATIVE SUM ON STACK
        BR FNEW            ;AND LOOP
DONE:   FPMP               ;ENTER FLOATING POINT
        FGET+STACK         ;GET FINAL RESULTS INTO FLAC
        CMP (SP)+,(SP)+    ;CLEAR UP STACK
        RTS PC             ;AND RETURN
```

6.3.3 FOCAL-11 Data Structure

6.3.3.1 Text Data

6.3.3.1 Text Data - FOCAL-11 text is stored in linked lines and each line is stored as a string of single 7-bit ASCII characters, byte to byte, with all terminators coded in a special internal form. The last character in a line is always 216 (the internal form for CR), and no further characters should be used beyond the CR until a new line is found. The internal codes for all terminators may be found in Appendix H.

Byte No.	Octal Code	Letter
0	101	"A"
1	102	"B"
2	200	space code
3	216	CR code

```
          102          101

          216          200
```

These characters may be accessed by means of the subroutine "GETC", which returns the next character in CHAR. If the character fetched by GETC was a terminator, the sign bit will be set on return.

```
GETC            ;READ TEXT
BMI DONE        ;TERMINATOR FOUND
```

6.3.3.2 Text Lines - The FOCAL-11 program is stored as a series of linked text lines.

Each line has associated with it an explicit line number, composed of a group number plus an extension. For example, 12.50 has a group number of 12. These two bytes of binary data form the second word of the line storage. The first word is a 16-bit pointer to the next line. The pointer is taken relative to the current location minus two (NEXT-.-2). This makes it easy to debug the program using Octal Debugging Technique program (ODT) to compute the relative address. Furthermore, a single word instruction is all that is required to chain to the next entry. ADD (R0)+,R0 makes R0 point to the next line.

All input is packed into a special buffer before it is either interpreted as a direct command or stored as part of the text buffer.

next-.-2		
005200		(12.5*256.0)
byte #2	byte #1	
. . .		
	216	

6.3.3.3 Text Input and Output - A 7-bit ASCII byte placed in the register CHAR can be printed by calling the PRINTC routine. Internal codes for terminators are automatically translated before printing. An input ASCII byte or internal terminator code may be read from the current input device by the routine READC. The result of a READC is left in the register CHAR. Pure 8-bit ASCII input and output are handled by the routines INCH and OUTCH respectively.

6.3.3.4 Variables (Paper Tape FOCAL)
To maximize the efficiency of memory usage, FOCAL-11 variables are created as they are used, and each variable is stored with its own subscript. Each variable has an exponent of eight bits and a signed 24-bit mantissa as follows:

name
subscript
exp + high mantissa
+ low mantissa

The name may be one or two characters. If it is one, the high-order byte is blank, as the first character of the name is always in the low-order byte. If the variable is unsubscripted, the subscript word is 0, while a double subscripted variable has the first subscript in the low-order byte of the subscript word and the second subscript in the high-order byte.

blank	name
sub 2	sub 1
exp + high mantissa	
+ low mantissa	

6.3.3.5 Variables (RT-11 FOCAL)
In general, the means of storing variables in RT-11 FOCAL is similar to that in Paper Tape FOCAL. The basic format is as follows:

name
subscript
exp + high mantissa
low mantissa

The major difference made to the Paper Tape structure is in the "name" and "subscript" field.

The "name" field consists of the two characters which make up the variable name. The first character, unlike the papertape version, is always in the high byte. If a second character exists, it is placed into the low byte.

Since no terminator character can be used as part of a variable name, it is guaranteed that each character will be in the range of 0-177 octal. This leaves the high bit in each of the bytes holding the name free to carry other information. In order to conserve memory, the highest bit of the highest byte (bit 15) is used to indicate if a non-zero subscript is used. If this bit is set, the next word contains the subscript value. If not set, the value of the variable immediately follows the name field.

name
exp + high mantissa
low mantissa

The high bit of the low byte (bit 7) is used to indicate whether the variable is to be interpreted as a singly or doubly subscripted variable. In the normal subscripting mode (FPRM(1)=0) this bit is ignored. If FPRM(1)=1, and if this bit is set, the following word will be treated as a double subscript.

name		(bit 15=1,bit 7=1)
sub2	sub1	
exp + high mantissa		
low mantissa		

The double precision version of RT-11 FOCAL extends the mantissa fields by an additional two words.

In extended subscript mode (FPRM(1) not equal to zero), if a user was to use the variable A(n), the user would be able to also use the variable A(n,m) to store information separately from the first variable. In addition, the scalar (non-subscripted) variable A can contain a different value from the subscripted variable A(0).

When using normal subscript mode, the variables A, A(0), and A(0,0) are the same variable.

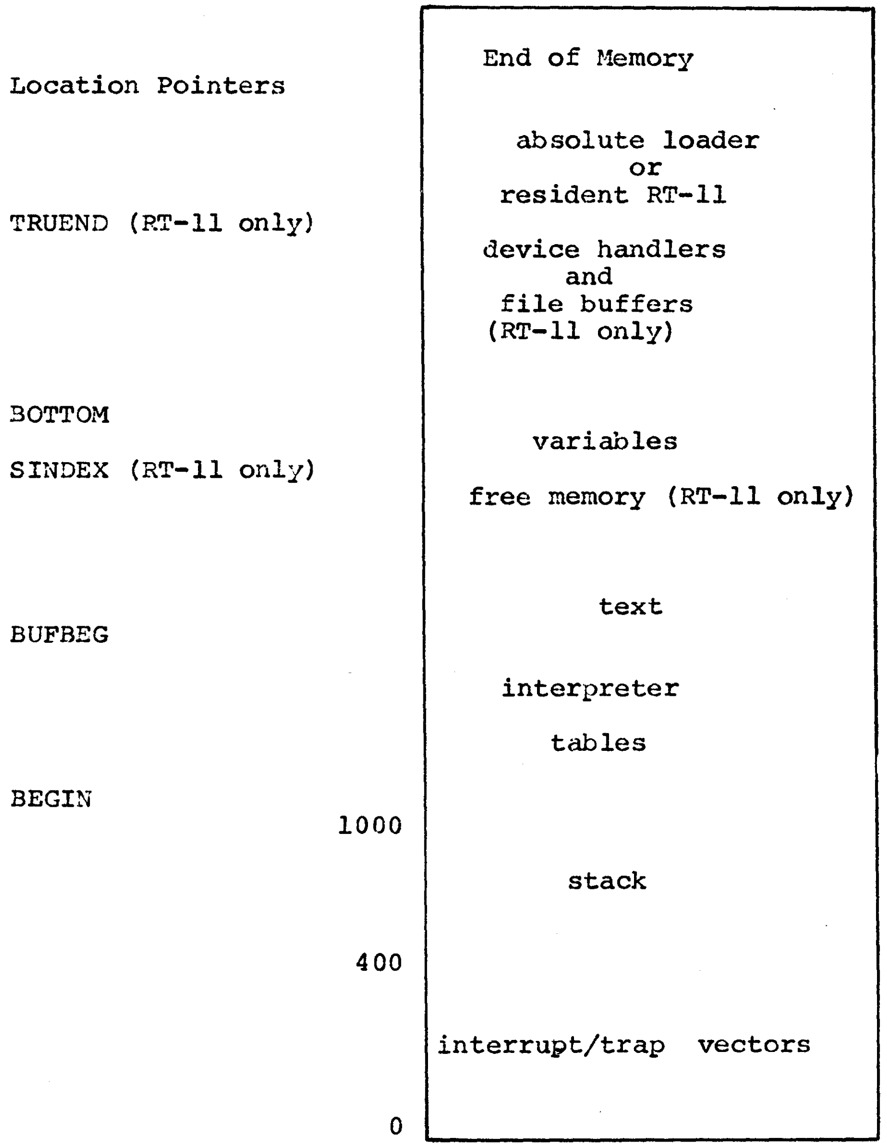
When changing subscript modes, it is essential that the user combine the FPRM function call with an ERASE command. If this is not performed, it is possible that variables previously stored as double subscripted values will only be able to be accessed as single subscripted variables. When converting from the extended subscript mode back to normal mode, it is possible that some variables will no longer be able to be referenced.

If it is essential that the subscript mode be changed during program execution, and the variables must remain intact, the following sequence of FOCAL statements can be used:

```
     LIBRARY MAKE 7,SYMBOL.TMP
     LIBRARY TYPE 7,$
     LIBRARY CLOSE 7
     ERASE
     XECUTE FPRM(1,x)
     LIBRARY GET SYMBOL.TMP
     LIBRARY DELETE SYMBOL.TMP
```

This will cause a file named SYMBOL.TMP to be created, and the current variables to be written to the file. The variables are then erased, and the subscript mode is changed. The LIBRARY GET command is then used to input the variables.

6.3.3.6 Memory Layout - The memory layout of FOCAL-11 is arranged in such a way as to take advantage of the hardware stack limit at 400 and also to allow dynamic allocation of resources between the text and the variables. In the paper tape version of FOCAL-11, the variable/text area is adjusted automatically upon loading to use all memory up to the beginning of the absolute binary loader. In the RT-11 version of FOCAL-11, the variable area resides in the highest portion of memory, just below the resident monitor.



6.3.4 Interfacing FNEW to FOCAL-11

Once the FNEW function has been written, its name must be entered in FOCAL-11's function table such that the function can be referenced by FOCAL-11 statements. First the name must be created, then added to the FOCAL function table.

6.3.4.1 Naming the Function - All function names in FOCAL must be prefaced by "F". There are no other restrictions on the mnemonic chosen. The user should keep in mind when choosing the function name that each character in the mnemonic will occupy one byte of memory each time the function is used, hence shorter names are more core efficient.

FOCAL-11 stores the function name internally, in a special 16-bit hash code. To form this code, perform the following operations:

	Step	Example Using "FSBR"

1.	Determine the 7-bit ASCII value for each letter in the mnemonic.	F=106
		S=123
		B=102
		R=122

2.	Multiply each of these letter values by its place value in a base 5 system; i.e., the rightmost number is multiplied by 4↑2, etc., until the "F" is multipled by 4↑ (# of chars in name). Remember to perform all arithmetic in octal. If the result is greater than 200000 octal, bits to the left are lost and zeroes are brought in from the right (Modulo 200000).	R=122 = 122
		B=102 x 4 = 410
		S=123 x 20 = 2460
		F=106 x 100 = 10600

3.	Sum the products to form the hash code (in this example, for FSBR).	14012

It is possible that this hash code conflicts with one of the codes already in the function table. Since all function codes must be unique, compare the hash code for the new name against the following list of hash codes for functions already in FOCAL-11. If the hash code is not unique, a new name must be chosen.

Function	Hash Codes
FCLK	13453
FITR	13662
FCOS	13477
FSIN	14042
FSQT	14110
FADC	13343
FX	560
FLN	2736
FLOG	13703
FEXP	13600
FPRM	14025
FINT	13634
FQUE	14051
FRAN	13762
FCHR	13442
FSBR	14012
FSGN	14032
FABS	13353

In order to help the user to generate the hash code needed for the FNEW function, the following FOCAL program can be used.

```
1.10 S A=0;S S=0;C-INITIALIZE
1.15 T !"ENTER THE FUNCTION NAME"
1.20 S A=FCHR(FCHR(-1));C-GET CHARACTER
1.30 I (A-@15) 1.2,1.9;C-DETECT CARRIAGE RETURN
1.35 C-DROP BITS OFF THE LEFT...
1.40 S S=S*4+A;I (S-65536)1.2,1.2;S S=S-65536;G 1.2
1.80 C-CONVERT FROM 16 BIT UNSIGNED TO 15 BIT SIGNED
1.90 I (FABS(S)),1.99;I (S-32768)1.95;S S=S-32768+@100000
1.95 T !"THE CORRECT HASH CODE IS: ",↑O(S),!;G 1
1.99 T !!;Q
```

6.3.4.2 Entering the Function Into FOCAL-11 - Once the hash code has been determined, the function is added to FOCAL-11 as follows:

6.3.5 FNEW Example

The following step by step example of implementing an FNEW function illustrates the passing of arguments and use of FPP. This particular FNEW, which we shall call FQUAD solves a quadratic equation by means of the quadratic formula. It requires three argument (A,B,C) representing the coefficients of the equation to be solved and a fourth argument for the root desired. The code to implement the function FQUAD in both the Paper Tape and RT-11 versions of FOCAL-11 follows:

Paper-Tape Version:

```
                .TITLE  FQUAD - FOCAL FNEW (PAPER-TAPE)
;
;
;               FUNCTION FQUAD(A,B,C,R)
;
;
;               THIS IS AN FNEW FUNCTION TO FIND ONE OF THE ROOTS
;               OF A QUADRATIC EQUATION OF THE FORM:
;
;                       A (X↑2) + B (X) + C = 0.0
;
;               THIS FUNCTION REQUIRES FOUR ARGUMENTS. THE FIRST
;               THREE ARE THE COEFFICIENTS A,B, AND C. "R" IS THE
;               SIGN OF THE SQUARE ROOT TERM IN THE EXPRESSION:
;
;                       -B +/- FSQT(B↑2-4AC)
;                       --------------------
;                               2A
;
```

```
                ;       WITH "R" BEING +1 FOR POSITIVE, AND -1 FOR NEGATIVE.
                ;
                ;
                ;       IMMAGINARY ROOTS GENERATE AN ERROR MESSAGE ?63 (UNIQUE)
                ;
                ;
                ;       FLOATING POINT CALL DEFINITIONS
                ;
        007000          FGET=7000
        007010          FADD=7010
        007020          FSUB=7020
        007030          FDIV=7030
        007040          FMUL=7040
        007050          FPOW=7050
        007060          FPUT=7060
        007070          FUNCT=7070
                ;
                ;       FLOATING POINT ADDRESSING MODES
                ;
        000005          IMMED=5                 ;OPERAND FOLLOWS THE INSTRUCTION
                                                ; AND MUST BE 4 WORDS LONG!
        000001          IPTR=1                  ;R2 POINTS TO ARGUMENT
        000003          STACK=3                 ;2 OR 4 WORD VALUE ON STACK
        000000          DIRECT=0                ;POINTER FOLLOWS INSTRUCTION
                ;
                ;       FLOATING POINT FUNCTIONS
                ;
        007074          FNEG=FUNCT+4            ;NEGATE THE FLAC
        007075          FLOAT=FUNCT+5           ;CONVERT R1 TO A FLT PT VALUE
                ;
        013404          FSQT=13404              ;TAKE THE SQR RT OF FLAC
                ;
        104660          EVAL.X=104400+260       ;EVALUATE THE EXPRESSION
        104662          FPMP=104400+262         ;ENTER FLOATING POINT MODE
                ;
        104400          ERROR=104400            ;BASE FOR ERROR MESSAGES
                ;
                ;
        000000          R0=%0
        000001          R1=R0+1
        000002          R2=R1+1
        000003          R3=R2+1
        000004          R4=R3+1
        000005          R5=R4+1
        000006          SP=R5+1
        000007          PC=SP+1
                ;
        001050          FBASE=1050              ;FUNCTION BASE ADDRESS
                ;
        001610          FLAC=1610               ;FLT PT ACCUMULATOR
                ;
        001652          BOTTOM=1652
                ;
        000000          .ASECT
                ;
        017314          .=17314                 ;CALCULATED BY USING
                                                ; ABSOLUTE LOADER BASE ADDRESS
                                                ; -6 (STACK SPACE) - FUNCTION
                                                ; SIZE. THIS IS FOR A 4K SYSTEM.
                ;
017314 162706 FQUAD:    SUB     #10,SP          ;OPEN 4 WORDS ON THE STACK
```

```
       000010
                                                ; BY USING 4 WORDS, THIS
                                                ; FUNCTION COULD BE USED
                                                ; BY BOTH THE SINGLE AND
                                                ; DOUBLE PRECISION PACKAGE.
017320 104662          FPMP                     ;TRAP CALL TO INVOKE FLT PT
017322 007065          .WORD    FPUT+IMMED      ;SAVE A
017324 000000 A:       .WORD    0,0,0,0         ;FOUR WORDS
017326 000000
017330 000000
017332 000000
017334 104660          EVAL.X                   ;GET B
017336 104662          FPMP                     ;FLOATING CALL
017340 007065          .WORD    FPUT+IMMED      ;SAVE B
017342 000000 B:       .WORD    0,0,0,0         ;B
017344 000000
017346 000000
017350 000000
017352 104660          EVAL.X                   ;GET C
017354 012701          MOV      #4,R1           ;SAVE A FOUR FOR LATER
       000004
017360 012702          MOV      #A,R2           ;POINT TO "A" FOR IPTR CALLS
       017324
017364 104662          FPMP                     ;FLOATING POINT
017366 007041          .WORD    FMUL+IPTR       ;MULTIPLY BY "A" (A*C)
017370 007063          .WORD    FPUT+STACK      ;SAVE IT ON THE STACK
017372 007075          .WORD    FLOAT           ;FLOAT R1 (R1=4)
017374 007043          .WORD    FMUL+STACK      ;MULTIPLY BY FOUR
017376 007063          .WORD    FPUT+STACK      ;PLACE IT BACK (4AC)
017400 007000          .WORD    FGET+DIRECT     ;RECOVER B
017402 017342          .WORD    B
017404 007040          .WORD    FMUL+DIRECT     ;MULTIPLY AGAIN FOR B2
017406 017342          .WORD    B
017410 007023          .WORD    FSUB+STACK      ;B2-4AC
017412 005767          TST      FLAC            ;SEE IF NEGATIVE NUMBER
       162172
017416 100424          BMI      SETERR          ;SET ERROR ?63
017420 004767          JSR      PC,FSQT         ;PERFORM SQUARE ROOT
       173760
017424 104662          FPMP                     ;ENTER FLOATNG POINT
017426 007063          .WORD    FPUT+STACK      ;SAVE FSQT(B2-4AC)
017430 104660          EVAL.X                   ;GET R
017432 012702          MOV      #A,R2           ;POINT AT "A"
       017324
017436 104662          FPMP                     ;ENTER FLOATING POINT
017440 007043          .WORD    FMUL+STACK      ;SET CORRECT SIGN
017442 007020          .WORD    FSUB+DIRECT     ;SUBTRACT OUT B
017444 017342          .WORD    B               ; (-B)
017446 007063          .WORD    FPUT+STACK      ;SAVE IT ON THE STACK
017450 007001          .WORD    FGET+IPTR       ;GET A
017452 007011          .WORD    FADD+IPTR       ;MULTIPLY BY 2
017454 007061          .WORD    FPUT+IPTR       ;SAVE BACK IN A
017456 007003          .WORD    FGET+STACK      ;RECOVER NUMERATOR
017460 007031          .WORD    FDIV+IPTR       ;DIVIDE BY 2A
017462 062706          ADD      #10,SP          ;FIX UP THE STACK
       000010
017466 000207          RTS      PC              ;RETURN WITH THE ANSWER
                                                ; IN THE FLAC
              ;
017470 104777 SETERR: ERROR+201+63.+63.         ;SEND ERROR 63 (IMMAGINARY
                                                ; ROOTS!)
```

```
                        ;
                        ;
                        ;---    HERE WE PATCH FOCAL PAPER TAPE TO HANDLE THINGS RIGHT!
                        ;
                        ;
                        ;
                        ;       HASH CODE CALCULATION:
                        ;
                        ;       1) DETERMINE ASCII VALUES
                        ;               F=106
                        ;               Q=121
                        ;               U=125
                        ;               A=101
                        ;               D=104
                        ;
                        ;
                        ;       2) MULTIPLY CORRECTLY
                        ;
                        ;       D=104             104
                        ;       A=101*4           404
                        ;       U=125*42         2520
                        ;       Q=121*43        12100
                        ;       F=106*44        43000
                        ;
                        ;       3) PERFORM SUM  -----
                        ;
                        ;                       60330
                        ;
                        ;
       001050                   .=FBASE                 ;FUNCTION BASE ADDRESS
                        ;
001050 017314                   .WORD   FQUAD           ;ENTER IT INTO THE TABLE
001052 060330                   .WORD   60330           ;ENTER THE HASH CODE ALSO
                        ;
       001044                   .=FBASE-4               ;TERMINATE THE TABLE
001044 000000                   .WORD   0,0             ;WITH ZEROS
001046 000000
                        ;
                        ;
                        ;       SET NEW TOP LIMIT FOR FOCAL
                        ;
       001652                   .=BOTTOM
                        ;
001652 017314                   .WORD   FQUAD
                        ;
                        ;
                        ;
       000001'                  .END

SYMBOL TABLE

A         017324        B         017342        BOTTOM= 001652
DIRECT= 000000          ERROR = 104400          EVAL.X= 104660
FADD  = 007010          FBASE = 001050          FDIV  = 007030
FGET  = 007000          FLAC  = 001610          FLOAT = 007075
FMUL  = 007040          FNEG  = 007074          FPMP  = 104662
FPOW  = 007050          FPUT  = 007060          FQUAD   017314
FSQT  = 013404          FSUB  = 007020          FUNCT = 007070
IMMED = 000005          IPTR  = 000001          PC    =%000007
R0    =%000000          R1    =%000001          R2    =%000002
```

```
R3     =%000003          R4     =%000004          R5     =%000005
SETERR= 017470           SP     =%000006          STACK = 000003
. ABS.  017472      000
        000000      001
ERRORS DETECTED: 0
FREE CORE: 9620. WORDS
FQUAD,FQUAD=FQUAD.PTP
```

RT-11 Version:

```
                         .TITLE  FQUAD - FOCAL FNEW (RT-11)
               ;
               ;
               ;         FUNCTION FQUAD(A,B,C,R)
               ;
               ;
               ;         THIS IS AN FNEW FUNCTION TO FIND ONE OF THE ROOTS
               ;         OF A QUADRATIC EQUATION OF THE FORM:
               ;
               ;                 A (X↑2) + B (X) + C = 0.0
               ;
               ;         THIS FUNCTION REQUIRES FOUR ARGUMENTS. THE FIRST
               ;         THREE ARE THE COEFFICIENTS A,B, AND C. "R" IS THE
               ;         SIGN OF THE SQUARE ROOT TERM IN THE EXPRESSION:
               ;
               ;                 -B +/- FSQT(B↑2-4AC)
               ;                 ____________________
               ;                         2A
               ;
               ;         WITH "R" BEING +1 FOR POSITIVE, AND -1 FOR NEGATIVE.
               ;
               ;
               ;         IMMAGINARY ROOTS GENERATE AN ERROR MESSAGE ?63 (UNIQU
               ;
               ;
               ;         FLOATING POINT CALL DEFINITIONS
               ;
         007000          FGET=7000
         007010          FADD=7010
         007020          FSUB=7020
         007030          FDIV=7030
         007040          FMUL=7040
         007050          FPOW=7050
         007060          FPUT=7060
         007070          FUNCT=7070
               ;
               ;         FLOATING POINT ADDRESSING MODES
               ;
         000005          IMMED=5                  ;OPERAND FOLLOWS THE INSTRUCT
                                                  ; AND MUST BE 4 WORDS LONG!
         000001          IPTR=1                   ;R2 POINTS TO ARGUMENT
         000003          STACK=3                  ;2 OR 4 WORD VALUE ON STACK
         000000          DIRECT=0                 ;POINTER FOLLOWS INSTRUCTION
               ;
               ;         FLOATING POINT FUNCTIONS
               ;
         007074          FNEG=FUNCT+4             ;NEGATE THE FLAC
```

```
        007075          FLOAT=FUNCT+5                ;CONVERT R1 TO A FLT PT VALUE
               ;
                        .GLOBL  FSQT                 ;TAKE THE SQUARE ROOT OF FLAC
               ;
                        .GLOBL  EVAL.X               ;EVALUATE THE EXPRESSION
                        .GLOBL  FPMP                 ;ENTER FLOATING POINT MODE
               ;
        104400          ERROR=104400                 ;BASE FOR ERROR MESSAGES
               ;
               ;
        000000          R0=%0
        000001          R1=R0+1
        000002          R2=R1+1
        000003          R3=R2+1
        000004          R4=R3+1
        000005          R5=R4+1
        000006          SP=R5+1
        000007          PC=SP+1
               ;
                        .GLOBL  FLAC                 ;FLOATING POINT ACCUMULATOR
               ;
                        .GLOBL  FQUAD                ;ENTRY POINT
               ;
 000000 162706 FQUAD:   SUB     #10,SP               ;OPEN 4 WORDS ON THE STACK
        000010
                                                     ; BY USING 4 WORDS, THIS
                                                     ; FUNCTION COULD BE USED
                                                     ; BY BOTH THE SINGLE AND
                                                     ; DOUBLE PRECISION PACKAGE.
000004 000000G         FPMP                         ;TRAP CALL TO INVOKE FLT PT
 000006 007065          .WORD   FPUT+IMMED           ;SAVE A
 000010 000000 A:       .WORD   0,0,0,0              ;FOUR WORDS
 000012 000000
 000014 000000
 000016 000000
 000020 000000G         EVAL.X                       ;GET B
 000022 000000G         FPMP                         ;FLOATING CALL
 000024 007065          .WORD   FPUT+IMMED           ;SAVE B
 000026 000000 B:       .WORD   0,0,0,0              ;B
 000030 000000
 000032 000000
 000034 000000
 000036 000000G         EVAL.X                       ;GET C
 000040 012701          MOV     #4,R1                ;SAVE A FOUR FOR LATER
        000004
 000044 012702          MOV     #A,R2                ;POINT TO "A" FOR IPTR CALLS
        000010'
 000050 000000G         FPMP                         ;FLOATING POINT
 000052 007041          .WORD   FMUL+IPTR            ;MULTIPLY BY "A" (A*C)
 000054 007063          .WORD   FPUT+STACK           ;SAVE IT ON THE STACK
 000056 007075          .WORD   FLOAT                ;FLOAT R1 (R1=4)
 000060 007043          .WORD   FMUL+STACK           ;MULTIPLY BY FOUR
 000062 007063          .WORD   FPUT+STACK           ;PLACE IT BACK (4AC)
 000064 007000          .WORD   FGET+DIRECT          ;RECOVER B
 000066 000026'         .WORD   B
 000070 007040          .WORD   FMUL+DIRECT          ;MULTIPLY AGAIN FOR B↑2
 000072 000026'         .WORD   B
 000074 007023          .WORD   FSUB+STACK           ;B↑2-4AC
 000076 005767          TST     FLAC                 ;SEE IF NEGATIVE NUMBER
        000000G
 000102 100424          BMI     SETERR               ;SET ERROR ?63
```

```
000104 004767          JSR     PC,FSQT              ;PERFORM SQUARE ROOT
       000000G
000110 000000G         FPMP                         ;ENTER FLOATNG POINT
000112 007063          .WORD   FPUT+STACK           ;SAVE FSQT(B↑2-4AC)
000114 000000G         EVAL.X                       ;GET R
000116 012702          MOV     #A,R2                ;POINT AT "A"
       000010'
000122 000000G         FPMP                         ;ENTER FLOATING POINT
000124 007043          .WORD   FMUL+STACK           ;SET CORRECT SIGN
000126 007020          .WORD   FSUB+DIRECT          ;SUBTRACT OUT B
000130 000026'         .WORD   B                    ; (-B)
000132 007063          .WORD   FPUT+STACK           ;SAVE IT ON THE STACK
000134 007001          .WORD   FGET+IPTR            ;GET A
000136 007011          .WORD   FADD+IPTR            ;MULTIPLY BY 2
000140 007061          .WORD   FPUT+IPTR            ;SAVE BACK IN A
000142 007003          .WORD   FGET+STACK           ;RECOVER NUMERATOR
000144 007031          .WORD   FDIV+IPTR            ;DIVIDE BY 2A
000146 062706          ADD     #10,SP               ;FIX UP THE STACK
       000010
000152 000207          RTS     PC                   ;RETURN WITH THE ANSWER
                                                    ; IN THE FLAC
               ;
000154 104777 SETERR:  ERROR+201+63.+63.            ;SEND ERROR 63 (IMMAGINARY
                                                    ; ROOTS!)
               ;
       000001'         .END


SYMBOL TABLE

A        000010R        B        000026R        DIRECT= 000000
ERROR =  104400         EVAL.X= ****** G        FADD  = 007010
FDIV  =  007030         FGET  = 007000          FLAC  = ****** G
FLOAT =  007075         FMUL  = 007040          FNEG  = 007074
FPMP  =  ****** G       FPOW  = 007050          FPUT  = 007060
FQUAD    000000RG       FSQT  = ****** G        FSUB  = 007020
FUNCT =  007070         IMMED = 000005          IPTR  = 000001
PC    =%000007          R0    =%000000          R1    =%000001
R2    =%000002          R3    =%000003          R4    =%000004
R5    =%000005          SETERR  000154R         SP    =%000006
STACK =  000003
. ABS.   000000     000
         000156     001
ERRORS DETECTED: 0
FREE CORE: 9628. WORDS
FQUAD,FQUAD=FQUAD
```

It should be noted that the paper-tape version of the FQUAD function required several equates. This is because the FQUAD function for this version of FOCAL is essentially a patch. An absolute binary tape of that routine would be generated, and loaded following the loading of FOCAL. FOCAL would then be restarted by starting the computer at location 0.

The RT-11 version of the FQUAD function used the availability of the GLOBAL variables supported by the RT-11 LINK program. This version of the FQUAD function must be incorporated into FOCAL by adding the file FQUAD.OBJ to the linking process just prior to the /F switch (see Appendix I).

In addition, the file PUBLIC.MAC must be modified. After the macro symbol FBASE, a FUNCT macro should be added prior to the definition of the symbol FNTABL. Full instructions for this will be found in the listing of PUBLIC.MAC.

It is necessary that the entry point to the FNEW appear in a .GLOBL directive within the FNEW code.

The following is a small FOCAL-11 program which uses the function FQUAD to determine both roots of a quadratic equation specified by the user:

```
C:FOCAL-11,PAPER-TAPE V1

  1.10 A "ENTER THE COEFFICIENTS",A,B,C
  1.20 S A1=FQUAD(A,B,C,1);S A2=FQUAD(A,B,C,-1)
  1.30 T "THE ANSWERS ARE",A1,A2,!;G 1
*G
ENTER THE COEFFICIENTS:1 :-2 :1
THE ANSWERS ARE=    1.0000=    1.0000
ENTER THE COEFFICIENTS:2 :-6 :3
THE ANSWERS ARE=    2.3660=    0.6340
ENTER THE COEFFICIENTS:1 :1 :1

?63 AT  1.20
*
```

6.4 ASYNCHRONOUS I/O PROCESSING (FINT)

It is possible with FOCAL-11 to service interrupts from various devices at the FOCAL language level. In other words, on interrupt from a device, execution of a FOCAL-11 statement can be stopped, control will pass to a specific group of FOCAL-11 instructions which process that interrupt, and once the interrupt is handled, control will return to the interrupted statement where it left off.

To accomplish this the FOCAL Function FINT is used.

```
          X FINT(vector,line/group,priority,CSR,mask)
```

vector	- This is the interrupt vector address for the device.
group	- This is the group or line number of the interrupt handling routine.
priority	- This is the software priority which the interrupt routine is to execute at.
CSR	- Control Status Register address. This location is written to with the value of "mask" at the time an interrupt is detected.
mask	- Since the requested routine does not execute immediately, but rather as soon as FOCAL comes to a convienient place to allow the routine to execute, FOCAL needs a means of disabling further interrupts from being handled. This value (mask) is sent to the "CSR address" specified above before the routine is executed. It is the routine's responsibility to re-enable interrupts via the FX function.

In order to disable interrupt processing for a particular device, i.e. turn it off, specify a zero group value in the call to FINT.

```
X FINT(vector,0)
```

This will also disable the hardware device interrupts as mentioned above.

Whenever FOCAL returns to its Command/Input mode, all interrupt scheduling is terminated. This can occur if the user either encounters an error (?XX) or issues a QUIT command.

If the user is using the Foreground/Background version of the RT-11 monitor, and a Foreground program is loaded, FOCAL "protects" the device vectors used in the FINT function by informing the monitor of its intended use. This is done in order to prevent possible conflict with the Foreground program. If a program uses a device under these conditions, and the monitor informs FOCAL that the vector has been previously protected, FOCAL will issue an error message (?39).

EXAMPLE

In RT-11, FOCAL-11's asynchronous I/O capability might be used as follows:

Let us suppose that our hypothetical installation is using FOCAL-11 to obtain data from an experiment by monitoring the analog output of the experiment's measuring device via the AR-11 analog-to-digital converter. The data rate is very slow, requiring 1000 points sampled at the rate of one every 10 seconds. Rather than tie up the computer for the time necessary to take these samples (2-1/2 hours), the data is sampled asynchronously every ten seconds and the values saved on paper tape, allowing another program to run during the time spent between the data points.

A second terminal has been connected to the system. The device registers are at 176500, and the vector address is 300.

The user wishes to obtain the value of a channel (0-7) whenever the respective key is struck on that terminal. The following section of a FOCAL program will accompish this:

```
    .
    .
    .
X FINT(@300,3,4,@176500,0);C-HOOK UP TERMINAL
X FX(-2,@176500,@100);C-ENABLE INTERRUPTS
    .
    .
    .
3.05 C - ACCEPT TERMINAL CHARACTER AND VALIDATE IT
3.10 S &=FX(0,@176502,@177)-@60;I(&)3.98;I(&-7)3.15,3.99,3.98
3.15 S &=FADC(&);S Z1=1E5; C - PERFORM OUTPUT CONVERSION
3.20 S Z2=@60
3.25 I (&-Z1)3.3;S &=&-Z1;S Z2=Z2+1;G 3.25;C - CALCULATE DIGIT
3.30 X FX(-2,@176506,Z2);DO 3.95;S Z1=Z1/10;I(Z1-1)3.35,3.35,3.2
3.35 S &=&+@60;X FX(-2,@176506,&);DO 3.95
3.95 I (-FX(1,@176504))3.95;R;C - WAIT FOR OUTPUT TO COMPLETE
3.98 X FX(-2,@176506,@15);DO 3.95;X FX(-2,@176506,@12);DO 3.95
3.99 X FX(-2,@176500,@100); RETURN; C - ENABLE INTERRUPTS AND EXIT
```

6.5 ERROR HANDLING IN FOCAL (FERR)

If the program has requested that errors be handled by a user written routine, control will pass to the desired line or group upon the occurrence of an error. This is set up by the execution of the FERR function.

```
XECUTE FERR(3)
```

This will cause group 3 to be entered on the occurrence of an error. The error number is stored in the variable "&". If the user wishes the program to report the error, simply return from the subroutine and error diagnostics will be handled correctly. However, if the FERR function is re-issued with a non-zero argument, returning from the subroutine will cause the program to continue at the next FOCAL COMMAND.

To disable this feature, simply execute the FERR function with a zero argument.

6.6 SCHEDULING ROUTINES BY TIME (FQUE)

It is possible to schedule FOCAL statements to be executed at specified times. These are processed in the same manner as the interrupt requests mentioned above, i.e., the routine will be run at the first convenient point in the user's program after the specified time has elapsed (usually only a few milliseconds). To do this, the following statement may be used:

```
SET ID=FQUE(count,group,time,delay,priority)
```

ID	- This is used when canceling a request. It is the user's responsibility to save this information.
count	- This is the number of times the routine is to be executed (0 will cancel the routine. see below.).
group	- Line or group number to be executed
time	- Time interval in seconds or ticks. FOCAL parameter eleven is used to determine the unit of time. If it is zero, then seconds will be used. The range of values allowed for this FQUE parameter is from +1 to +32,767.
delay	- Time delay in seconds until the first of the scheduled routines is executed. Zero will cause immediate execution.
pri	- Software priority level

To cancel a request before it executes:

```
XECUTE FQUE(0,id,group)
```

where "id" is the value returned when "group" was scheduled.

It is imperative that the system clock be running if the user intends to use the FQUE function. If the clock is disabled, no routines will ever be scheduled.

6.7 GENERAL NOTES ON SCHEDULING ROUTINES IN FOCAL

FOCAL treats both scheduling on time (FQUE function) and scheduling by device interrupt (FINT function) in the same manner. There are certain concepts which are common to both methods of scheduling FOCAL routines which the user should be familiar with in order to effectively use these functions.

Software Priority

The concept of Software Priority enables the user to specify the relative importance of a particular routine (a line or group of FOCAL STATEMENTs) which is to be run. Whenever a program is requested to be interrupted, the current software priority is checked to see if the requesting routine is higher in priority than the one currently running. The range of possible priorities is from zero (0) to seven (7) inclusive. The initial program priority is -1, and any scheduled subroutine to be run, will be. If another routine becomes ready to be executed, it will begin execution immediately only if it has a higher priority than the routine currently running. If it does not, it must wait until there are no other routines with a higher priority which are able to be executed. Only at that time will the new routine will begin execution.

As an example, suppose it is desired to use FOCAL to sample the AR-11 A/D converter 1000 times on channel 3 at a rate of one sampling each ten seconds. These points are to be saved in a file called DATA.FCL on the system device for later use.

First the size of the file must be determined by using the largest subscript:

999 / 64 = 16 blocks (when truncated)

now the main program must open the file:

```
     1.1 LIBRARY MAKE 7,DATA[16]/Z/V:DATA(0)
```

and schedule the function:

```
     1.2 COUNT=0;COMMENT: ITERATION COUNT
     1.3 SET ID=FQUE(1000,9,10,0,7)
         .
         .
         .
     9.1 SET DATA(COUNT)=FADC(3);SET COUNT=COUNT+1
     9.2 IF(COUNT-1000)9.3;LIBRARY CLOSE 7; C - CLOSE FILE WHEN DONE
     9.3 RETURN
```

Program Considerations

The method by which FOCAL actually executes the scheduled routine is quite simple. It is performed by issuing a DO command to the routine requiring execution at the completion of the current FOCAL command in the user's program.

For this reason, a program which uses scheduling must be careful to refrain from using any command which may take a long time to complete. For instance, if a routine is scheduled to execute and the user program is currently performing an ASK command, the scheduled routine will not be executed until the user responds to the ASK command at the terminal, and allows the command to be completed. Certain LIBRARY commands may also take a long time to be completed. This is especially true if the command is being executed to a slow device.

Another consideration which the programmer should be aware of is that in order for a scheduled routine to execute, the user program must be running. This means that once a QUIT command is encountered, no further scheduling may take place. One method of insuring that this never happens is to place the program into an infinite loop at its conclusion, instead of issuing a QUIT command. This is quite common in programs which maintain several time scheduled events. In this instance, the user program would issue the required FQUE functions to schedule the tasks to be performed. An additional routine is then scheduled to occur at a time after the last of the previously scheduled routines should have completed. This routine is used to perform any necessary clean-up (close LIBRARY files, for instance), and then issue a QUIT command. The program could then continue to execute. At its conclusion, it would then place itself into an infinite loop by using a GO command directed to itself. In this manner, the program would allow all scheduled routines to complete, and the program will QUIT after all of the scheduled routines have been completed.

Terminating FOCAL Scheduling

Both the FINT and the FQUE functions allow the user to cancel further scheduling of their routines. In addition, whenever FOCAL returns to its Command/Input mode via an error or by a QUIT command, all scheduling is terminated.

Concurrent Routine Scheduling

FOCAL is provided with a default maximum of eight (8) "slots" in which to store all the information necessary to schedule a task. This means that a total of eight routines may be scheduled. It is possible to have all eight routines to be scheduled by time, or all eight by device interrupt, or any combination of time and interrupt routines, as long as the combined total does not exceed eight.

This number of "slots" may be altered by modifying the FOCAL source file PUBLIC.MAC. The symbol MAXTSK should be changed to have the maximum number of concurrently scheduled routines. This file should then be reassembled and linked into FOCAL as described in Appendix I.

CHAPTER 7

RT-11 FOCAL FILE CAPABILITIES

(LIBRARY Command)

The LIBRARY command is used in FOCAL in order to access the RT-11 file structure. This command has already been treated somewhat in Section 3.5 of this manual.

Due to the power and complexity which this command is capable of attaining, Chapter 7 is devoted to defining the structure of the LIBRARY command, and explaining the various forms which the command may take.

7.1 GENERAL COMMAND FORMAT

All LIBRARY commands subscribe to a general command format. Not every command will require every field, but if a field is required by a command, it will be in the same relative position, and follow the same rules as any other LIBRARY command which uses it.

The general form of the LIBRARY command is shown below. Angle brackets denote specific argument fields.

```
     LIBRARY <cmd> <file #,><file specification><switches><args>
```

LIBRARY	This is the FOCAL "LIBRARY" command. "LIBRARY" may be abbreviated to the letter "L". This part of the command must be followed by at least one space.
<cmd>	This is the LIBRARY sub-command. This field specifies which of the library operations are to be performed. This field may be abbreviated to a single character and must be followed by at least one space.
<file #,>	This is the file number required by some of the library commands. If this number is required, it must be placed in this position in the library command and terminated by a comma. The file number must be in the range zero (0) to seven (7).
<file specification>	This field is used by some library commands to define the file on which the operation is to be performed. If no switches are

	supplied, a space should be placed after the specification if arguments are to follow.
<switches>	Some commands allow the user to further specify the types of operations to be performed on the file specified. This field should be terminated by a space. If more than one switch is specified, no space should appear between them.
<args>	This is the field for the arguments, if any, of the particular LIBRARY command.

7.2 LIBRARY COMMAND FIELD SYNTAX

As specified above, the LIBRARY command is formed by placing several fields of information together in a specific order.

The rules for each of these fields defined in section 7.1 are explained below.

7.2.1 LIBRARY Field

The FOCAL command LIBRARY is specified by a string of one or more characters beginning with the letter "L" followed by at least one space. This is the same convention used with all FOCAL commands.

7.2.2 <cmd> Field

The <cmd> field is used to define the sub-command to the LIBRARY routines. These commands may be abbreviated to one character. This field must be terminated by a space (blank) if any additional information for the command is to follow it. The possible commands are:

Command	Abbreviation	Function
OPEN	O	Opens an RT-11 file for use by a FOCAL program.
MAKE	M	Creates a new RT-11 file for use by a FOCAL program.
INPUT	I	Opens an existing RT-11 file for use by a FOCAL program.
GET	G	Causes the program to be loaded from a RT-11 file.
RUN	R	ERASEs all variables and program statements, then "GET"s the program specified, and a "GO" is then performed for the new program. No files are closed by this operation and they remain open when the new program starts.
NEXT	N	This command erases only the program from memory, GET"s the specified program, and a

		"GO" is then executed. Variables will be left intact unless the new program is to large to fit in the remaining space in which case they will be deleted. All files are left intact, and remain in the same state as before the execution of the command.
SAVE	S	This command causes a copy of the entire program currently in memory to a be written to specified file for later use. It may be retrieved by a LIBRARY GET, RUN, or NEXT command.
CLOSE	C	This command terminates all activity for a specified file number. New files are made permanent and all changes made to old files are completed.
WRITE	W	This command allows the program to WRITE statements to an previously opened file.
TYPE	T	This command TYPEs output to a previously opened file.
ASK	A	This command allows the program to ASK from an opened file.
DELETE	D	This command allows the user to delete an RT-11 file from a mass storage device.

7.2.3 <file #,> Field

This field is used to identify an opened file to the library routines.

A user is allowed to open up to eight files at a time. The values of the file number may range from 0 to 7.

This field can contain a constant (number), a variable, or an expression. The value specified is truncated (made into an integer with the fractional part discarded) and is checked to make sure that it is within the range allowed. When a file is opened, the accompanying file number is associated with the specified file. This association remains in effect until a LIBRARY CLOSE has been issued or until an error message has been printed.

This field must be terminated by a comma (,) if any additional information is to be included in the command string.

7.2.4 <file specification> Field

The file specification field is used to define the name of the desired file, as well as the device to be used and the size of the file.

The format of this field is:

```
DEVICE:NAME.EXT[SIZE]
```

where:

DEVICE:	is an optional RT-11 type device specification. If no device is specified, the system device is assumed. Some of the more common device names are shown below. It should be noted that all device names terminate with a colon.

NAME	DEVICE
SY:	System device
DTn:	Dectape unit "n"
LP:	Line printer
PP:	High speed paper tape punch

NAME	is the name of the file to be used. This should always be specified, as no default is assumed.
.EXT	is the qualifying extension field for the file. If this part of the specification field is left blank, ".FCL" is assumed.
[SIZE]	is the optional parameter used to specify the size of the file to be created in 256 word blocks. The value inside the brackets must be a constant. (Expressions or variables are not allowed without quotes. See the section on quotes below.). The following table can be used as a guide to the file size needed to store variables in the VIRTUAL FILE mode:

Variables / 256 word block	
Double Precision:	64
Single Precision:	128
Integer:	256
Byte (character):	512

A feature of tne RT-11 version of FOCAL-11 is to allow an expression to be incorporated anywhere in the file specification field. This feature is not available in the 8K version of FOCAL.

This is performed by enclosing the expression in single quotation marks ('). The result is that the expression is evaluated, and then the result is truncated to an integer. Negative numbers are made positive by taking the absolute value of the result.

This value is then converted into a string of ASCII characters (without any leading zeroes) and inserted into the command in place of the expression.

For example:

```
*SET M=1;SET K=5;SET L=10.1
*LIBRARY MAKE M,TEST'M*K*L'.DAT['K']
```

is the same as:

```
*LIBRARY MAKE 1,TEST50.DAT[5]
```

since M*K*L=1*5*10.1=50.5=50

If there is more information to be entered for the current command, and the file specification is not followed by switches, the field should be followed by a blank character.

7.2.5 <switches> Field

The switches field is used on the OPEN, MAKE, and INPUT commands to further specify the type of operations to be performed on the file. Usually switches are only used on virtual files. A switch is specified by a slash character (/) followed by a single character. No spaces (blank character) may be placed in front of the slash character.

If a switch requires an argument, this is specified by placing a colon after the switch character (i.e. the character following the slash), and following this by the argument.

The possible switches and their meanings appear below. The 8K version of RT-11 FOCAL does not support the /D, /F, /X, /I, /B, and the /T switches. If any of these switches are specified in the 8K version of RT-11 FOCAL, they will be ignored. Only the default formats (/D for double precision, and /F for single precision) will be used.

Switch	Argument?	function
/D	no	File is to be used to store variables in double precision format.
/F	no	File is to be used to store variables in single precision floating point format.
/X	no	File is to be used to store variables as a 16-bit positive magnitude. (range is from 0 to 65534 inclusive) Negative numbers are made positive by taking their absolute value.
/I	no	File is to be used to store variables in standard signed integer format. (-32768 to 32768 inclusive)
/B	no	File is to be used to store integer values in the range -128 to 127 inclusive (Byte).
/T	no	The file is used to store character data (text). (Range of 0 to 127)
/Z	no	The newly created file is to be zeroed (made empty) before use. This switch should only be used if a file is created with a specific length specified (using [n]). Otherwise, all the space allocated to the file will be used. (This may be one half the the largest area on the device specified.) If this switch is specified when no new file is created, it will be ignored.
/V	yes	The file is to be used as a virtual file. The argument is to be a model of the type of

variable used to represent the file. For example:

```
/V:A(0)
```

would assume that any time a reference is made to A(x), the value will be stored in this file at location x.

If additional information is to be placed in the command, the switch field should be terminated by a blank character.

7.2.6 <args> Field

The argument field is always the last section of a LIBRARY command. In general, the format for this field follows that of the equivalent FOCAL command. The LIBRARY RUN and NEXT commands use arguments similar to those of the FOCAL "GO" command.

For example:

```
LIBRARY TYPE IFILE,"THIS IS A LINE.",!
```

This will send the same character string to file number "IFILE" that:

```
TYPE "THIS IS A LINE.",!
```

would send to a terminal.

```
LIBRARY WRITE 3,5
```

will write group 5 of a FOCAL program to file 3.

7.3 LIBRARY COMMANDS

7.3.1 LIBRARY OPEN Command

The LIBRARY OPEN command is a general purpose command to prepare a program to use a file structured device. This command first checks to see if the file number specified is currently in use. If it is, an error (?37) is given, and the command is aborted. If the file number is not in use, FOCAL determines if the file specified already exists. If it does, that file is used for all further operations which reference that file number.

It should be noted that if an existing file is used, only the space currently allocated to the file may be used. If a user attempts to place more data into the file than it originally contained, an error (?33) will result.

If the file does not exist, a new file will be created. This file will not become permanent on the disk until a LIBRARY CLOSE command is issued. This frees the user from having to decide if a file already exists or not, and having to use special LIBRARY commands for each case.

The format for the LIBRARY OPEN command is shown below.

```
        LIBRARY OPEN <file #,><file specification><switches>
```

As an example, let us assume that a file is to be used as a virtual file by a program. It is desired that if no file already exists, all of the values be initialize to zero. If the file does currently exist, the current values are to be used. The following section of a FOCAL program would accomplish this:

```
     1.10 LIBRARY OPEN 1,FILE.DAT[5]/Z/V:DATA(0)
     1.20 FOR I=0,99;SET DATA(I)=DATA(I)+10.0
     1.30 LIBRARY CLOSE 1
```

The first time that this program is run, and if no file "SY:FILE.DAT" exists, the file will be created, and set to xero (/Z switch). Successive executions of this program will add 10 to each of the first 100 elements of the file.

7.3.2 LIBRARY INPUT Command

The LIBRARY INPUT command is identical to the LIBRARY OPEN command with the exception that if the file does not already exist, no new file is created, and an error (?34) is returned.

7.3.3 LIBRARY MAKE Command

The LIBRARY MAKE command is identical to the LIBRARY OPEN command with the exception that a new file is always created, regardless of whether or not a file already exists.

7.3.4 LIBRARY CLOSE Command

This command releases a file and makes all modifications to it permanent.

The user is warned that if an existing file was opened, and the file was modified, unless the file is closed, the file may be left with only part of the modification actually in the file. If a newly created file is not closed, the file will be deleted upon completion of the session with FOCAL (i.e. striking ↑C). If a file number is specified only that file will be closed. If no number is specified, or "ALL" is used in place of a file number, all files will be closed.

The format of the LIBRARY CLOSE command is:

```
                       LIBRARY CLOSE <file #>
                                 or
                            LIBRARY CLOSE
```

7.3.5 LIBRARY GET Command

The LIBRARY GET command is used to temporarily direct FOCAL to accept statements from the file specified. No statements currently stored by FOCAL will be erased unless a new statement from the input file replaces it. Variables are not altered unless the program becomes too large, and then the variables will be erased.

The form of the LIBRARY GET command is:

```
                    LIBRARY GET <file specification>
```

For instance, if a user wished to load a program called "PROG" into memory, and ready it for execution via the GO command, the following command caould be used:

```
     *ERASE ALL
     *LIBRARY GET PROG
```

7.3.6 LIBRARY RUN Command

The LIBRARY RUN command loads and starts a specified program from an RT-11 file. Optionally, an argument may be specified giving the statement number to begin execution. Otherwise, a GO is performed, and execution will begin with the first statement. The form of the LIBRARY RUN command is as follows:

```
                LIBRARY RUN <file specification><arg>
```

For example:

```
                      LIBRARY RUN DT:COMPUT,2.3
```

would erase both text and variables, then read the program from DECtape called COMPUT.FCL into memory. This program would then be started automatically at line 2.3. If the user wants to start the program from the first statement of the program, then the following command would be used:

```
                         LIBRARY RUN DT:COMPUT
```

7.3.7 LIBRARY NEXT Command

The LIBRARY NEXT command is equivalent to the LIBRARY RUN command, with the exception that the variables are not erased. As in the LIBRARY RUN command, an argument may be specified to direct program execution once the program is stored in memory.

Variables are preserved between programs, unless a program is too large to allow the variable area to remain intact. In the case the variables are erased.

When the called program gains execution, the special variable "&" will contain the line number following the LIBRARY NEXT command in the calling program.

The LIBRARY NEXT command can not be used within a DO or FOR series of commands.

As an example, below are three programs which call each other via the LIBRARY NEXT command. The first (CHAIN.FCL) calls the other two which alter a variable and then return to the calling program.

```
            CHAIN.FCL:

     C:FOCAL-11S V1  (RT-11) 06-NOV-74
       1.10 C LIBRARY NEXT TEST PROGRAMS
       1.20 C  CHAIN.FCL - MAIN PROGRAM
```

```
  1.30 C  CHAIN1.FCL - ROUTINE 1
  1.40 C  CHAIN2.FCL - ROUTINE 2
  1.50 S N=1 ; C-CALL ROUTINE 1
  1.60 S I=100
  1.65 T !"CALLING CHAIN1.FCL WITH I = ",↑I(I),!
  1.70 L N CHAIN'N'
  1.80 T !"RETURNED FROM CHAIN1.FCL AND I = ",↑I(I),!
  2.10 S I=I+100; S N=N+1
  2.20 T !"CALLING CHAIN2.FCL WITH I = ",↑I(I),!
  2.30 L N CHAIN'N'
  2.40 T !"RETURNED FROM CHAIN2.FCL WITH I = ",↑I(I),!
  2.50 QUIT

       CHAIN1.FCL:

C:FOCAL-11S V1  (RT-11) 06-NOV-74
  1.10 C CHAIN1.FCL
  1.20 C
  1.30 T !"CHAIN1.FCL CALLED. WILL RETURN TO LINE ",%4.02,&
  1.35 S I=I+66
  1.40 L N CHAIN,&

       CHAIN2.FCL:

C:FOCAL-11S V1  (RT-11) 06-NOV-74
  1.10 C CHAIN2.FCL
  1.20 C
  1.30 T !"CHAIN2.FCL CALLED. RETURN WILL BE TO ",&,!
  1.40 S I=I+33
  1.50 L N CHAIN,&

       EXECUTION:

*L R CHAIN
CALLING CHAIN1.FCL WITH I = 100
CHAIN1.FCL CALLED. WILL RETURN TO LINE =  1.80
RETURNED FROM CHAIN1.FCL AND I = 166
CALLING CHAIN2.FCL WITH I = 266
CHAIN2.FCL CALLED. RETURN WILL BE TO =  2.40
RETURNED FROM CHAIN2.FCL WITH I = 29 [
*
```

7.3.8 LIBRARY SAVE Command

The LIBRARY SAVE command is used to save a program in an RT-11 file.

This command creates a new file with the name specified and performs a "WRITE ALL" to the file. The file is then closed.

The only error possible is if there is insufficient space on the specified device for the entire program.

The format of this command is:

```
LIBRARY SAVE <file specification>
```

For example, if the program currently in memory was needed to be saved on the system device, the following command would be used:

```
LIBRARY SAVE PROG1
```

This would save the program in a file called PROG1.FCL on the system device.

7.3.9 LIBRARY WRITE Command

The LIBRARY WRITE command performs the function of the FOCAL WRITE command, but directs the associated output to the specified file. The file must have been already opened by a LIBRARY OPEN, MAKE, or INPUT command.

An optional argument may be specified. This argument is in the same form as the one for the WRITE command in FOCAL. This argument is either a statement number, group number, or "ALL".

The format for this command is:

```
                    LIBRARY WRITE <file #><,arg>
```

If, for example, the user wished to make a file containing just section 3 of the current program, the following commands could be used:

```
    *LIBRARY MAKE 1,SEC3
    *LIBRARY WRITE 1,3
    *LIBRARY CLOSE 1
    *
```

This would create a new file by the name of "SEC3.FCL"" on the system device which would only contain section 3 of the program currently in memory.

7.3.10 LIBRARY TYPE Command

The LIBRARY TYPE command was provided to allow the user to direct the output of a TYPE command to an RT-11 file which has been opened with a LIBRARY OPEN, MAKE, or INPUT command. Any argument legal for a FOCAL TYPE command may be used in a LIBRARY TYPE command.

The format of the LIBRARY TYPE command is as follows:

```
                    LIBRARY TYPE <file #><,args>
```

For example, if a file called OUTPUT.LST was to be created on the system device for later printing, the following program would generate a table of random numbers in that file.

```
    1.10 LIBRARY MAKE 0,OUTPUT.LST
    1.20 FOR I=1,10;LIBRARY TYPE 0,!;DO 2
    1.30 LIBRARY CLOSE
    1.40 QUIT

    2.10 FOR J=1,10;LIBRARY TYPE 0,FRAN(),"   "
    2.20 RETURN
```

At the conclusion of this program's execution, PIP could be run to print the file on the line printer.

7.3.11 LIBRARY ASK Command

The LIBRARY ASK command allows the user to accept input from an RT-11 file which has been previously opened by a LIBRARY OPEN, MAKE, or INPUT command.

The command follows the same formats as the FOCAL ASK command. If character strings are placed in the LIBRARY ASK command, they will be ignored.

The format of the LIBRARY ASK command is shown below.

```
LIBRARY ASK <file #><,args>
```

For example, if a file called INPUT.DAT had been previously created either by using a LIBRARY TYPE in another program, or was prepared using a text editor, a program could be written in FOCAL to read this numeric data.

```
     1.10 LIBRARY INPUT 1,INPUT.DAT
     1.20 COMMENT: SUM THE FIRST 100 VALUES AND PRINT THE RESULT
     1.30 SET SUM=0
     1.40 FOR I=1,100;LIBRARY ASK 1,X;SET SUM=SUM+X
     1.50 TYPE !"THE SUM IS ",SUM,!
     1.60 QUIT
```

7.3.12 LIBRARY DELETE Command

The LIBRARY DELETE command is used to remove a specified file from a RT-11 structure mass storage device such as a disk. If the file does not exist, an error (?35) will be given.

The format of the LIBRARY DELETE command is:

```
LIBRARY DELETE <file specification>
```

For example, if the user wished to delete a file called "DATA.DAT" on DECtape unit 1, the following command would be given:

```
*LIBRARY DELETE DT1:DATA.DAT
```

7.4 VIRTUAL FILES

In many instances, it would be helpful if the user were able to manipulate large blocks of data in an array (list or table of values), but the memory size of the user's computer is insufficient to store all of the information. For this reason, the concept of virtual files was implemented in FOCAL. This allows FOCAL to use a file on a random access device, such as disk or DECtape, to store the contents of user defined arrays. A program on a small machine is now able to utilize vast amounts of array space.

In order to utilize virtual files, the user specifys a subscripted variable in a LIBRARY OPEN, MAKE, or INPUT command. This associates the subscripted variable with a file.

Then, whenever the program references the subscripted variable specified in the LIBRARY command, FOCAL will automatically use the file as the storage area. This means that a program can be written as if it were to be run on a large machine, and then, without major changes, be altered to run on a small machine. Additionally, this makes it very easy to write several programs which reference the same file, allowing the values of the array to be passed from program to program. This is the fastest, and most efficient means of saving data for later use, since the file contains the data in FOCAL's internal format and conversion to printable characters is not required.

To specify an array as a virtual file, a standard LIBRARY OPEN MAKE, or INPUT is used, with the "V" switch. For example:

```
LIBRARY MAKE 1,ARRAY.DAT[10]/Z/V:A(0)
```

This will cause the creation of a file on the system device, ten blocks in length, named ARRAY.DAT. The file will be initially filled with zeroes. Each time the subscripted variable "A" is referenced, FOCAL will refer to the file ARRAY.DAT.

The range of subscript permissible in a virtual array is 0 through 32,767. Subscripts within the range -1 to -32,768 may be used, but are treated as 65,536 plus the subscript value. This would require vast amounts of storage area to be wasted if not all the positive subscripts were in use.

In order to determine the size requirements for a virtual array, the largest suscript value should be used with the following table.

Variable type	# variables / block
(D) Double precision	64
(F) Single precision	128
(I,X) Integer	256
(B,T) Byte	512

For instance, if a virtual array with the largest subscript of 1000 in single precision format was used, this would take:

BLOCKS = MAX SUBSCRIPT / # VARIABLES PER BLOCK

7.8 = 1000/128

requiring a file of 8 blocks in length.

As an example of the power of virtual files, suppose an inventory needed to be kept on a series of items ranging in number from 0 to 99. Each time a transaction was to be made, a separate record was to be kept. At the end of the day, these transaction files were to be merged with the master inventory file, and a list of the current inventory was required. This is a fairly straight forward type of problem, and one which is fairly typical of a user's needs.

The following series of three programs perform this task using the virtual file concept. Many other features of RT-11 FOCAL are demonstrated by this example.

```
C:FOCAL-11S V1  (RT-11) 13-NOV-74
  1.10 C - INVENTORY PROGRAM 1
  1.20 C   *** UPDATE PROGRAM - GENERATE TRANSACTION FILE ***
  1.30 X FERR(2)
  1.40 SET IFILE=1
  1.50 L I 1,FILE'IFILE'.TRN
  1.60 L C 1;SET IFILE=IFILE+1;G 1.5
  2.10 X FPRM(3,3);T !!"TRANSACTION ",↑I(IFILE),!!
  2.15 L C
  2.20 L M 1,FILE'IFILE'.TRN[5]/I/Z/V:DATA(0)
  2.30 T "ENTER TRANSACTIONS BY ITEM,"
  2.31 T " FOLLOWED BY NUMBER OF UNITS:"!!
  2.35 T "ENTER A -1 FOR THE ITEM NUMBER TO EXIT"!!
  2.40 A "ITEM #:",I," # OF UNITS:",J
  2.50 I (I) 2.6;SET DATA(I)=DATA(I)+J;T !;G 2.4
  2.60 L C 1;T !"TRANSACTION #",↑I(IFILE)," COMPLETED."!!!;Q

C:FOCAL-11S V1  (RT-11) 13-NOV-74
  1.10 C - INVENTORY PROGRAM 2
  1.20 C    * MERGE TRANSACTION FILES INTO MASTER AND RUN REPORT *
  1.30 X FERR(2);SET IFILE=1
  1.40 L O 2,MASTER.FIL[5]/I/Z/V:MASTER(0)
  1.45 SET SWITCH=0
  1.50 L I 1,FILE'IFILE'.TRN/I/V:TRANS(0)
  1.55 IF (SWITCH) 1.8;C
  1.60 FOR I=0,99;SET MASTER(I)=MASTER(I)+TRANS(I)
  1.70 L C 1;L D FILE'IFILE'.TRN;SET IFILE=IFILE+1;GO 1.5
  1.80 L R INVEN3
  2.10 LIBRARY CLOSE
  2.20 T !!↑I(IFILE-1)," TRANSACTIONS WERE MERGED."!
  2.30 T "REPORT PROGRAM WILL NOW BE RUN."!!
  2.35 SET SWITCH=-1
  2.40 X FERR(2);R

C:FOCAL-11S V1  (RT-11) 13-NOV-74
  1.10 C - INVENTORY PROGRAM 3
  1.20 C    *** REPORT GENERATOR ***
  1.25 X FPRM(3,3)
  1.30 L I 1,MASTER.FIL/I/V:MASTER(0)
  1.40 O L;X FCHR(@14);T !"MASTER INVENTORY LISTING"!!!
  1.50 T "ITEM  UNITS     ITEM  UNITS"!!
  1.60 F I=0,49;DO 1.65
  1.62 GO 1.68
  1.65 T !" ",%2,I," ",%5,MA(I),"     ",%2,I+50,"  ",%5,MA(I+50)
  1.68 T !!!!!
  1.70 LIBRARY CLOSE;OPERATE T
  1.80 QUIT

*L R INVEN1

TRANSACTION 1

ENTER TRANSACTIONS BY ITEM, FOLLOWED BY NUMBER OF UNITS:

ENTER A -1 FOR THE ITEM NUMBER TO EXIT

ITEM #:1  # OF UNITS:4
ITEM #:2  # OF UNITS:89
ITEM #:73  # OF UNITS:5
ITEM #:14  # OF UNITS:7
ITEM #:66  # OF UNITS:8
ITEM #:-1  # OF UNITS:0
TRANSACTION #1 COMPLETED.

*L R INVEN1
```

```
TRANSACTION 2

ENTER TRANSACTIONS BY ITEM, FOLLOWED BY NUMBER OF UNITS:

ENTER A -1 FOR THE ITEM NUMBER TO EXIT

ITEM #:6   # OF UNITS:2
ITEM #:1   # OF UNITS:-3
ITEM #:99   # OF UNITS:6
ITEM #:50   # OF UNITS:3
ITEM #:-1   # OF UNITS:0
TRANSACTION #2 COMPLETED.

*L R INVEN2

2 TRANSACTIONS WERE MERGED.
REPORT PROGRAM WILL NOW BE RUN.
```

Output from line printer:

```
MASTER INVENTORY LISTING

ITEM  UNITS     ITEM  UNITS

   0      0      50      3
   1      1      51      0
   2     89      52      0
   3      0      53      0
   4      0      54      0
   5      0      55      0
   6      2      56      0
   7      0      57      0
   8      0      58      0
   9      0      59      0
  10      0      60      0
  11      0      61      0
  12      0      62      0
  13      0      63      0
  14      7      64      0
  15      0      65      0
  16      0      66      8
  17      0      67      0
  18      0      68      0
  19      0      69      0
  20      0      70      0
  21      0      71      0
  22      0      72      0
  23      0      73      5
  24      0      74      0
  25      0      75      0
  26      0      76      0
  27      0      77      0
  28      0      78      0
  29      0      79      0
  30      0      80      0
  31      0      81      0
  32      0      82      0
  33      0      83      0
  34      0      84      0
  35      0      85      0
  36      0      86      0
```

37	0	87	0
38	0	88	0
39	0	89	0
40	0	90	0
41	0	91	0
42	0	92	0
43	0	93	0
44	0	94	0
45	0	95	0
46	0	96	0
47	0	97	0
48	0	98	0
49	0	99	6

CHAPTER 8

AR11/LPS AND DR11-K FUNCTIONS

8.1 INTRODUCTION

NOTE

The term AR11/LPS refers to either the AR11 or the LPS.

The AR11 (Analog Real-Time Interface) and the LPS (Laboratory Peripheral System) are real-time interface devices adaptable to a wide variety of applications including biomedical research, analytical instrumentation, psychological research, data collection, monitoring, data logging, industrial testing, engineering, and technical education. Each consists of a multi-channel A/D converter, a programmable real-time clock, and two D/A converters which may be used to control a CRT. The LPS can have up to 64 A/D channels. Its digital values are in the range 0 to 4095 (12 bits). It has two Schmitt triggers. The AR11 has 16 A/D channels. Its digital values are in the range 0 to 1023. It has one external event line. The LPS also includes a numeric display consisting of six LED (Light Emmitting Diode) matrices.

The DR11-K is a general purpose digital input-output interface. Under program control it can parallel transfer up to sixteen bits of data betweeen a PDP-11 computer and an external device or another DR11-K.

The PDP-11 FOCAL functions described in this chapter allow PDP-11 FOCAL to make full use of these devices.

The FTIC and FDLY functions allow access to the AR11/LPS programmable clock. Using FTIC the FOCAL program can start or stop the clock, set its rate to any value between 10000 ticks per second and one tick every 2.55 seconds on the AR11 and every 640 seconds on the LPS, and read and set the value of a timer which maintains a count of the number of ticks that have occurred. Using FDLY the program can wait a specified number of ticks before continuing the FOCAL program, or continually execute a given function for a specific amount of time or until the function returns a positive value.

FTOI, the time of interrupt function, returns to the user the exact AR11/LPS time of an interrupt scheduled by the FINT function. This is useful when the exact time that an event occurs constitutes an important experimental parameter.

The FSAM function allows the FOCAL program to sample the A/D channels. The program can read the current value of a specified channel directly or can create a ring buffer into which it may cause asynchronous sampling of any number of values at the rate at which the AR11/LPS clock is ticking or upon the occurrence of external events. Samples may be taken from a single channel or from a series of channels. This form of sampling is performed while the FOCAL program is executing. While sampling proceeds the FOCAL program can read the samples taken or perform other operations. Under FOCAL/RT-11 the program can also cause data to be written to an RT-11 file directly from the ring buffer allowing for high data aquisition and storage rates. If the file written to is a virtual file, the program can access the data already loaded into the file even as it is loaded with additional data.

The FDMA function, which applies only to the LPS, allows for direct memory access sampling in any of several modes.

The FCRT function gives the FOCAL program powerful display capabilities. It allows the program to create an addressable display file of arbitrary length, load it with graphics instructions, and cause display on a CRT at a specified level of intensity. The graphics instructions available are instructions to display an absolute (or fixed) point, a relative (or movable) point, points plotted along the x or y axis, ASCII characters, and some control characters. The display file's addressability enables the FOCAL program to alter the screen location of graphic data causing the display to move, to alter characters on display, continually add to or delete from the display file, and, in general, manipulate the display.

The FFRM function allows RT-11 FOCAL to save the display file in an RT-11 file and to retrieve it at a later time. Each RT-11 file can contain a single display file or a series of separate frames. The FLED function allows FOCAL to load the numeric display on an LPS with a floating point value.

FFNS is a function which executes sequentially the set of functions specified in its argument list. This greatly decreases the waiting time between function executions that sequential executions using the Xecute command would entail. By specifying a wait function as one or more of its arguments, the FOCAL program can cause exact time delays between execution of functions.

FBIT allows the FOCAL program to perform various logical operations between 16-bit quantities, and to set, clear, and complement individual bits. It performs AND, OR, and XOR (exclusive or), and converts bit numbers to 16-bit values with the corresponding bits set. It is especially useful in conjunction with the UNIBUS function, FX, for accessing device registers, particularly those of the DR11-K.

All these functions except the two that require an RT-11 file will run under both RT-11 and Paper Tape FOCAL. The user may create a version of FOCAL that includes all or only some of these functions.

The next sections describe the functions in detail giving examples of their use. Appendix M, 'ASSEMBLING AND LOADING FOCAL LAB EXTENSIONS', describes how to create FOCAL for the AR11/LPS and DR11-K from the FOCAL language files and the function file, LABFNS.MAC.

8.2 THE FUNCTIONS

The functions described in this chapter offer complete access to the features of the AR11/LPS and the DR11-K. They also provide certain generally useful capabilities. The format of each function as well as its name determines what operation the function will perform. That is, the same function called with a different number of arguments, or arguments of different values, can perform several related operations. The function in its most general format preceeds the section describing that function. The function in its specific format preceeds each subsection describing the particular operation that the function in that format performs. The last section of this chapter summarizes all possible formats for all the functions.

Unless otherwise specified, all the functions described in this chapter take integer arguments from -32768 to +32767 and return integer values. A function will round non-integer arguments to their integer part and will cause an integer overflow error (?38) on integer values outside the range -32768 to +32767.

8.2.1 The AR11/LPS Clock

The AR11/LPS clock can operate at rates from 1MHz to 100 Hz (Hz represents cycles per second). An internal counter which the program can load determines how many cycles comprise one clock tick. Thus, a clock running at 100 Hz with its counter set to 10 will tick 10 times per second. Using the rate and the counter in conjunction, the clock can be programmed to run at rates form 1 million ticks per second to 1 tick every 2.5 seconds for an AR11 and one tick every 640 seconds for an LPS. FOCAL allows its user a maximum clock tick rate of 10,000 ticks per second. Due to processor and interrupt overhead considerations it cannot handle a faster clock tick rate than this.

The clock can also be programmed to run at line frequency (50 Hz or 60 Hz) or at some other externally supplied frequency. Moreover, it can be programmed to tick repeatedly or to tick once and stop. FOCAL provides access to both these features.

The FTIC function which starts and stops the clock is the essential function for FOCAL's handling of the AR11/LPS since FOCAL uses the clock when sampling from the AR11/LPS analog to digital channels and when running the display. Like the other AR11/LPS functions, the number of arguments as well as their values determines the operation performed by the FTIC function. The FDLY function allows the FOCAL program to wait a specified number of clock ticks. The FTOI function allows the FOCAL program to determine the time of interrupt through an interrupt vector specified in a call to FINT.

FUNCTION FORMAT	OPERATION
FTIC(N,M)	START/STOP CLOCK
FTIC(N)	RETURN TIMER VALUE LESS N
FDLY(N)	WAIT N TICKS
FDLY(N,FN)	WAIT N TICKS WHILE EVALUATING FN
FTOI(V)	SAVE TIME OF EACH INTERRUPT THROUGH VECTOR V
FTOI(-V)	STOP SAVING TIME OF INTERRUPT THROUGH VECTOR V
FTOI(V,N)	RETURN LAST TIME OF INTERRUPT THROUGH V LESS N

8.2.1.1 Controlling the Clock: FTIC(N[,M])

The FTIC function allows the program to set the clock ticking repeatedly (or for a single tick) at a specified rate and count, to stop the clock, and to read and set a 16-bit timer that keeps track of the number of ticks that have occurred.

8.2.1.2 Start/Stop

FUNCTION FORMAT: FTIC(N,M) , N>0

To start or stop the clock the program uses FTIC with two arguments, the first greater than 0. The first argument specifies a rate, the second argument the number of cycles at this rate per clock tick. If the second argument is greater than 0, the clock will tick repeatedly at the specified clock tick rate. If the second argument is less than 0, the clock will tick once using the absolute value of the second argument as the number of cycles before the tick. FTIC will interpret the second argument modulo 256 for an AR11.

The first argument (which specifies the rate) can take any value 1 or greater. FOCAL interprets this value as an integer modulo 8. That is, M set to 8 will be interpreted as rate 0. The value of this argument corresponds to the hardware programmable clock rates except that FOCAL will automatically decrease rates that it cannot handle, i.e. those faster than 10,000 cycles per second. Thus, codes 1, 2, and 3 in the table below all set the rate to 10,000 cycles per second.

FIRST ARGUMENT	RATE
1	10 KHZ
2	10 KHZ
3	10 KHZ
4	1 KHZ
5	100 HZ
6	EXTERNAL INPUT(AR11) SCHMITT TRIGGER 1 (LPS)
7	STOP OR EXTERNAL FREQUENCY(AR11) LINE FREQUENCY (LPS)
8	STOP

To stop the clock the program specifies code 8 (interpreted as hardware rate 0). Stopping the clock will automatically set a 16-bit timer described in the next section to the value of the second argument. The program can also stop the clock by specifying a rate other than 8 as the first argument, and specifying a counter value of 0. This method of stopping the clock does not alter the timer value.

Examples:

```
X FTIC(3,5)   ;C: START THE CLOCK TICKING AT 2000 T.P.S.

X FTIC(8,0)   ;C: STOP THE CLOCK SETTING THE TIMER TO 0

X FTIC(6,4)   ;C: SET CLOCK SPEED TO LINE FREQUENCY DIVIDED BY 4
              ;C: FOR A 60 HZ CLOCK, 15 TICKS PER SECOND

X FTIC(4,-3)  ;C: MAKE CLOCK TICK ONCE IN EXACTLY 3/1000THS SEC.

X FTIC(3,0)   ;C: TURN THE CLOCK OFF LEAVING THE TIMER UNCHANGED
```

Starting the clock using FTIC causes one of FOCAL's task scheduling "slots" to be filled. When FOCAL returns to command mode, the clock stops and the "slot" is released.

8.2.1.3 Reading the Timer

FUNCTION FORMAT: FTIC(N), 0<=N<65536

A 16 bit counter called the timer maintains a count of the number of clock ticks that have occurred. Because each clock tick increments the counter by 1, the counter's value can be divided by the clock tick rate (expressed in ticks per second) to yield a current time measurement in seconds. Thus, a timer with a value of 5000 represents 2500 seconds when the clock tick rate that produced it is 2 ticks per second.

When FOCAL is loaded, the timer has the value 0. Thereafter, clock ticks or setting the timer using rate code 8 will alter the timer's value. The timer can assume integer values from 0 to 65535. When it overflows it returns to 0.

FTIC(N) with an argument greater than or equal to 0 and less than 65536 returns the value of the timer less the value of the argument, N. The program uses argument values to simplify determination of the number of ticks between events: When the first event occurs the program reads the clock and saves its value in a variable. At the occurrence of the next event it supplies this value as the argument to FTIC. The value returned is the time between events (inter-event interval).

Example:

```
    1.10 X FTIC(8,0);C:SET TIMER TO 0
    1.20 X FTIC(5,10);C: SET CLOCK TICKS AT 10 TICKS PER SECOND
    1.30 T "WHAT IS 2+2?";C: ASK THE USER A QUESTION
    1.40 S T=FTIC(0);C: RECORD TIME AT END OF QUESTION
    1.50 A A;C: WAIT FOR A RESPONSE
    1.60 S D=FTIC(T);C: GET TIME OF RESPONSE IN TENTHS OF A SECOND
    1.70 T "RESPONSE"%5.02,D/10," SECONDS",!
```

If the argument, when subtracted from the current value of the timer, would produce a value less than 0, the value returned equals 65536 plus this negative result. In other words, FTIC(N) returns a possible timer value, 0 to 65535, only.

8.2.1.4 Waiting

FDLY(N[,M])

The FDLY function allows the FOCAL program to wait a specified number of clock ticks before continuing program execution or to continually execute a function until the function returns a positive value or until a specified time expires. It requires that the AR11/LPS clock be running at the time of its executicn.

FUNCTION FORMAT: FDLY(N), N>0

FDLY with a single argument greater than 0 causes FOCAL to wait the number of ticks specified before returning to the next statement in the FOCAL program. This form of FDLY returns as its functional value the current value of the timer.

Example:

```
        X FDLY(99);T "DONE",!  ;C:  WAIT 99 TICKS, THEN TYPE 'DONE'
```

FUNCTION FORMAT: FDLY(N,M), N>0

FDLY with two arguments, the first of which is greater than 0, causes FOCAL to continually evaluate the second argument until the value returned by the second argument becomes greater than or equal to one, or until the number of ticks specified by the first argument have elapsed, whichever comes first. The second argument is usually a function whose value depends on some external operation such as input of a character from a remote keyboard. FDLY returns the last value returned by the second argument. Thus, FDLY will return a value greater than or equal to 1 whenever it returns before having reached its time limit. A return value less than or equal to 0 indicates that the time expired. If the clock is not running when the program calls FDLY, FDLY will wait indefinitely. That is, the program will hang.

Example:

```
   1.10 X FTIC(3,1);C:SET CLOCK TICKING AT 10000 TICKS PER SECOND
   1.20 T FTIC(5,FX(2,@177570));C:WAIT FIVE TICKS OR UNTIL THE VALUE
   1.30 C: IN THE REGISTER BECOMES GREATER THAN ZERO
```

This example waits 5/10000ths of a second for data to appear in the specified register. It then types out either some positive non-zero value, or, if the time has expired, a value equal to or less than zero.

FUNCTION FORMAT: FDLY(0,M)

If the value returned by the wait format of FDLY is positive and non-zero, then the time limit specified by the first argument had not elapsed when the the return occurred. That is, the second argument took on a positive value before the number of ticks specified in the first argument occurred. FDLY(0,M) returns the number of ticks that did occur on the previous execution of a wait, less the value of the second argument. Thus, FDLY(0,0) returns the number of ticks that occurred in the previous wait function. If that function had used up all its allotted time, the value returned by FDLY(0,0) will equal the number of ticks specified as that wait function's first argument.

The following line added to the previous example will type out the amount of time that passed before the data buffer went positive.

```
        1.40 T !,"WAIT TIME:",FDLY(0,0)/10000," SECONDS",!
```

8.2.1.5 Time of Focal Interrupt

The FOCAL program may specify a FOCAL group or line as an interrupt handler using the function FINT. When such an interrupt occurs FOCAL takes the first opportunity to transfer control to this group or line. It must wait, however, until it enounters a delimiter in the FOCAL line being executed when the interrupt occured. This can mean a substantial delay between interrupt time and the execution of the FOCAL interrupt routine. Certain applications need to determine the precise time of interrupt. The interrupt may, for instance, represent an experimental subject's response time, this time being the parameter under study.

FTOI(V[,N])

FTOI, the time of interrupt function, allows the FOCAL program to obtain the exact time of interrupt through any interrupt vector set up by a call to FINT.

8.2.1.6 Saving Interrupt Time

FUNCTION FORMAT: FTOI(V), V>0

The FOCAL program first executes FTOI with one argument. This argument specifies an interrupt vector previously included in a call to FINT. This call to FTOI tells FOCAL to save the AR11/LPS timer whenever an interrupt occurs through this vector. It is capable of saving the time of interrupt through any vector including, that of the console terminal.

FINT allows for eight separate interrupt routines. Using FTOI, the FOCAL program can save the time for each of them. An attempt to save the time through an additional vector will cause FTOI(N) to return 0 rather than a positive value. Appendix M, ASSEMBLING AND LOADING FOCAL LAB EXTENSIONS, describes how to create a version of FTOI that can save more than eight interrupt times.

FUNCTION FORMAT: FTOI(V), V<0

FTOI with a single, negative argument tells FOCAL to discontinue saving interrupt times for the vector indicated by the magnitude of the argument's value.

8.2.1.7 Returning Time of Interrupt

FUNCTION FORMAT: FTOI(V,N)

The FOCAL program calls FTOI in the FOCAL interrupt routine with two arguments to return the timer value at interrupt time. The first argument indicates the vector through which the interrupt that caused the transfer to the FOCAL interrupt routine occurred. The second argument specifies a value which will be subtracted from the AR11/LPS timer value that was preserved at the time of the interrupt. The FOCAL program uses the second argument's value in the same way it uses the single argument to FTIC when reading the timer. If it equals 0, the value returned by this form of FTOI will equal the value of the AR11/LPS timer at the instant that the interrupt occurred.

In the next example, FINT sets up group 5 as an interrupt processor and enables saving of the timer using FTOI. The program enables interrupts to the device at 170400, and then continues processing. At a later time, an interrupt occurs through vector 340. FTOI returns the elapsed time between the interrupt enabling and the interrupt.

Example:

```
1.10 X FTIC(3,1);C: START CLOCK
1.20 X FINT(@340,5,4,@170400,0);C: SCHEDULE GROUP 5 ON A/D INTERRUPT
1.30 X FTOI(@340);C: AND SAVE THE TIME OF INTERRUPTS
1.31 C: THROUGH THE VECTOR
1.50 S D=0

2.05 X FX(-2,@170400,@120);C:ENABLE INTERRUPTS BY EXTERNAL EVENT
2.07 S T=FTIC(0);C: GET CURRENT TIME
2.10 T "Z";I (D-1) 2.1;C:THEN WAIT FOR AN INTERRUPT TO OCCUR
2.20 T "TIME TILL INTERRUPT:",D/10000,"SECONDS";C:TYPE DELAY TIME
2.30 Q

5.10 S D=FTOI(@340,T);C: ON INTERRUPT,SAVE TIME SINCE
5.15 C: INTERRUPT ENABLED
5.20 X FTOI(-@340);C: THEN STOP SAVING INTERRUPT TIMES
5.30 RETURN;C: AND RETURN TO PROGRAM
```

8.2.2 Analog to Digital Conversions

The AR11/LPS provides access to a set of analog to digital channels which can take on digital values of from 0 to 1023 for the AR11 and from 0 to 4095 for the LPS. The values correspond to supplied voltages in either the range -2.5 to +2.5 volts, or 0 to +5 volts for the AR11 (depending on a software setting), and to voltage ranges dependent on the LPS hardware configuration. The LPS also allows programmable gain settings of 1, 4, 16 or 64. Individual channels may be sampled instantaneously, that is, at any given moment, or may be sampled automatically on each clock tick or on the occurrence of external events.

FOCAL provides two functions useful in sampling at rates of up to 5000 samples per second. Sampling may be from up to 8 separate channels in any specified sequence. FOCAL/RT-11 provides, in addition, a function for writing samples to a file at up to an aggregate rate of 3333 samples per second. This allows high speed storage of samples.

FUNCTION FORMAT	OPERATION
FSAM(N), N>=0	RETURN VALUE ON CHANNEL N
FSAM(N), N<0	WAIT FOR -N EXTERNAL EVENTS
FSAM(N,A,B,C...)	SAMPLE N POINTS INTO RING BUFFER SEQUENTIALLY FROM CHANNELS A, B, C ...
FSAM(0,N)	RETURN NTH SAMPLE FROM RING BUFFER
FBUF(N)	ALLOCATE BUFFER AREA N WORDS LONG
FFIL(N,M)	WRITE FROM RING BUFFER TO FILE ON CHANNEL N STARTING WITH SAMPLE M
FDMA(N,M,L)	INITIATE DIRECT MEMORY ACCESS TRANSFER OF N WORDS FROM CHANNEL M USING SAMPLE MODE SPECIFIED BY L

8.2.2.1 Taking Samples

FSAM(N[,M])

The FOCAL program may use the FSAM function to sample instantaneously from a specified channel or to read a specified number of samples into a buffer. Therefore, it provides all of the capabilities of the FADC function. The FADC function remains available in FOCAL for compatibility. The call to FSAM also tells FOCAL whether to sample on each clock tick or upon the occurrence of external events.

8.2.2.2 Direct Sampling

FUNCTION FORMAT: FSAM(N), N>=0

The FSAM function used with a single argument greater than or equal to 0 returns the current digital value on a channel. The channel sampled is specified by the argument taken modulo 16 for the AR11, modulo 64 for the LPS. The digital value returned will be in the range 0 to 1023 for the AR11 and 0 to 4095 for the LPS.

For the AR11 only, adding 32 to the value of the channel specified causes unipolar sampling, sampling in the range 0 to 5 volts. Otherwise, sampling is bipolar in the range -2.5 to +2.5 volts. When directed toward an LPS only, each argument can specify a gain as follows:

NUMBER ADDED TO ARGUMENT	GAIN SETTING
0	1
16	4
32	16
48	64

The gain setting on the LPS determines the amplification of the incoming voltage. Thus, a gain setting of 4 would amplify a voltage of .1 to .4.

Example:

```
S D=FSAM(0);C: GET THE CURRENT VALUE ON AR11 CHANNEL 0, BIPOLAR

S D=FSAM(0+32);C: GET THE CURRENT VALUE ON 0, UNIPOLAR

S D=FSAM(2+16);C: GET THE VALUE ON LPS CHANNEL 2, GAIN 2
```

8.2.2.3 Waiting for External Events

FUNCTION FORMAT: FSAM(N), N<0

FSAM with one negative argument causes a delay while external events take place. The number of external events is the absolute value of the argument. This form of FSAM allows the FOCAL program to wait for a number of external events, but does not return an A/D value when completed. For the LPS, an external event is one firing of Schmitt trigger one, while for the AR11 an event is a TTL high-to-low transition on the External Event line.

Example:

```
   X FSAM(-100);C:  WAIT FOR 100 EXTERNAL EVENTS TO TAKE PLACE
```

8.2.2.4 Asynchronous Sampling

FUNCTION FORMAT: FSAM(N,A,B,C,...), N<>0

FSAM with two or more arguments, the first non-0, allows for high speed asynchronous data sampling into a ring buffer described in the next section. The absolute value of its first argument specifies the number of samples to take. The remaining arguments, up to 8 of them, specify a sequence of channel numbers from which to take these samples. FSAM will return immediately after being called having initiated sampling from these channels. Thereafter, sampling will occur asynchronously. That is, while the FOCAL program runs, sampling will continue until the number of samples specified have been loaded into the ring buffer. The FOCAL program can read these values from the buffer while sampling occurs.

FSAM can sample on each clock tick or upon occurrence of each external event. The sign of the first argument specifies which method of sampling it will use. A positive argument indicates sampling by clock tick; a negative one indicates sampling by external event. The AR11 has a single external event input line which causes the sample to occur. On the LPS, Schmitt trigger one causes the sample. If the clock is started using rate 6, the external event rate, external events rather than passage of time cause clock ticks. In this case, requesting sampling on clock tick, that is, using a positive first argument, will, in effect, cause sampling by external event. In any case, sampling will not begin until a clock tick or an external event occurs, and sampling will discontinue when clock ticks or external events cease to occur. Thus, the FOCAL program that calls FSAM to sample by clock tick need not start the clock until after the call to FSAM and may stop the clock, or even change its rate, while sampling occurs.

FSAM can sample at up to 5000 points per second. This corresponds to a clock rate of 3 and a counter value of 2 (i.e. the clock tick rate produced by FTIC(3,2)).

The arguments that follow the first argument determine the sequence in which FSAM will read data from channels into the buffer. The first sample will come from the channel whose number appears in the first argument, the second from the channel specified by the next argument, and so on for up to eight arguments. Then the sampling sequence will start again. The arguments may each specify a different channel or may specify several channels repeatedly. When directed toward an AR11 each argument may specify unipolar (channel number + 32) or bipolar sampling. When directed toward an LPS each argument can specify a gain as described previously. If only one channel argument is specified in the call to FSAM, sampling will occur on that channel only.

Example:

```
        X FTIC(5,1);X FSAM(1000,0,32+1)
```

This example initiates the sampling of 1000 values from an AR11 alternately from channel 0 in bipolar mode and channel 1 in unipolar mode. Thus, every odd sample will come from channel 0, every even one from channel 1. A sample will occur each time the clock ticks, that is, 100 times per second.

When sampling by clock tick, FSAM writes the first sample into the first slot in the ring buffer, the second sample into slot 2 and so on. When sampling by external event, FSAM can also save the AR11/LPS timer value when each sample occurs. If the FPRM function parameter 13 has the value 0, sampling will occur as for clock tick sampling, one sample per buffer slot. If parameter 13 is greater than 0, then each time an external event occurs FSAM will save the timer value in the next available buffer slot and the A/D value in the slot following. Each sample taken in this mode uses two buffer slots.

When FSAM has loaded the required number of samples, it stops sampling. This does not affect execution of the FOCAL program.

Example:

```
        X FPRM(13,1);X FSAM(-100,4)
```

This example initiates sampling by external event from channel 4. It first sets parameter 13 positive so that sampling occurs in timer/data pairs. It will use 200 buffer slots to save the 100 timer/data pairs. This mode of sampling can be used to create Post-Stimulus (PST) and Time-Interval (TIH) histograms. For PST histograms, the first external event, sensed as described in 8.2.2.3, acts as the stimulus. Generally, the analog values obtained are not utilized in creating histograms.

FUNCTION FORMAT: FSAM(0,N)

FSAM with two arguments, the first equal 0, provides retrieval from the sampling buffer. The sampling buffer is a ring buffer. Data taken by the asyncrhonous form of FSAM starts loading into the first slot in the buffer and continues sequentially to higher slots until it reaches the end of the buffer. Then, if the sample count has not been reached the next sample will go into the first word of the buffer overwriting data previously placed there. This wraparound process continues until all samples have been taken.

The second argument to this form of FSAM specifies a sample number. If the sample specified has not yet been taken, this form of FSAM returns a -1. If the sample called for has been taken, but has already been overwritten by subsequent samples, the function will return a -2. Otherwise, it will return the value of the nth sample. If the slot contains an A/D value, this value will be in the range 0 to 1023 for the AR11 and 0 to 4095 for the LPS. If it contains a timer value, it will be in the range 0 to 65535.

Example:

```
        1.1 F I=1,1,100;S D(I)=FSAM(0,I);I (D(I)) 5
```

This example loads array D with values from the A/D buffer presumably filled or being filled by a previous call to FSAM. It goes to group 5 when it encounters a value not yet taken or already overwritten.

8.2.2.5 Allocating Buffer Space

FBUF(N)

FBUF allows the FOCAL program to set aside space in memory.

FUNCTION FORMAT: FBUF(N)

FBUF allocates or deallocates buffer area. For an argument greater than zero it deallocates any previously allocated buffer area, and then allocates as buffer area the number of words of memory specified as the argument. It returns the base address of this memory space or causes a ?09 error if unsuccessful. For an argument of zero it deallocates any previously allocated buffer area and returns the value zero. In the Paper Tape version of FOCAL, a call to FBUF also erases all variables.

The buffer area allocated can be used by the FX function as storage area. FBUF returns as its value the base address of the usable area. The FOCAL program can calculate the top address of this area by adding the buffer's size in bytes to the base address. The size, in bytes, under Paper Tape FOCAL, equals twice the number of words requested in the call to FBUF. Under FOCAL/RT-11, FBUF allocates enough 256 word blocks to satisfy the request. Thus the size, in words, of the area allocated is the multiple of 256 greater than the number of words requested. The size in bytes is twice this value. On FBUF(300), for example, FOCAL/RT-11 allocates 512 words (1024 bytes) of memory.

Example:

```
1.10 S BA=FBUF(100);C: ALLOCATE 100 WORDS AS BUFFER AREA
1.30 S TA=BA+200;C: SET MAXIMUM ADDRESS
1.50 F I=BA,1,TA;X FX(-1,I,FCHR(-1))
1.60 F I=BA,1,TA;T FX(1,I),!
```

This example stores characters by byte in the buffer area. It then lists their ASCII values.

The space allocated by FBUF is automatically used by the asynchronous call to FSAM as its ring buffer area.

The next example allocates a ring buffer 250 words long and starts sampling of 500 A/D values into it at 50 per second. As sampling occurs the program reads data from the buffer and stores it in an array. If it encounters overwritten data while doing so, it halts with an error message.

Example:

```
1.10 X FTIC(5,2);C: SET CLOCK TO 50 TICKS PER SECOND
1.20 X FBUF(250);C: ALLOCATE 250 WORDS
1.30 X FSAM(500,0,2.1+32,2);C: SAMPLE 0,2,1,2,0,2,1,2(1 UNIPOLAR)
1.35 S I=1;C: INIT COUNTER
1.40 S D(I)=FSAM(0,I);C: GET A SAMPLE
1.50 I (D(I)+1)2,1.4;C: QUIT IF OVERTAKEN.WAIT IF NO DATA YET
1.60 S I=I+1;I (I-500)1.4,1.4
1.61 C: IF DATA TAKEN, GO RAD NEXT SAMPLE
1.70 T "ALL DATA SAVED",!;C: WHEN DONE,QUIT
1.80 Q

2.10 T"DATA RATE TOO FAST",!;C: STOP IF PROGRAM CAN'T KEEP UP
2.20 Q
```

8.2.2.6 Saving Samples in an RT-11 File

FFIL(N)

A FOCAL/RT-11 program can save samples in an RT-11 file by opening the file using the Library Open command, retrieving each sample from the ring buffer using FSAM, then writing the sample to the file. This method limits the transfer rate to the rate at which FOCAL can execute the requisite 'FOR' loop. Function FFIL increases the maximum transfer rate to 3333 samples per second by enabling the FOCAL program to write directly from the ring buffer to a file.

FUNCTION FORMAT: FFIL(N)

FFIL writes data from the sample buffer to a file opened on the channel specified as its argument. It continues writing sequentially higher numbered samples from the ring buffer until all existing samples have been transferred. Then it returns to the FOCAL program with the number of this last sample as its functional value. FFIL with an argument greater than or equal to zero initializes for output to the channel specified by the argument. The FOCAL program must open a file on this channel using the LIBRARY OPEN command before issuing this call to FFIL. Once output is initialized a call to FFIL with a negative argument writes samples from the ring buffer to the file on disk. The first time it is called it starts writing at sample 0 and continues writing until it encounters a sample not yet written. It then returns the number of the first untaken sample as its functional value. The next call to FFIL with a negative argument starts writing from this untaken sample and continues until an untaken sample is again encountered. The FOCAL program can determine when FFIL has written all the data to be sampled by comparing the value it returns to the sample count requested in the call to FSAM. When the value returned exceeds the sample count, FFIL has written out all the samples. At this time, or any time during sampling, the FOCAL program can execute a Library Close to the opened channel to make the file there permanent. If the file was a virtual file, the FOCAL program can access the data already stored in it during sampling while the file is still open. If FFIL is called specifying an unopened channel, it will return a -2.

Usually FFIL is scheduled by the FQUE function to execute often enough to keep up with the sampling rate as in the next example. If it cannot keep up, it will return a -1 indicating that it has tried to obtain overwritten data. In this case, the programmer must modify the program to either execute FFIL more frequently, decrease the sampling rate, or increase the size of the ring buffer.

The rate at which FFIL can write samples to an RT-11 file is limited by the speed of the output device. For an RK disk, the maximum sustainable transfer rate is 3333 samples per second. That is, with the clock rate set to 3 and the clock counter set to 3, given sufficient ring buffer size, FFIL will write samples from the ring buffer as fast as samples are being added to it. The ring buffer must however be sufficiently large so that when FFIL pauses to write out a block, the values input do not catch up to the current sample being output by FFIL. A ring buffer 512 to 2048 entries long is sufficient for this purpose depending upon the transfer rate.

The next example maintains a rate of 1000 samples per second using a ring buffer 1024 long. Notice that before beginning sampling but after the buffer allocation, it issues a call to FFIL with a count argument less than 0. This call performs no function other than to initialize certain I/O operations which, if first executed during sampling, would consume valuable time. Executed before sampling starts, their overhead does not detract from the time available for writing from ring buffer to disk.

Example:

```
1.10 X FTIC(8,0);C: STOP CLOCK AND CLEAR TIMER
1.20 L MAKE 1,SAMP.DAT/I/V:S(0);C: OPEN A FILE FOR OUTPUT
1.30 X FBUF(1000);C: ALLOCATE RING BUFFER
1.35 X FFIL(1);C: OPEN CHANNEL 1
1.40 X FSAM(10000,0);C: SAMPLE 10000 POINTS
1.50 S I=0;C: SET COUNTER TO 0
1.55 X FPRM(11,1);C: SET FOR LINE CLOCK BY TICK
1.57 X FTIC(3,10);C: TICK AT 1000 TICKS PER SECOND
1.60 S ID=FQUE(1000,3,30,0,4);C: SCHEDULE GROUP 3 EVERY 10 TICKS
1.80 S J=1;F I=100,100,10000;D 2;C: DO 2 WHILE GROUP THREE INTERRUPTS
1.90 T "10000 POINTS SAMPLED",!;C: WHEN DONE, SAY SO
1.95 D 4;Q;C: THEN TYPE OUT EVERY 50TH SAMPLE

2.05 C:  COPY EVERY 100TH VALUE INTO AN ARRAY DURING SAMPLING
2.10 I (C-I)2.1;C: WAIT FOR THIS POINT TO BE SAMPLED
2.20 S D(J)=S(I);C: GET THIS ELEMENT INTO AN ARRAY IN CORE
2.30 S J=J+1
2.40 R

3.10 S C=FFIL(-1);C: WRITE TO DISK
3.12 T C,!;C: TYPE LAST SAMPLE
3.15 I (C) 3.5;C: QUIT IF OVERTAKEN
3.20 I (10000-C) 3.3,3.3;C: QUIT IF ALL SAMPLES DONE
3.25 R;C: ELSE RETURN FROM INTERRUPT
3.30 X FQUE(0,ID,3);C: WHEN ALL DONE, UNSCHEDULE THIS ROUTINE
3.35 T "CURRENTLY COPYING ",I,!;C: SAY WHAT SAMPLING BEING ALTERED BY NOW
3.40 R
3.50 T "TOO FAST";C: IF OVERTAKEN, SAY SO
3.60 Q;C: AND QUIT

4.10 F I=1,1,100;T !,I*100,D(I);C: TYPE OUT A VALUE
```

This example opens the file SAMP.DAT on channel 1 as an integer virtual file, schedules FFIL to run every 60 ticks (1 second), then begins sampling. Every 60 ticks FFIL writes out all the samples taken since its last execution. It does so by calling FFIL with a negative argument. It also types out the value of the last untaken sample to indicate what sample it has reached.

As sampling continues, the FOCAL program alters every 50th value in the virtual file to change their range to -2048 to 2048. When all samples have been taken, the program types out what sample it has already altered and continues to alter samples until it reaches the last sample taken, the 10000th. Then it halts.

At the sampling rate of 1000 samples per second, about 1000 points are loaded every second. So, on the average, FFIL must write 1000 values to the disk each time it is called. In fact, due to varying amounts of time required for disk writes on each call, FFIL writes out a different number of values each time it is called. The sample buffer size must be large enough so that under worst case conditions, that is, when FFIL must write out the largest number of blocks, the incoming samples do not wraparound the ring buffer and catch up to the current sample that FFIL is outputting. The ring buffer size used, 1024 is large enough, so that FFIL always has enough time to complete writes to disk. A faster sampling rate might require either a larger buffer size or faster scheduling of FFIL.

8.2.2.7 Direct Memory Access Sampling

The LPS provides the user with the capability of direct memory access sampling. This form of sampling reads samples into memory directly without the intervention of an interrupt routine. This enables sampling at high rates.

FDMA(N,M,L)

The FDMA function allows the FOCAL program to use the direct memory access feature of the LPS. This increases the possible sampling rate by clock tick to 10,000 samples per second and the external event sampling rate to the maximum that the hardware can sustain. Using the FDMA function the FOCAL program can initiate direct memory access sampling into the sampling buffer in either single or dual mode, in burst or non-burst mode, and either by clock tick or by external event.

FUNCTION FORMAT: FDMA(N,M,L)

The first argument to FDMA specifies the number of samples to take. When sampling is initiated the number of samples specified will be read directly into the sampling buffer without intervention of an interrupt routine. The number of samples specified must be less than the size of the sampling buffer since direct memory access samples cannot wraparound the end of the buffer but must be read into a contiguous area of memory. FDMA will automatically limit the number of samples to the length of the buffer.

The second argument to FDMA specifies the channel (modulo 64) from which to sample. All samples will come from this channel. Adding the value 128 to the second argument, however, enables dual sampling. In dual sampling two values are read at the same instant from two separate channels and entered one after the other into the sampling buffer. In dual mode sampling the channel number is interpretted modulo 8 and the second channel is determined by adding 8 to the number of the first channel. Thus, a second argument of 129 would cause dual mode sampling from channels 1 and 9. When doing dual mode sampling the first argument indicates the total number of samples to take, not the number of pairs to take, and should, therefore, be an even number.

The third argument to FDMA specifies the mode of sampling as indicated in the table below:

VALUE OF ARGUMENT	OPERATION
0	BURST MODE SAMPLING
1	BURST MODE SAMPLING
2	SCHMITT TRIGGER 1 CAUSES EACH SAMPLE
3	CLOCK OVERFLOW CAUSES EACH SAMPLE

In burst mode sampling, samples are taken at the maximum possible rate that the hardware will allow.

FDMA returns control to the FOCAL program which executes while sampling takes place. The FOCAL program can read samples from the sample buffer using FSAM in the same way it would retrieve non-DMA samples. Since, for DMA sampling, the buffer does not act as a ring buffer, all sample numbers specified in the call to FSAM must be less than the size of the ring buffer. Otherwise, FSAM will return a -1 indicating sample not yet taken. The FOCAL program can also use the function FFIL with DMA sampling to write the samples to an RT-11 file. Since the sample buffer is not a ring buffer, the total number of samples written to the RT-11 file can never exceed the size of the sampling buffer.

Example:

```
        1.10 X FTIC(3,1);C: START CLOCK AT 10,000 TICKS PER SECOND
        1.20 X FBUF(2000);C: ALLOCATE SAMPLING BUFFER 2000 LONG
        1.30 X FDMA(1998,22,3):C: START SAMPLING BY CLOCK TICK
        1.40 I (FSAM(0,1999))1.4;C: WAIT FOR COMPLETION
        1.50 X FITC(3,0);C: THEN SHUT CLOCK OFF
        1.60 F I=1,1,1999;T FSAM(0,I);C: THEN LIST ALL SAMPLES TAKEN
```

This example initiates DMA sampling of 1999 values from channel 22 at 10000 values per second. It waits for completion then types out all the values.

8.2.3 Graphics

Under program control the AR11/LPS can output x and y coordinates to a CRT such as the VR14. The CRT uses these coordinates to either display a point or set the location of the display beam. FOCAL simplifies access to these coordinate registers by providing its user with a set of graphics commands. When FOCAL executes each command, it converts the command to a series of output operations through the coordinate registers. The FOCAL program can create a display file and load it with these graphics commands. The commands can specify visible and invisible fixed points, relative offset points, points plotted along either the x or y axis, or characters including format control characters. The FOCAL display processor internally converts the instructions in the file into a series of output operations to the AR11/LPS x and y registers. But the user need not concern himself with the mechanics of the conversion process.

The display file is an area of memory allocated by the FOCAL program. It is divided into 2-word slots called 'LOC's each of which can contain an individual graphics instruction. The LOCs are numbered consecutively starting at 1 so that the FOCAL program can load them with instructions individually in any order. This allows the program to substitute one graphics instruction for another, and one character for another. For instance, to add to or delete from the file, to cause the display to move, and, in general, to completely manipulate the image displayed.

When the FOCAL program starts the display, the display processor is activated. It interrupts the FOCAL program's exececution at regular intervals to execute the instructions in the display file. It starts execution with the instruction contained in LOC 1, performing whatever graphics operation it specifies. It continues executing graphics instructions in sequentially higher numbered LOCs until it reaches the end of the display file, a LOC that has not been loaded with a graphics instruction, or a LOC that contains an end of file instruction. It then returns control to the interrupted FOCAL program. The more instructions in the display file, the longer it takes to execute a pass through it, character instructions requiring about ten times as long to execute as point instructions. The longer it takes to execute one pass, the dimmer the display at a given intensity setting, the greater the tendency to flicker, and the less processor time available to the executing FOCAL program. Actual time to execute a pass through the file depends on hardware factors such as memory speed and the type of processor used. As a rough guide, loading the display file with up to 500 points or up to 50 characters or some combination of the two will produce a relatively flicker free image. There is no limit, however, to the number of instructions the FOCAL program can load and attempt to display.

Each time the display processor passes though the display file, it produces an image on the screen. At about 30 images per second, it acts like a motion picture projector to produce a steady image. As instructions in the file are changed by the executing program, the image produced by a pass through the display file changes, too. So the FOCAL program can create an image that moves or changes dimensions by continually changing the instructions contained in the display file.

The display processor determines when to execute a pass through the display file by counting AR11/LPS clock ticks. Therefore, in order to produce an image on the screen, the FOCAL program must first start the AR11/LPS clock using the FTIC function. The clock must be running at 30 ticks per second or faster to produce a steady image. The FOCAL program may set the time interval between images by using function FPRM to set parameter 13. The value of this parameter determines the amount of time betweeen each image. That is, the amount of time available to the FOCAL program. The value of parameter 13 divided by 100 gives this time interval in seconds. Thus, setting its value to 4 indicates a time interval of 1/25th (4/100ths) of a second between images. After calling FPRM to change the value of parameter 13, the program must call FTIC in order to effect the change in the interval. Parameter 13 set to 0 gives the same interval as parameter 13 set to 1, namely, 1/100th second.

FUNCTION FORMAT	OPERATION
FCRT(0,N)	ALLOCATE/DEALLOCATE DISPLAY FILE
FCRT(N), N>=0	TURN DISPLAY ON/OFF AT INTENSITY N
FCRT(L,N[,M]), L<>0	LOAD GRAPHICS INSTRUCTION INTO LOC L
FCRT(N), N<0	SET CHARACTER SCALING TO SIZE N
FFRM(N,M)	SAVE FRAME M IN FILE ON CHANNEL N

8.2.3.1 Creating the Display

FCRT(N[,M])

The single function FCRT in its various formats offers the FOCAL program access to all the graphics capabilities of FOCAL. It allocates/deallocates display file, turns the display on or off, sets the size of characters, and loads all graphics instructions.

8.2.3.2 Allocating Display File

FUNCTION FORMAT: FCRT(0,N)

When the first of two arguments to FCRT equals 0 the second argument specifies the number of LOCs to allocate as a display file. If the second argument equals 0, the display file is deallocated and the display turned off. Otherwise, any previous display file is deallocated and the number of LOC's specified is allocated. Under RT-11, which can only allocate memory in blocks of 256 words (128 LOC's), enough space is allocated to satisfy the request. This form of the function returns the number of LOC's actually allocated which, under RT-11, may differ from that requested. If the amount of memory requested is unavailable, this form of FCRT will cause a ?23 error. In the paper tape version display file allocation/deallocation also erases all variables.

8.2.3.3 On/Off

FUNCTION FORMAT: FCRT(N),N>=0

With one non-negative argument, FCRT specifies the intensity level of the display. A level of 0 turns the display off. Any other value indicates display on. The higher the value, the brighter the display but the less processor time available for executing the FOCAL program. Setting the argument to 1 or 2 produces a bright enough display for most puposes. Until the display file is loaded with graphics instructions, turning the display on will produce no image on the screen.

8.2.3.4 Loading Graphics Instructions

FUNCTION FORMAT: FCRT(L,N[,M]), L<>0

FCRT with two or three arguments, the first of which is non-zero loads a graphics instruction into the LOC specified by the absolute value of the first argument. It returns as its functional value a positive value one greater in value than the first argument. This number is that of the next succeeding LOC in the display file. This simplifies sequential loading of the display file: the FOCAL program need only supply the returned value as the first argument to the next loading function in order to load the next sequential LOC. If the magnitude of the first argument is greater than the number of LOC's allocated, LOC 1 is loaded and the value 2 returned.

FCRT with three arguments loads a point into the LOC specified by the first argument. This point can be either an absolute point, a relative point, or an incremental point depending on the form of the next two arguments. An absolute point is a point displayed at a fixed location on the screen. This point will remain at this location regardless of the beam location set by the graphics instruction in the preceeding LOC. The screen is addressable on a coordinate system starting at x=0 and y=0 at the lower left to x=1023 and y=1023 at the upper right for an AR11 and x=4095 and y=4095 at the upper right for an LPS. Those CRT screens set up in a rectangular format (equal dot spacing on both x and y axes) have a maximum y value of 767 (AR11) or 3071 (LPS). Whenever neither the x or y arguments to FCRT is preceeded by a + or - sign, they specify the coordinates of an absolute, fixed point. Since all screen locations can be specified by positive coordinates, there is no need for a minus sign when specifying an absolute point.

If a + or - sign preceeds both the second and third arguments, they specify the offset of a relative point. A relative point is a point displayed at a distance relative to the beam location set by the graphics instruction in the previous LOC. Thus, FCRT(2,+10,-20) specifies a point in LOC 2, 10 units to the right and 20 units below the point or character matrix in LOC 1. A relative point loaded into LOC 1 specifies an offset from the lower left corner of the screen. If a series of LOCs contain relative points and a LOC containing an absolute point preceeds them, then when the program moves the absolute point by re-loading the LOC that contains it, all the relative points following it will also move. Such manipulations allow the FOCAL program to move figures around the screen.

If a + or - sign preceeds only one of the coordinate arguments, that argument will display relatively while the other coordinate will display absolutely. This enables point plotting in either the x or y direction. FCRT(LOC,+5,100) loads a point of y-coordinate 100, 5 to the right of the point in the preceeding LOC. Loading sequential LOCs leaving the second argument as +5 and changing the y argument plots points along the x axis each separated by 5 units.

Example:

```
        1.10 X FTIC(4,1);C: START CLOCK
        1.30 X FCRT(0,128);C:CREATE DISPLAY FILE OF 128 LOCS
        1.40 X FCRT(2);C: TURN DISPLAY ON WITH INTENSITY OF 2
        1.50 X FCRT(1,0,0);C: PLOT A ABSOLUTE POINT AT (0,0)
        1.60 S D=30;C: SET DELTA X VALUE TO 30
        1.70 F I=2,1,101;D 2;C: PLOT 100 USER SUPPLIED POINTS
        1.80 Q

        2.10 A "POINT",Y; (Y)1.8;C:GET A POINT, QUIT IF<0
        2.20 X FCRT(I,D,Y);C:PLOT THE POINT WITH DELTA X=30
        2.30 R
```

This example plots the values input by the user along the x axis. It leaves 30 units between each point.

If the first argument to a three argument FCRT has a negative value, its magnitude specifies the LOC to load and the remaining two arguments specify the type of point to load as for the case of a positive first argument. But with this form of call, absolute points and points plotted along the x axis will not appear on the screen. That is, they will be invisible and will only alter the location of the display beam relative to which the instruction in the next LOC will plot. The next example sets an invisible absolute point in LOC 1 to indicate the start of a curve drawn with relative points starting at that screen location.

Example:

```
        1.10 X FTIC(5,1);C: START CLOCK
        1.20 X FCRT(0,100);C: ALLOCATE DISPLAY FILE
        1.30 X FCRT(1);C: TURN ON DISPLAY
        1.40 X FCRT(-1,100,500);C: SET LOCATION OF DISPLAY BEAM
        1.50 S L=2;C: POINT TO LOC NUMBER 2
        1.60 F I=2,1,80;S L=FCRT(L,+10,+I);C: ADD RELATIVE POINTS
```

When FCRT has two arguments, it causes loading of a character into the LOC specified by the first argument. The second argument specifies the character. If it is positive, it specifies one of the 64 ASCII characters of octal codes 40 to 137. Decimal argument values 1 through 31 specify ASCII characters of codes 100 through 137. Values 32 through 63 specify ASCII characters of octal codes 40 through 77. An alternate method of specifying values 1 through 26 (decimal) is described in section 2.4 of this manual.

The characters are each displayed on a 5x7 matrix, the lower left hand corner of the matrix positioned at the location on the screen set by the graphics instruction in the preceeding LOC. Once displayed, a character leaves the beam positioned one column to the right of the lower right hand corner of the 5x7 matrix so that characters loaded into sequential LOCs display horizontally across the screen.

In a call to FCRT with two arguments if the value of the first argument is less than 0, then the second argument specifies one of five control characters listed in the table.

CODE	CHARACTER
0	NULL
1	LINE FEED
2	UPSPACE
3	CARRIAGE RETURN
4	END OF FILE

The null character effectively erases anything in the LOC loaded. It causes no change in the display. The FOCAL program, after having allocated a display file may, before loading it with graphics instructions, first fill it completely with nulls. Then any LOC loaded, no matter where in the file, will be executed by the display processor and cause display. Otherwise, the display processor stops executing the display file as soon as it encounters a LOC not yet loaded with a graphics instruction. Any LOC beyond this loc, even though loaded, will not be executed and will cause no display.

Line feed moves the display beam down 9 matrix rows.

Upspace positions it up 9 rows.

Carriage return moves the beam to the left edge of the screen without altering its height.

End of file tells the display processor to discontinue executing graphics instructions, to ignore graphics instructions in all succeeding LOCs.

Example:

```
        1.10 X FTIC(4,1);C: START CLOCK
        1.30 X FCRT(0,128);C: CREATE DISPLAY FILE OF 128 LOCS
        1.40 X FCRT(1);C: TURN DISPLAY ON WITH INTENSITY 1
        1.50 X FCRT(-1,0,1000);C: SET BEAM TO (0,1000)
        1.60 S L=2;C: POINT TO LOC 2

        2.10 T "TYPE CHARACTER VALUES (0 TO QUIT)",!;C: ASK FOR INPUT
        2.20 A !,C;C: GET A CHARACTER VALUE
        2.30 I (C-1) 2.8;C: IF VALUE 0,GO DO NEXT LABEL
        2.40 S L=FCRT(L,C);C: ELSE,LOAD CHARACTER INTO NEXT LOC
        2.50 S L=FCRT(-L,3);S L=FCRT(-L,1);C: THEN <CR><LF>
        2.60 G 2.2;C: GO GET NEXT CHARACTER
        2.80 S L=FCRT(-L,4);C: WHEN LABEL DONE, END FILE
        2.90 G 1.6;C: AND GO DO NEXT LABEL
```

This example asks the user to input a series of character values and plots the corresponding characters vertically down the left side of the screen by inserting a carriage return and a line feed character after each character.

CHARACTER SCALING

FUNCTION FORMAT: FCRT(N), N<0

FCRT with one negative argument specifies the size of all characters displayed. It can be called with display on or off. FOCAL allows five characters sizes indicated by arguments from -1 to -5, -1 indicating the smallest. Size doubles for each higher magnitude argument. Characters of size 5 are 16 times larger than those of size 1. When loaded, and whenever it allocates a display file, FOCAL sets character size to size 2.

Example:

The following lines added to the previous example allow the user to specify the size of the characters displayed:

```
        1.8 A "TYPE SCALING(1-5)",S
        1.9 X FCRT(-S);C: SET SCALING TO VALUE INPUT
```

The next example makes use of all the display functions. Note that all graphics programs require that the clock be running at 20 ticks per second or faster in order to produce a steady display. If the clock stops, the display stops but will restart as soon as the clock restarts. The FOCAL program may change the rate of the clock without affecting the display as long as the rate exceeds 30 ticks per second. Clock rates slower than 30 ticks per second cause the display to blink. The rate of blink equals the clock rate. Thus, setting the clock to 1 tick per second will cause the display to flash on about once per second.

Example:

```
  1.05 C:PROGRAM SLASH
  1.10 X FPRM(13,1);X FTIC(5,1);S SC=1;C: 1/100TH BETWEEN IMAGES START CLOCK
  1.11 C: START CLOCK
  1.15 X FCRT(0,100);X FCRT(-3);C:ALLOCATE FILE,ETC.
  1.20 A "LPS ON SYSTEM? ",L;C: IF LPS ON SYSTEM,
  1.30 I (L-0YES) 1.5,1.4,1.5
  1.40 X FPRM(12,2);S SC=4;C:COMMENT
```

```
1.70 F L=1,1,20;X FCRT(L,512*SC,L*50*SC);C: DRAW CENTER LINE
1.75 C: SET SOME USEFUL VARIABLES
1.80 S CX=20*SC;S CY=28*SC;S PL=80*SC;S PR=954*SC;S SR=980*SC
1.85 A "SPEED OF PLAY (1-100)",V;C: GET START X VELOCITY
1.90 S CW=2*CX;S CH=2*CY;S SL=44*SC;S ST=SR;S SB=SL;S V=V*SC

2.10 S LY=FSAM(0);S RY=FSAM(3);S PH=80*SC;C: PADDLE LOCATIONS AND HT
2.20 S LP=21;X FCRT(LP,PL,LY);C: ENTER BASE OF LEFT PADDLE
2.30 F L=LP+1,1,LP+11;X FCRT(L,+0,+(PH/10));C: DRAW LEFT PADDLE
2.40 S RP=LP+12;X FCRT(RP,PR,RY);C: RIGHT PADDLE FOLLOWS LEFT
2.50 F L=RP+1,1,RP+11;X FCRT(L,+0,+(PH/10))
2.60 A "HOW MANY POINTS WINS? ",GP;C: GET POINTS PER GAME
2.70 S LS=RP+12;X FCRT(-LS,200*SC,900*SC);C: SET BASE OF LEFT SCORE
2.80 S RS=LS+3;X FCRT(-RS,700*SC,900*SC);C: AND OF RIGHT SCORE
2.90 S LT=0;S RT=0;D 9;C: SET SCORES TO 0 AND DISPLAY THEM

3.10 S B=RS+3;C: SET LOC WHERE BALL STARTS
3.20 S X=512*SC;S Y=X;C: SET BALL'S COORDS TO CENTER SCREEN
3.30 X FCRT(-B,X-CX,Y-CY);D 11;C: DRAW IT THERE
3.40 X FCRT(1);C: START DISPLAY
3.50 A "WHO SERVES,R OR L?",L;C: GET SERVER
3.60 S DX=V; I (L-0L) 4.1,3.7,4.1;SET DELTAX NEG. OR POS.
3.70 S DX=-V

4.10 S DY=0;S S=0;C: SET DELTAY=0 AND SPIN=0
4.20 I (V-FABS(DX)) 4.25;S DX=DX*1.5;C: KEEP GAME FAST
4.25 S X=X+DX;C: ALTER X LOCATION OF BALL
4.30 S Y=Y+DY;C: ALTER Y
4.40 X FCRT(-B,X-CX,Y-CY);C: DRAW NEW CENTER OF BALL
4.50 S L1=LY;S LY=FSAM(0);C: GET LEFT PADDLE LOCATION
4.60 S R1=RY;S RY=FSAM(3);C: AND RIGHT
4.70 X FCRT(LP,PL,LY);X FCRT(RP,PR,RY);C: MOVE PADDLE IMAGES
4.80 D 10;C: DO BOUNCE OFF CIELING AND FLOOR

5.20 I (PL-(X-CW)) 5.7;C: IF BALL TO LEFT OF LEFT PADDLE,
5.30 I (-DX) 4.2;C: AND TRAVELING LEFT
5.40 I (Y+CY-LY) 6.5 ;C: AND ABOVE BASE OF LEFT PADDLE
5.50 I (LY+PH-(Y-CY)) 6.5;C: AND BELOW ITS TOP,
5.60 S S=((LY-L1)+S)/2;S DY=DY-S;S S=LY-L1;G 6.3;C: ALTER SPIN AND DY
5.70 I (X+CW-PR) 6.8;C: SAME FOR RIGHT RACKET
5.80 I (DX) 4.2
5.90 I (Y+CY-RY) 6.8

6.10 I (RY+PH-(Y-CY)) 6.8
6.20 S S=((R1-RY)+S)/2;S DY=DY+S;S S=R1-RY
6.30 S DX=-DX;C: REFLECT X VELOCITY
6.40 G 4.2;C: THEN GO BACK FOR NEXT INCREMENT
6.50 I (SL-(X-CW)) 6.8;C: IF BALL OFF SCREEN LEFT,
6.60 S RT=RT+1;D 9;S X=SL;S DX=V;C: SCORE PT
6.70 I (RT-GP) 7.15,7.4,7.4;C: AND CHECK FOR GAME OVER
6.80 I (X+CW-SR)4.2;C: CHECK FOR POINT FOR LEFT
6.90 S LT=LT+1;D 9;S X=SR;S DX=-V;C: ADD 1 IF SO

7.10 I (GP-LT) 7.4,7.4
7.15 X FCRT(-B,4);C: ON A POINT ERASE THE BALL WITH EOF
7.20 F L=1,1,25;X FTIC(-1);D 12;C:  WAIT CHANGING Y
7.30 G 4.2;C: THEN HAVE BALL COME IN FROM SIDE IT WENT OUT
7.40 X FCRT(B,500*SC,600*SC);C: IF GAME OVER, WRITE IT OVER BALL
7.45 S L=FCRT(B,300*SC,600*SC)
7.50 S L=FCRT(B+1,0G);S L=FCRT(L,0A);S L=FCRT(L,0M);S L=FCRT(L,05)
7.60 S L=FCRT(L,0BL);S L=FCRT(L,0O);S L=FCRT(L,0V)
```

```
 7.70 S L=FCRT(L,05);S L=FCRT(L,0R)
 7.80 A "PLAY AGAIN? ",L
 7.90 I (L-0YES)8.3,8.1,8.3

 8.10 X FCRT(-B,4);X FCRT(-B-1,4);C: ERASE 'GAME OVER'
 8.20 G 2.6;C: AND GO ON TO NEXT GAME
 8.30 X FTIC(5,0);Q;C: WHEN NO MORE GAMES, TURN CLOCK OFF, QUIT

 9.05 C: THIS SECTION PUTS UP THE SCORE
 9.10 S D1=FITR(LT/10);C: GET TWO DIGITS
 9.20 S D2=LT-(D1*10)
 9.30 X FCRT(LS+1,D1+48);X FCRT(LS+2,D2+48);C: INSERT AS ASCII
 9.40 S D1=FITR(RT/10);C: SAME FOR RIGHT HAND SCORE
 9.50 S D2=RT-(D1*10)
 9.60 X FCRT(RS+1,D1+48);X FCRT(RS+2,D2+48)

10.05 C: THIS SECTION CHECKS FOR BOUNCE OF FLOOR OR CIELING
10.10 I (Y+CH-ST) 10.3;C: IF BALL ABOVE CIELING,
10.15 I (DY) 10.3;C: AND GOING UP
10.20 S DX=DX-(S*.5);G 10.5;C: ADD SPIN TO DX AND GO CHANGE DY
10.30 I (SB-(Y-CH)) 10.6;C:IF BALL HIT FLOOR,
10.35 I (-DY) 10.6;C: AND GOING DOWN
10.40 S DX=DX+(S*.5);C: ADD SPIN TO DELTAX
10.50 S DY=-DY*.9;S S=S*.5;C: REFLECT DY AND DECREASE SPIN
10.60 R

11.05 C:THIS ECTION DRAWS THE BALL
11.10 X FCRT(B+1,0O);C: BALL IS LETTER O
11.20 X FCRT(-B-2,4);C: WRITE EOF AFTER IT
11.30 R

12.05 C: THIS SECTION MOVES THE BALL THRU Y AND MOVES THE PADDLES
12.10 D 4.3;D 10;D 4.5;D 4.6;D 4.7
```

This example requires that two users supply A/D input by manipulating the knobs for A/D channel 0 and 3. The program uses the values input to set the base of two short vertical lines one at either edge of the screen. The lines act as paddles. A moving character 'o' which simulates a ball will bounce off these paddles as well as bouncing off the top (ceiling) and bottom (floor) of the screen. The ball picks up spin when it bounces off a moving paddle and will react to subsequent impacts accordingly. The ball loses its spin each time it hits the ceiling or floor. The amount of spin it gets depends upon the speed of the paddle at the moment of impact. When the ball gets past one of the paddles and reaches a wall it scores a point for the opposing player. This point registers by incrementing one of the scores displayed near the top of the screen. When started, the program asks a number of questions. It needs to determine whether the system includes an AR11 or an LPS so it can set the correct screen scaling. (The LPS screen is scaled to four times that of the AR11.) It also asks the user to input a value that determines the average speed of play. The value input sets the ball's initial incremental movement along the x axis.

Note how an invisible point at the lower left corner of the character 'o' is continually altered to change the character's location. The paddles, which consist of relative points preceeded by an absolute point, are moved similarly.

8.2.3.5 Saving and Restoring the Display File

A program may spend a considerable amount of time performing the calculations it uses to create an image. The image it creates may, also, reflect experimental data taken during the program's execution and be impossible to reproduce. In these and similar cases it would be convenient to save the display, once created, in a file on disk so that, at some later time, it can be restored without performing all the original calculations or data aquisitions.

A graphics application may, also, require the rapid display of a series of complete images. Creating each image separately, that is, loading each LOC of the display file, requires far more time than pulling a completed image from an RT-11 file and loading it directly into the display file. If a program can create a series of images and write them one after another to the disk, then, when the time comes to rapidly display the images, it can just retrieve and display them one by one at high speed.

FFRM(N,M)

FFRM allows the FOCAL program to save the entire display file in an RT-11 file opened with the Library Make or Open command and to retrieve a complete display file from a previously saved RT-11 file. Each file can contain one or several display files, each called a frame and accessible individually.

FUNCTION FORMAT: FFRM(N,M)

The first argument to FFRM specifies an RT-11 channel number. When the FOCAL program executes FFRM, the channel specified must have been opened with a Library Open or Library Make command. The second argument of FFRM specifies a frame number. Each RT-11 file written by FFRM can contain several display files, each called a frame. The length in blocks of each frame in the file equals the length in blocks of the display file it contains. Under RT-11, the length of any display file is always a multiple of 256 words, 1 block. The display file created by FCRT(0,500), for instance, is four blocks long and contains 512 LOCs (1024 words). The entire four blocks constitutes one frame. A frame number argument greater than zero means output of the frame specified; less than zero means input of the frame specified by the argument's absolute value. On output FFRM calculates which blocks of the opened file correspond to the frame specified and starts writing the display file there. Thus, for a display file four blocks long, FFRM(0,3) starts writing the display file at RT-11 file block 8. On input a similar calculation determines from what blocks of the RT-11 file to load the display file. Note that on input the length of the current display file must equal the length of the display file used when the RT-11 file was created. Otherwise, input will use a different frame length (that of the current display file) in its calculations and result in a different block offset (except for frame 1 which starts at file block 0).

Group one of the example below saves a series of twenty images. Each image is one circle out of a series of concentric circles each consisting of 127 points. The display file contains one block (128 LOCs) so that the completed RT-11 file which contains twenty frames is twenty blocks long.

The user first runs section 1 by typing 'D 1' to create the file. The program draws each circle and saves it as a separate frame in the opened file. Next the user types 'D 2' to run section 2. Section 2 first allocates a one block long display file then starts up the display so that anything loaded into the display file will cause an image to appear on the screen. Next it executes a 'FOR' loop which rapidly retrieves each circle from the RT-11 file. This causes a ripple or explosion effect on the screen.

FOCAL can load one block long frames at the rate of approximately 24 frames per second. It can load two block long frames at half this rate, that is, 12 frames per second, 3 block frames at one third this rate and so on.

Example:

```
  1.05 C: PROGRAM EXPLODE
  1.10 L MAKE 0,FILE.DAT/I/V:D(0);C: CREATE AN RT11 FILE
  1.20 X FCRT(0,100);X FTIC(5,1);X FCRT(1);C: START DISPLAY
  1.40 F I=1,1,20;D 3;C: DO 20 FRAMES
  1.80 L CLOSE 0;C: CLOSE THE COMPLETED FILE

  2.10 L OPEN 0,FILE.DAT/I/V:D(0);C: OPEN THE FILE OF FRAMES
  2.20 X FCRT(0,100);X FTIC(5,1);X FCRT(1);C: START THE DISPLAY UP
  2.30 F J=1,1,10;F  I=1,1,20;X FFRM(0,-I);C: DISPLAY ALL FRAMES 10 TIMES
  2.40 L CLOSE 0;C: CLOSE THE FILE

  3.05 C: THIS SECTION DRAWS A CIRCLE IN THE DISPLAY FILE
  3.06 C: AND WRITES IT AS A FRAME IN THE OPEN FILE
  3.10 S TH=0;S R=I*60;S C=2000
  3.20 F J=1,1,127;S TH=TH+.0494;X FCRT(J,C+(FSIN(TH)*R),C+(FCOS(TH)*R))
  3.30 X FFRM(0,I)
```

8.2.3.6 Accessing the Display Status Register from Focal

The FOCAL program that uses the FCRT function to produce a display on a refresh-type scope need never directly access the display status register. Certain FOCAL users may, however, wish to alter the bits in the high byte of the display status register. These bits set the display mode. The FX function enables loading of the display status register directly by the FOCAL program. FCRT, when called to allocate/deallocate display file, sets the status register's high byte to 0. No other calls to FCRT alter the setting in this byte. Therefore, once the display file has been allocated, the FOCAL program can use FX to modify the high byte of the display status register. This modification will remain in effect until the display file is again allocated.

The standard display status register address for the LPS is @170416 and for the AR11 is @170410.

The high byte of the display status register allows the following settings:

BIT	NAME	MEANING AND OPERATION
8*	COLOR(VR20)	=0, GREEN =1, RED
9**	CHANNEL(VR14)	=0, CHANNEL 1 =1, CHANNEL 2
10	STORE	=0, INTENSIFIED POINTS NOT STORED =1, INTENSIFIED POINTS STORED
11	WRITE THRU	=1, INTENSIFIED POINT WILL NOT BE STORED EVEN THOUGH THE USER IS IN THE STORE OPERATION
12	ERASE	=1, ERASE DATA IN THE STORAGE SCOPE
13-15	UNUSED	

*LPS only
**Bit 9 may be used for other control function when the CRT is not a VR14 or VR20.

Bits 10, 11, and 12 refer to storage scope operations. The FCRT function normally operates assuming a refresh-type scope. To use FCRT with a storage scope, the FOCAL program may simply set the storage scope to refresh mode by setting bit 11 in the display status register after allocating the display file.

Example:

```
        1.10 X FPRM(12,20;X FCRT(0,100);C: ALLOCATE FILE
        1.20 X FX(-1,@170417,@10);C: SET WRITE THRU BIT IN LPS STATUS
```

Alternatively the user can leave the storage scope in store mode and proceed as for a refresh scope when creating an image, setting the erase bit in the status register to erase an image.

Example:

```
        1.10 X FCRT(0,100);X FTIC(5,1);C: ALLOCATE AND START CLOCK
        1.20 X FCRT(1);X FTIC(5,1);C: START UP THE DISPLAY
        1.30 X FCRT(1,100,100);C: SET THE BEAM TO (100,100)
        1.40 F I=1,1,10;X FCRT(I,I);C: PLOT 10 CHARACTERS THERE
        1.50 D 5;C: GO PERFORM SOME OTHER PROCESSING
        1.60 X FX(-1,@170410,@10);C: SET ERASE BIT TO ERASE THE IMAGE
        1.70 F I=1,1,20;X FCRT(I,+5,+5);C: DRAW A LINE ON THE SCREEN
```

This example starts up the display and loads the display file with character instructions. This causes the display processor to continually send the points that comprise the characters to the storage scope which displays them. After the first pass through the display file the data sent to the scope is redundant since it specifies points already plotted on the scope. The program performs some processing then just sets the erase bit in the status register of the AR11. The storage scope then erases the points it had previously plotted.

The second method, though it makes use of the flicker free storage facility, continually causes loading of redundant points. This uses up valuable processor time. Displaying points on the storage scope really requires only one pass through the display file. After one pass, the FOCAL program may turn off the display and continue other processing. To execute one pass though the display file, the FOCAL program can turn the clock off, turn the display on by calling FCRT(1), then cause the clock to tick once using FTIC with a negative second argument. The number of ticks per second specified by the 1st and second arguments to this call to FTIC should cause an interval between ticks at least as long as the interval between images. For example, if the FOCAL program has set parameter 13 to 5 producing an interval of 1/20th of a second between images, executing FTIC(5,-5) will cause the clock to tick once and the display processor to execute one display pass. Once the display processor has executed one pass, causing an image to appear permanently on the scope, the program can turn the display off using FCRT(0).

Example:

```
 1.10 X FCRT(0,100);C: ALLOCATE DISPLAY FILE
 1.20 F I=1,1,100;X FCRT(I,I*10,I*10);C: LOAD THE FILE
 1.30 X FX(-1,@170417,@4)
 1.40 C: SET LPS DISPLAY STATUS REGISTER FOR STORE MODE
 1.50 X FCRT(1);C: GET DISPLAY PROCESSOR READY
 1.60 X FTIC(5,-100);C: TICK ONCE CAUSING STORAGE OF DATA
 1.70 D 5;C: DO SOME OTHER PROCESSING
 1.80 X FX(-1,@170417,@20);C: SET FOR ERASE
```

This example plots a diagonal line on the storage scope by setting store mode and causing a single display pass. After doing some other operations it erases the line by setting the erase bit.

8.2.4 Loading the LEDS

The numeric display on the LPS consists of six LED matrices. Using the left most LED for the sign, FOCAL can display any value between -99999. and 99999.

FLED(N,M)

The argument to FLED contains the floating point value to display. The first argument indicates the number of digits to display to the right of the decimal point. Since FOCAL uses the left most character position of the LED display for the sign of the value being displayed, the arguments to FLED must specify a value requiring at most five digits. Otherwise an overflow error will occur.

Example:

```
        X FLED(3,-99.9534)
```

This example displays '-99.953'

Example:

```
        X FLED(4,123.5)
```

This example causes an error since it requires 7 digits.

8.2.5 Rapid Function Execution

In processing a series of Set or Xecute commands FOCAL spends an appreciable amount of time getting from one to the next. When these commands cause evaluation of functions that perform some external operation like sampling an A/D channel, the program may wish to determine the time between their execution more precisely. For example, the program may wish to determine the exact time an A/D operation began without including the delay caused when FOCAL begins to evaluate the function that returns the time.

FFNS(F1,F2,...)

The arguments to FFNS are usually functions that perform external operations. FFNS evaluates these arguments one after another causing execution, in rapid succession, the external operations they perform. The value returned by FFNS is that obtained from the last function executed by FFNS. Executing FFNS by this method saves 1 to 2 milliseconds per function execution over executing the same functions with Xecute commands.

Using FFNS the FOCAL program can, for example, cause rapid execution of a series of events and, by making a time returning function the last argument, return the time of the last of these events.

Example:

```
        S T=FFNS(FSAM(100,0),FTIC(0))
```

This example initiates sampling from channel 0, then obtains the AR11/LPS timer value and types it.

The FOCAL program can also use FFNS to cause exact time delays between execution of functions by placing a wait function between arguments.

Example:

```
        X FFNS(CHR(I),FTIC(-6),FCHR(J),FTIC(-6),CHR(K))
```

This example outputs three characters with 6 clock ticks between each output.

8.2.6 16-bit Logical Operations

Particularly when using the UNIBUS function, FX, the FOCAL program may need to perform the 16-bit logical operatons AND, OR and XOR (eXclusive OR), and also, to access individual bits in a word. These operations are particularly useful when setting, clearing or testing bits in words to be output or input.

FBIT(V,C1,V1,C2,V2,...)

FBIT performs a series of logical operations on the 16-bit value specified as its first argument. The following even numbered arguments each contain a code interpreted modulo 4 which specifies a logical operation. The arguments following each code specify the operator for this operation. The first argument contains the original operand. Each time a logical operation is performed upon it, the result of the operation becomes the new operand. In this way a series of logical operations can be performed and the result returned as the function's value. FBIT allows 3 logical operations:

CODE	OPERATION
1	OR
2	AND
3	XOR(EITHER/OR BUT NOT BOTH)

In addition, negative codes complement the first operand before performing the operation specified by the magnitude of the code.For example, -1 first complements the value specified in the argument following the code argument, then peforms an 'OR' between the complemented value and the current operand.

To perform the logical operation 'NOT', the FOCAL program can use code -1 with an operand value of 0: FBIT(0,-1,V) returns the complement of the value specified by V.

Example:

```
   S V=FBIT(I,1,@377);C: RETURN THE LOW BYTE OF I

   S V=FBIT(I,-2,J);C: RETURN ALL BITS SET IN I AND NOT SET IN J

   S V=FBIT(I,3,@177777,1,@377);C: COMPLEMENT THE VALUE IN I, THEN
   C: SET ALL BITS IN ITS LOW BYTE
```

FBIT also allows for a bit oriented operation requested by a code of zero. The value following code 0 represents a bit number between 0 and 15 (the rightmost bit is bit number 0). A code value of zero causes this bit to be set to 1 in the operand.

Example:

```
   S V=FBIT(I,0,5);C:  SET BIT 5 IN THE 16 BIT VALUE CONTAINED IN I
```

The program uses code 0 with an operand (first argument) of 0 to create a mask word. FBIT(0,0,5,0,4,0,3), for instance, returns a mask word with bits 3, 4, and 5 set.

The FOCAL user who wishes to manipulate individual bits in 16-bit words can think of the operations that FBIT performs in terms of bit setting, testing, complementing, and clearing. To use FBIT this way, the program uses code 0 to create a mask word, then uses the codes in the following table to perform an operation between the mask word and the operand.

CODE	BIT OPERATION
1	BIT SET
2	BIT TEST
3	BIT COMPLEMENT
-2	BIT CLEAR

The operations listed in the table above apply to all bits in the operand specified by the mask word.

BIT SET returns a value with the specified bits turned on in the operand.

BIT TEST returns a non-zero value if the specified bits match any bits set in the operand.

BIT COMPLEMENT returns a value with the specified bits in the operand having had their state change; on bits turned off, and off bits turned on.

BIT CLEAR returns the value of the operand with the specified bits cleared.

In the next example, the FOCAL program creates a mask word with bits 5 and 9 set, then clears the specified bits in a value which it loads into register @170400.

Example:

```
        2.10 S M=FBIT(0,0,5,0,9);C: CREATE THE MASK WORD
        2.20 S V=FBIT(V,-2,M);C: CLEAR BITS 5 AND 6 IN A VALUE
        2.30 X FX(-2,@170400,V);C: LOAD THE VALUE INTO A REGISTER
```

8.3 THE DR11-K

NOTE

The methods discussed in this section apply equally to the DR11-C and the LPS with the LPSDR-A digital input/output option.

The DR11-K input-output interface provides three registers used in performing I/O operations. The DR11-K status register allows the program to enable output and/or input interrupts. The output register allows output by byte or word. The input register allows 16 bit parallel input.

The FOCAL program outputs data by loading a word or a byte into the 16 bit output register with or without output interrupt enabled. If output interrupts are enabled, receipt of the data by the external device will cause an interrupt.

Data is input by reading the 16-bit input register. Input interrupts may be caused by the setting of any bit in this register or by two control lines between the DR11-K and the external device or devices. A hardware setting determines which method will cause interrupts. The program can clear bits in the DR11-K input register by moving a one to the bit to be cleared. The input interrupt enable bit of the status register is cleared when the interrupt is accepted. The resetting of this enable bit will retrigger the program service subroutine. Therefore, new data bits will not be lost.

There are no FOCAL functions specifically designed for the DR11-K. However, function FX, the Unibus function, in conjunction with the interrupt scheduling function, FINT, the time of interrupt function, FTOI, and the bit manipulation function, FBIT, give the FOCAL program complete access to all the DR11-K's capabilities. The FOCAL program needs to know the addresses of the input and output registers and of the interrupt vector of each DR11-K on the system. The standard addresses selected for the DR11-K and the LPSDR (in parentheses) are:

167770 (170410)	STATUS REGISTER ADDRESS
167772 (170412)	INPUT ADDRESS
167774 (170414)	OUTPUT ADDRESS

The standard vector addresses for the DR11-K are:

300 (350)	INPUT VECTOR ADDRESS
304 (354)	OUTPUT VECTOR ADDRESS

The meaningful bits in the status register are:

BIT	USE
6	INPUT INTERRUPT ENABLE
7	INPUT FLAG (DATA HAS BEEN LOADED INTO THE INPUT REGISTER)
14	OUTPUT INTERRUPT ENABLE
15	OUTPUT FLAG (DATA IN OUTPUT REGISTER HAS BEEN ACCEPTED)

8.3.1 Output Operations to the DR11-K

The FOCAL program can output to a DR11-K using function FX by byte or by word. FX(-2,@167770,I), for instance, outputs the value in I, taken as a 16-bit quantity. FX(-1,@167775,@377) outputs the octal byte 377 to the high byte of the output register.

The FOCAL program may use the function FBIT to create the word or byte to output. If, for example, the FOCAL program must output a word with bits 2 and 4 set as flags to the low byte of the DR11-K, FX(-1,@167774,FBIT(0,0,2,0,4) will do so.

If the FOCAL program must ascertain that the external device has received the data loaded into the output register, it can do so by using FX to enable output interrupts (setting bit 14 of the status register) and scheduling a FOCAL interrupt processor using FINT.

Example:

```
1.10 X FX(-2,@167770,FBIT(0,0,14));C: ENABLE OUTPUT INTERRUPTS
1.20 X FINT(@304,5,4,@167770,0);C: SCHEDULE GROUP 5
1.30 X FX(-2,@167774,@17345);C:OUTPUT A WORD
1.40 S DFLG=0;CLEAR OUTPUT DONE FLAG
1.50 G 2;C: CONTINUE PROCESSING

5.10 S DFLG=1;C: SET A FLAG SAYING THE OUTPUT HAS COMPLETED
5.20 RETURN;C: RETURN TO INTERRUPTED LINE
```

8.3.2 Input Operations

To perform input the FOCAL program can wait for the input flag in the DR11-K status register to set. Alternatively the program can enable the DR11-K input interrupt by setting the input interrupt enable bit (bit 6) in the status register and setting up a FOCAL interrupt processor using FINT. The next example uses the first method to create an array of DR11-K input values. After each input it loads the value read back into the input register to clear the register for the next input.

Example:

```
1.10 X FX(-2,@167770,0);C: SET STATUS REGISTER, NO INTERRUPTS
1.20 S J=0;C: CLEAR A COUNTER
1.30 S I=FX(0,@167772,@100);C: GET INPUT READY BIT
1.40 I (I-@100) 1.3;C: WAIT FOR IT TO SET
1.50 S K=FX(-2,@167772,FX(2,@167772))
1.60 C: READ DATA AND CLEAR INPUT REGISTER
1.70 S J=J+1;S D(J)=K;C: STORE THE DATA IN THE NEXT ARRAY ELEMENT
1.80 I (K-@15) 1.3,1.9,1.3;C: QUIT WHEN <CR> RECEIVED
```

The next example sets up an interrupt processor to perform the same storage operation while other processing continues.

Example:

```
1.10 X FINT(@302,5,4,@167770,0);C: SET GROUP 5 FOR INTERRUPTS
1.20 X FX(-2,@167770,FBIT(0,0,6));C: ENABLE INPUT INTERRUPTS
1.30 G 2;C: CONTINUE PROCESSING

5.10 S K=FX(2,@167772);C:GET THE DATA
5.20 X FX(-2,@167772,K);C: CLEAR THE INPUT REGISTER
5.30 S J=J+1;S D(J)=K;C: SAVE THE INPUT DATA
5.40 X FX(-2,@167770,FBIT(0,0,6));C: RE-ARM THE INTERRUPTS
5.50 RETURN;C:RETURN TO INTERRUPTED LINE
```

It is important to note when processing input interrupts from a DR11-K connected to many separate devices, that by the time the FOCAL program initiates the interrupt processing routine in response to an input interrupt from one of these devices, one or more of the remaining devices may have caused other bits in the input register to set. In this environment the interrupt processor should check for input from all the devices. It must, of course, know which device is connected to which bit, or bits, in order to perform the correct processing for each bit set.

In the next example, eight devices are connected one each to bits 0 through 7 of the DR11-K input register. The interrupt processor checks each bit and performs the necessary operations if the bit is set.

Example:

```
5.10 S K=FX(2,@167772);C:GET THE DATA
5.20 X FX(-2,@167772,K);C: CLEAR THE INPUT REGISTER
5.30 F I=0,1,7;D 6;C: DO GROUP SIX FOR EACH OF 8 BITS
5.40 X FX(-2,@167770,@100);C: THEN RE-ARM THE INTERRUPTS
5.50 RETURN;C:RETURN TO INTERRUPTED LINE

6.10 S B=FBIT(K,2,FBIT(0,0,I));C: TEST ONE BIT IN THE INPUT WORD
6.20 I (B-1) 6.5;C: RETURN IF BIT NOT SET
6.30 D 7+B;C: OTHERWISE DO THE GROUP NUMBER THAT HANDLES THIS
6.35 C: BIT FOR PROCESSING, THEN RETURN
6.40 Q
```

CHAPTER 9

FUNCTIONS FOR USING THE VT-11

9.1 INTRODUCTION

The VT11-A is a high-performance display system that operates with a PDP-11 computer. It is designed for applications that require both a visual display and a computation capability. The system can display either alphanumeric information, graphics data such as drawings, diagrams, and patterns, or any combination of these. It is particularly valuable for displaying dynamic data.

The VT11's display processor retrieves display data and commands from the PDP-11's memory. It decodes and executes this information and carries out vector and character calculations that are required by the CRT for display presentations. The CRT display is a self contained unit that provides a 9.25 by 9.25 inch viewing area consisting of 1024 by 1024 rasters. The display is an automatically refreshing type rather than the storage type so that a bright, continuous image, with excellent contrast ratio, is provided during motion or while changes are being made in the elements of the picture. The VT11 includes a light pen for interactive graphics. The light pen is a pencil shaped light detector for use in a wide range of interactive applications. The VT11 character generator has both upper and lower case capability with a large repertoire of display characters. In addition to the 96 ASCII characters, 31 special characters are included. These special characters include some greek letters, architectural symbols and math symbols. Scope resolution is precise enough to allow overprinting. eight intensity levels permit the brightness and contrast to be varied so that the scope can be viewed in a normally lighted room. A hardware blink feature is applicable to any characters or graphics drawn on the screen.

Under RT-11 the VT11 may be used for display of terminal I/O. With this feature in operation, FOCAL's terminal interaction will also display on the VT11 screen at 31 lines of 72 characters each. The functions described in this chapter allow FOCAL to also make use of the VT11's graphics capabilities. Graphics may be used with terminal I/O displayed on the screen or on the console terminal.

The functions allow the FOCAL program to create an addressable display file and to fill it with graphic data. The graphic data can take the form of lines, points, and characters. Lines are defined by a pair of coordinates that indicate the offset of the line's endpoint from its starting point. Points are defined as fixed locations on the screen. Characters display as italics or upright print horizontally across the screen. Lines and points may be displayed visibly or invisibly. Lines can be displayed as one of four separate line types: solid, long dash, short dash, and dot dash. All graphics can be displayed at one of 8 intensity settings, as blinking or non-blinking, and as light pen sensitive or non-light pen sensitive. Light pen sensitive graphics when touched with the VT11's light pen may be used by the FOCAL program to automatically alter the location of part of the display. The FOCAL program may also obtain the coordinates of the light pen hit and the location in the display file of the graphic element upon which the hit occurred. The program can also wait for the next light pen hit to occur.

Additional functions allow the FOCAL program to discontinue display of some or all data in the display file, to clear part or all of the display file, to discontinue the display, and to alter the number of lines, the intensity and the top location of the terminal I/O displayed on the screen.

Together the VT11 functions give the FOCAL program the ability to create an image on the screen and to alter any part of it. By continually altering the starting point of an image, the FOCAL program can cause it to move around the screen. By drawing a number of images, each with a separate starting point and continually altering each of the starting points, the program can cause several separate pictures to move independently of each other around the screen. By altering the lines that compose a given image, the program can change its shape and its size. By accessing the light pen, the program can allow the user to interactively alter the image. The automatic tracking feature of the FOCAL light pen handler allows high speed tracking of any tracking element the user might construct.

In short, the VT11 functions give the user a set of powerful tools by means of which a FOCAL program can create a complex graphic display on the VT11. The next sections describe the functions in detail.

9.2 THE FUNCTIONS

A FOCAL program that does graphics employs the graphics functions along with all the other statements available to FOCAL. Rather than simply storing its calculations in a file or typing them out for the user to read, the program can convert them to a graphic form and display them on the screen. A simple graphics program might look something like the one below.

```
1.05 X FSCR(5,4,740)
1.07 A "TYPE <CR> TO START",A!!
1.10 X FVT(0,200)
1.15 X FSKP(100,-1)
1.20 S X=500;S Y=500
1.30 S LOC=1
1.40 F TH=0,.1,6.28;D 2
1.41 S LT=LOC
1.42 S LOC=FSET(LOC,450,350)
1.44 S LOC=FTXT(LOC,0R,0A,0D,0I,0A,0N,0S)
1.46 S LOC=FTXT(LOC,0BL,50);S LOC=FSPC(LOC,16)
1.48 S LOC=FTXT(LOC,61,51,54,48);S LOC=FSPC(LOC,11)
1.50 X FVT(1)
1.60 X FSET(LOC,500,500)
1.70 F TH=0,.1,6.28;D 3
```

```
1.75 X FMOV(100,0,0)
1.79 D 4;D 6
1.80 A "TYPE <CR> TO EXIT",A
1.90 X FCLR(0,LOC+2)
1.95 X FSCR(30,4,740)
1.99 Q

2.10 S DX=FCOS(TH)*100;S DY=FSIN(TH)*100
2.20 S LOC=FPT(LOC,X+DX,Y+DY)
2.30 I (FITR(TH+.09)-TH+.09)2.5,2.4,2.4
2.40 S LOC=FTXT(LOC,48.1+TH)
2.45 X FDIS(-1,TH+.09,-1,1)
2.50 R

3.10 D 2.1
3.15 X FDIS(2,4,0,1)
3.20 X FVEC(LOC+1,DX,DY)

4.10 A "BY WHAT VALUE TO MULTIPLY THE RADIUS",P
4.20 F I=1,LT;D 5

5.10 S DX=FXCO(I);S DY=FYCO(I)
5.20 I (DX) 5.5
5.30 S DX=(DX-500)*P;S DY=(DY-500)*P
5.35 X FDIS(FDIS(I))
5.40 X FPT(I,DX+500,DY+500)
5.50 R

6.05 T "HIT THE CIRCLE WITH THE LIGHT PEN",!
6.10 S L=FLP(0)
6.20 S DX=FXCO(L)-500;S DY=FYCO(L)-500
6.25 X FDIS(2,5,0,0)
6.30 X FVEC(LOC+1,DX,DY)
6.40 S R=(L-1)*.1
6.50 T R," RADIANS",!!
```

The example makes use of all the graphics functions available to FOCAL. These functions are now described in detail using lines from the program above for most examples. Unless otherwise indicated all functions take integer arguments in the range -32568 to +32567 and will cause an integer overflow error (?23) on arguments outside this range. The section describing each function begins with a line portraying the function in its most general form.

9.2.1 Setting Up the Display File

FVT(N[,M])

Function FVT can perform two operations: it creates or eliminates the display file and turns the display on or off. When called with a single argument greater than 0, it turns the display on. Turning the display on consists of directing the VT11 display processor to display all graphic data loaded into the display file. An argument less than or equal to 0 turns the display off. The VT11 display processor will then ignore the graphic data in the display file and display only terminal I/O on the VT11 screen provided that RT-11 terminal I/O has been directed to the screen by the 'GT ON' command. Turning the display on or off has no effect if no display file has been created yet. And turning the display on with no data in the display file will cause no image to appear on the screen.

To create the display file the FOCAL program calls FVT with two arguments. The first argument specifies whether to turn the display on or off as it does for FVT with one argument. The second argument indicates the size of the display file to allocate. If the second argument is 0, the current display file will be eliminated and the first argument will have no effect. If the second argument is greater than 0, it will cause elimination of any current display file and creation of a new display file. The argument's value indicates the number of slots available for receipt of graphic data. Each such slot is called a LOC. Each LOC can contain either one line, one point, four characters, a display jump, or a display null (described in later sections). Each LOC has a number associated with it that indicates its offset from the start of the display file and by means of which it can be addressed for loading. The first LOC in the file is LOC 0, the second LOC 1, and so on up to the last LOC. The second argument to FVT determines the total number of such LOCs in the display file. In terms of PDP-11 memory, each LOC requires 3 words. FVT will allocate enough 256 word blocks of memory to provide the requested number of LOCs and will return as its functional value the number of LOCs allocated. This value may be greater than the number of LOC's requested since it is the number of LOCs available in the total number of blocks required. For example, the function FVT(1,100) requests 100 LOCs. This requres 300 words of memory plus 2 words as the display file delimeter, thus FVT must allocate two 256 word blocks, 512 words, to satisfy the request. The 512 words make room for 170 LOCs, and FVT returns as its functional value 170.

If the request requires too much memory, the call to FVT will result in a ?23 error.

Example:

```
FVT(1,100);C: ALLOCATE A 100-LOC DISPLAY FILE;TURN DISPLAY ON

FVT(0,0);C; DE-ALLOCATE THE CURRENT DISPLAY FILE

FVT(1)C: TURN THE DISPLAY ON

FVT(0);C: TURN THE DISPLAY OFF
```

Once the display file has been set up using FVT, it can be loaded with graphics data. When the display is on, any data the display processor encounters in the display file will be displayed on the screen in the order it is encountered. Thus, the graphic datum in LOC 0 will be the first displayed, then the data in LOC 1 and so on. If no data has been loaded into a given LOC, the display processor will ingore it and go on to the next LOC. Thus, if LOC 5 contains a line, LOCs 6,7, and 8, have not been loaded, and LOC 9 contains another line, the line in LOC 9 will display immediately following the line in LOC 5. If a line is subsequently loaded into LOC 6, this line will display between the two lines in LOCs 5 and 9. The data contained in the display file determines the image displayed on the screen.

The next sections describe the types of graphic data that the FOCAL program may load into the display file.

9.2.2 Loading the Display File with Lines and Points

The FOCAL program can create images consisting of straight lines (vectors) and points. Lines are displayed "relatively." This means that the starting point of each line is the point on the screen where the previously displayed line or point ended (called the current screen location) and the ending point of the line is an offset from this point. Points are displayed "absolutely." This means that points display at a fixed screen location that does not depend on the current screen location produced by the graphic data in the previously displayed LOC. Points and lines can be displayed visibly or invisibly. The coordinates of lines and points are interpreted relative to a screen scaled from (0,0) at the lower left hand corner to (1023,1023) at the upper right.

Each line or point occupies a single LOC in the display file. The FOCAL program can load a line or point into any LOC in the file at any time, overwriting and destroying any data the LOC may have contained.

The format of the four functions that cause loading of lines and points is the same. The first argument indicates the number of the LOC to load. The next two arguments indicate the x and y coordinates of the line or point the LOC will contain. Every function will load the LOC indicated and return the value of the first argument plus one. The FOCAL program may use this returned value as the first argument to the next function that loads a LOC. This simplifies sequential loading of LOCs. If the LOC indicated by the first argument is greater in magnitude that the highest numbered LOC in the display file, the highest LOC will be loaded and the negative of the magnitude of the highest LOC will be returned. Thus, if the display file contains 170 LOCs and the program attempts to load LOC 200, the loading function will return the value -200 and load LOC 170. If the program attempts a load without having created a display file it will cause a ?09 error.

9.2.2.1 FVEC(LOC,X,Y)

FVEC loads a visible vector into the LOC specified by its first argument. Its second two arguments determine the offset (x and y) of the endpoint of this vector from its starting point. Since all vectors display relatively, its starting point will be the current screen location as produced by the datum previously displayed. A vector will leave the current screen location at its own endpoint.

Example:

```
1.10 X FVT(1,100);C:ALLOCATE 100 LOCS;TURN DISPLAY ON
1.20 S LOC=0;C: SET POINTER TO FIRST LOC
1.30 F J=0,.5,40;S LOC=FVEC(LOC,10,J);C: LOAD 80 VECTERS
```

This example sets up a display file and loads it with 80 vectors to form a parabola on the screen.

9.2.2.2 FMOV(LOC,X,Y)

FMOV acts exactly like FVEC except that the line indicated will display invisibly. It will alter the current screen locations but otherwise not alter the image displayed, unless other visible vectors and characters are located in succeeding LOCs.

Example:

```
        1.40 F J=1,100;X FMOV(0,I,I)
```

This line, when added to the previous example, causes the parabola to move across the screen at a 45 degree angle. It does so by continually loading LOC 0 with an invisible vector of increasing length. This causes the starting point of the parabola to move.

9.2.2.3 FPT(LOC,X,Y)

FPT loads an absolute point into the LOC specified as its first argument. The second two arguments determine the x and y coordinates of the point. An absolute point will remain at the screen location specified regardless of movement of the remainder of the image.

Example:

```
        1.10 X FVT(0,200);C: ALLOCATE FILE,DISPLAY OFF
        1.20 S X=500;S Y=500;C: SET CENTER COORDINATES OF A CIRCLE
        1.30 S LOC=1;C: SET POINTER TO POINT TO LOC 1
        1.40 F TH=0,.1,6.28;D 2;C: DRAW A CIRLE IN 63 LOCS
        1.50 X FVT(1);C:DISPLAY THE CIRCLE

        2.10 S DX=FCOS(TH)*100;S DY=FSIN(TH)*100
        2.15 C: LOCATE DOT ON PERIMETER OF CIRCLE
        2.20 S LOC=FPT(LOC,X+DX,Y+DY);C: DRAW IT IN THE NEXT LOC
```

This example draws a circle consisting of 62 points with its center at (500,500).

9.2.2.4 FSET(LOC,X,Y)

FSET acts exactly like FPT except that the point loaded is not displayed. It can be used to set the current location on the screen but will otherwise not alter the image on the screen unless other visible vectors and points are located in succeeding LOCs.

Example:

```
        1.60 X FSET(LOC,500,500)
        1.61 C: DRAW AN INVISIBLE POINT AT CIRCLE'S CENTER
        1.70 F TH=0,.1,6.28;D 3;C: MOVE A CLOCK HAND AROUND THE CIRCLE
        1.80 Q

        3.10 D 2.1;C: LOCATE POINT ON CIRCLE'S PERIMETER
        3.20 X FVEC(LOC+1,DX,DY);C:  DRAW LINE FROM CENTER TO PERIMETER
        3.25 C: IN THE LOC FOLLING THE INVISIBLE POINT
```

These lines when added to those of the previous example cause a vector to move about the circle like the hand of a clock. It sets the starting point of the vector using an invisible point, then constantly loads the LOC that contains the vector with vectors that extend from the center of its circle to a point on the perimeter.

9.2.3 Loading Characters

VT11 characters are 16 rasters wide and 12 rasters high. Each character displays with its lower left hand corner at the current screen location and leaves the current screen location at its lower right hand corner. Therefore, characters sequentially loaded into the display file display horizontally across the screen.

9.2.3.1 FTXT(LOC,A,B,C,...)

FTXT loads characters into the display file starting at the LOC indicated in its first argument. Each display LOC can contain four characters. FTXT will take the next four arguments in its argument list, convert them to ASCII characters, and load them into the current LOC. If it encounters a fifth arugment, it will go on to the next LOC. It will continue loading LOCs sequentially each with four characters until it encounters the end of the argument list. At this time it will return the number of the LOC following the last LOC loaded. The conversion of argument values to ASCII characters maps the decimal arguments 0 to 32 to the ASCII characters of octal values 100 to 137 (characters 'A' through ']') and decimal arguments 33 to 127 as ASCII octal values 40 to 177. Thus, the FOCAL constants 0A through 0Z when used as arguments will cause display of the corresponding character. Due to its value of 32, the FOCAL constant 0BL will display as a blank.

Example:

```
        2.30 I (FITR(TH+.09)-TH+.09)2.5,2.4,2.4
        2.35 C: IF TH IS AN INTEGER VALUE NOW
        2.40 S LOC=FTXT(LOC,48.1+TH)
        2.45 C: WRITE ITS INTEGER VALUE IN THE LOC
        2.46 C: FOLLOWING THE CURRENT POINT
        2.50 R
```

These lines added to the previous example label the circumference of the circle with the angle in radians. They do this by loading the ASCII value for the current angle in radians into the LOC following each point that corresponds to a whole integer radian value.

Example:

```
        1.42 S LOC=FSET(LOC,450,350);C:SET POINT BELOW CIRCLE
        1.44 S LOC=FTXT(LOC,0R,0A,0D,0I,0A,0N,0S);C: LABEL THE IMAGE
```

These lines added to the example label the circle on the screen with the title 'RADIANS'.

9.2.3.2 FSPC(LOC,A[,B])

FSPC allows the FOCAL program to load an individual LOC with 1 or 2 special characters. The GT40 USER'S GUIDE contains a list of the VT11's special characters and their corresponding codes. The first argument to FSPC indicates which LOC to load. The next argument inidicates the code for a special character other than shift out (octal code 17). The optional third argument indicates the code for a second special character. FSPC loads the LOC with the the character or characters indicated and returns the value of the first argument plus one.

Example:

```
        1.46 S LOC=FTXT(LOC,0BL,50);S LOC=FSPC(LOC,16);C: ADD '2 PI'
        1.48 S LOC=FTXT(LOC,61,51,54,48);S LOC=FSPC(LOC,11)
        1.49 C: ADD '=360 DEGREES'
```

These lines added to the previous example use special characters in the label.

9.2.4 Inserting Display Jumps

Once the FOCAL program has loaded the display file it may need to temporarily discontinue display of some of the graphics loaded. It can do this, as well as ignore the terminal I/O, by loading individual LOCs with display jumps, jumps to somewhere in the display file. When the display processor encounters a LOC containing a jump, it uses the LOC specified in the jump as the next LOC to execute rather than executing the next sequential LOC. The jump can be forward or backward. It can be to a LOC within the display file, or to the end of the display file, or to its beginning.

FSKP(N[,M])

FSKP with one argument indicates a jump to the end of the display file. The LOC indicated by the first argument is loaded with this jump. All graphics in LOCs subsequent to the one loaded will disappear from the screen.

If FSKP has two arguments, the first indicates the LOC to load and the second indicates the LOC to jump to. Thus, FSKP(5,100) means load LOC 5 with a jump to LOC 100. Graphics in LOCs 6 through 99 will disappear from the screen. If the second LOC is beyond the display file, FSKP will insert a jump to the end of the display file. If the second LOC is a LOC preceeding the first LOC, FSKP will perform the load. However, the user must realize that such a jump creates an infinite loop for the display processor. The display processor will continually reach the jump and revert back to the earlier LOC. Thus, any terminal I/O displayed on the screen will not display. Any part of the display file not incuded in the loop will not display. If the second argument is a negative value, FSKP inserts a jump to the start of the display file. Terminal I/O will not be displayed.

Example:

```
        1.15 X FSKP(100,-1);C:JUMP TO START OF FILE TO IGNOR TTY I/O
        1.75 X FVEC(100,0,0);C: LOAD NULL VECTOR OVER SKIP TO
        1.77 C: RE-DISPLAY TTY I/O
```

Line 1.15 added to the example causes terminal I/O to disappear while the hand is rotating. Line 1.75 redisplays teletype I/O by loading a null vector (a vector of 0 length) over the display jump.

9.2.5 Choosing the Graphic Modes

The graphic data in each LOC has a set of modes associated with it which determine how it will display. Lines may be solid, short dash, long dash, or dot dash. All graphics may have one of eight intensity settings, may be blinking or non-blinking, and may be light pen sensitive or non-light pen sensitive.

FDIS(T,I,B,S)

The first word of each LOC (a LOC contains 3 PDP-11 words) specifies the "type" of graphic data the LOC contains as well as up to four graphic "modes" associated with the data. The type of data-vector, point, text, or jump is determined by the function that loads the LOC. FVEC, for instance, sets the first word to vector type data. The graphic modes- four of them- are determined by a call to FDIS. The four graphic modes determine line type, intensity, blink, and light -pen sensitivity of the graphic datum contained in the LOC. When the display processor encounters a LOC, it changes the current modes of display to the modes indicated in the LOC. It will display the graphic element in the LOC in the current modes and will apply these modes to all subsequently displayed LOCs until it encounters a LOC which again changes one or mode of the modes thus altering the current graphic modes.

The FOCAL program can call FDIS with 1 or 4 arguments. These arguments specify the graphics modes that will be loaded into the first word of the next LOC loaded by a graphics function. A call to FDIS does not alter the display; it only determines the mode of the next load and applies only to that load. Subsequent loads which have no preceeding call to FDIS, will have no associated graphics modes. The data in the LOCs they load will take on the modes of the previously displayed LOCs in the display file.

When the FOCAL program calls FDIS with four arguments, each argument specifies one of the four available graphics modes.

The first argument to FDIS determines the type of line:

FIRST ARGUMENT	LINE TYPE
0	SOLID
1	LONGDASH
2	SHORTDASH
3	DOTDASH

The second argument determines the intensity from 0 (dimmest) to 7 (brightest). The third argument determines the blink. 0 means steady; 1 means blink. The fourth argument determines the light pen sensitivity. 0 means non-light pen sensitive; 1 means light pen sensitive.

If any of the arguments are less than zero, then the corresponding mode will be left unspecified. For this unspecified mode, the datum in the next LOC loaded will take on the corresponding mode of the previously displayed LOC. This is the default condition for LOCs loaded with no prior call to FDIS. The LOC loaded will have no associated modes and will take on the modes of the previously displayed graphic element. The original modes, those which LOC 0 takes on by default, are solid lines, intensity 4, non-blinking, and non-lightpen sensitive.

Example:

```
        2.45 X FDIS(-1,TH+.09,-1,1)
        2.46 C: SET INTENSITY AND LIGHT PEN SENSITIVE

        3.15 X FDIS(2,4,0,0);C: SET LINE TO SHORTDASH
        3.17 C: NON-BLINK, LIGHT PEN SENSITIVE
```

These lines added to the example make the dots that compose the circle light pen sensitive, reflect in their intensity the number of radians, and change the clock hand to a short dash line.

When the FOCAL program calls FDIS with a single argument, the sign of the argument determines the operation the function will perform. For an argument greater than or equal to zero, the argument represents a LOC number. FDIS will, in this case, return as its functional value the actual contents of the first word of this LOC. Since the first word specifies the four graphics modes associated with the LOC, the value returned can be used to represent all four modes. This value is always negative. If it is supplied to FDIS as a single argument, FDIS will use it to determine all four graphics modes to associate with the next LOC loaded. In other words, the FOCAL program can specify the four graphics modes either by calling FDIS with four arguments, each specifying one mode, or by calling FDIS with one negative argument which specifies all four modes. FDIS's ability to return the modes of a LOC is especially useful when reloading a LOC with new coordinates. The FOCAL program can obtain the mode of the LOC using FDIS with one non-negative argument, specify these modes in a call to FDIS with one negative argument, then reload the LOC. The graphics modes associated with the LOC will then not change even though the data in the LOC has changed. The example in the section describing FXCO and FYCO makes use of this feature. The example uses the function recursively, FDIS(FDIS(I)), to at once obtain the modes of LOC I and set these modes for the next load.

9.2.6 Loading Display Characteristics

In addition to the four graphic modes set by using FDIS, the FOCAL program can determine three other characteristics of the display; sync, italics, and light pen intensify. The program determines these characteristics by loading a LOC with a display null which specifies them.

FSTA(LOC,C)

FSTA loads the LOC specified as its first argument with the three characteristics specified by its second argument. A LOC thus loaded is essentially a display null. It causes no new graphics to appear on the screen and does not alter the current screen location. It only determines the three characteristics as they apply to all subsequently displayed LOCs, until a LOC containing another change to the characteristics is encountered.

When set, the sync feature causes the display processor to pause and wait for the next cycle of the AC line voltage (i.e. 50 or 60 Hz.) before resuming display. Normally, each graphic element added to the display causes a slight decrease in overall intensity because the display processor has more commands to decode. With the sync feature enabled, intensity changes occur only after a large number of graphics instructions have been added. The intensity change is then a large change.

NOTE

The FOCAL program should not load more than one loc with sync on.

Loading a LOC with the italics feature enabled causes all characters in subsequent LOCs to display in italics. Loading a LOC with italics off sets characters back to upright.

Loading a LOC with the light pen intensify feature enabled causes light pen sensitive graphics in all subsequent LOCs to intensify at the point of a light pen hit thus making the location of the hit obvious to the operator.

The second argument to FSTA determines all three characteristics. Its value is determined by adding up a set of values, each value corresponding to one of the characteristics. A value not added sets the corresponding characteristic to the complementary state from the one indicated in the table below.

VALUE ADDED	CHARACTERSITIC
4	SYNC ON
16	ITALICS ON
64	LIGHT PEN INTENSIFY ON

Thus, a second argument of 20 (16+4) sets the sync and italics features on, and the light pen intensify feature off.

FSTA returns as its functional value the value of its first argument plus 1.

Example:

```
1.43 S LOC=FSTA(LOC,16+64);C:  SET ITALICS AND INTENSIFY
```

This line added to the main example causes all characters to display as italics and light pen hits to cause intensification.

9.2.7 Clearing the Display File

The FOCAL program can clear individual LOCs, that is, erase the graphics in them, by loading them with zero length vectors, i.e. vectors with an x and y offset of zero. A vector of zero length produces no screen image and does not alter the current screen location. To erase more than one LOC, the program can load a series of such vectors using a FOR loop. The FCLR function provides a simpler and faster way to erase graphics.

FCLR(L1[,L2])

FCLR called with one argument erases the display file starting at the LOC specified by the argument and continues to the end of the display file. FCLR called with two arguments erases the file starting at the first LOC and continuing up to, but not including, the second LOC. If the second LOC is beyond the display file, FCLR will erase to the end of the display file. If the second LOC is one greater than the first LOC, FCLR will erase a single LOC. FCLR loads all LOCs with display nulls. When the display processor encounters a display null, it simply goes on to the next LOC, having left the screen image, the current screen location, and the current graphic modes unchanged.

Example:

```
1.80 A "TYPE <CR> TO EXIT",A;C: WAIT FOR CARRAGE RETURN
1.90 X FCLR(0,LOC+2);C: CLEAR ALL GRAPHICS
1.99 Q
```

These lines added to the example wait for the operator to type a carraige return when the rotation has completed, then clear all the LOCs that comprise the image.

9.2.8 Returning Coordinates

The FOCAL program that has loaded the display file with lines and points may at some later time need to know the coordinates of the line or point in a given LOC. This data might be useful, for instance, when blowing up an image by increasing the length of each line that comprises it by some percentage. The functions FXCO and FYCO return respectively the x and y coordinate of a specified LOC.

9.2.8.1 FXCO(LOC)

FXCO returns as its functional value the x coordinate of the line or point in the LOC specified as its argument. If the LOC contains characters, a display jump, or a display null, FXCO returns the value -4095.

9.2.8.2 FYCO(LOC)

FYCO acts exactly like FXCO to return the y coordinate in the specified LOC.

Example:

```
1.79 D 4;GO ALTER THE SIZE OF THE CIRCLE

4.10 A "BY WHAT VALUE TO MULTIPLY THE RADIUS",P
4.20 F I=1,LT;D 5

5.10 S DX=FXCO(I);S DY=FYCO(I)
5.20 I (DX) 5.5
5.30 S DX=(DX-500)*P;S DY=(DY-500)*P
5.35 X FDIS(FDIS(I))
5.40 X FPT(I,DX+500,DY+500)
5.50 R
```

These lines, when added to the main example, allow the operator to change the size of the circle. Section 4 asks for the value by which to multiply the size of the radius. It then calls section 5 for each point on the circle. Section 5 obtains the coordinates of the next LOC, changes these to an offset from the center of the circle, multiplies this offset by the value input, then reloads the LOC with a new pair of coordinates. Before reloading each LOC it obtains the modes of the LOC by using FDIS with one positive argument, then sets the mode of the next load by using the negative value returned as the argument to FDIS. Thus, the modes associated with each LOC remain the same as the coordinates in the LOC change.

9.2.9 Handling the Light Pen

The VT11 light pen consists of a photocell in a pencil-shaped container. The photocell, when activated by being placed near screen graphics, causes a pulse to be sent to the display processor. When the pulse arrives, the processor decides whether the graphics which caused the pulse has been designated as light pen sensitive. If so, the processor interrupts its activity and makes a note of the x and y coordinates of the screen location where the hit occurred, and of the LOC in the display file that contains the graphic element hit.

The VT11 graphics package allows the FOCAL program to make use of the light pen in three ways. The FOCAL program can determine the coordinates of the last hit as well as the number of the LOC containing the graphic datum which caused the hit. It can also cause the coordinates of the hit to be automatically used for light pen tracking.

Whenever a light pen hit occurs on light pen sensitive graphics, the coordinates of the hit are automatically entered into the graphic datum in LOC 0, provided LOC 0 contains a vector or a point. This means that the vector or point in LOC 0 automatically alters so that its coordinates are those of the latest light pen hit. Thus, a vector in LOC 0 will extend, after a light pen hit, from the lower left corner of the screen to the location of the hit. A point in LOC 0 will automatically move to the location of the hit. Moreover, the FOCAL program can always read the last location of a light pen hit by using FXCO and FYCO to read the coordinates of the graphic element in LOC 0. The program can make use of these coordinates in another way, for light pen tracking.

Light pen tracking consists of drawing a polygon or other 'tracking element' which will follow the light pen as the operator moves the pen around the screen. The program could continually read the coordinates of the center of this tracking element to determine at any moment where the light pen is located. The FOCAL program could accomplish tracking by drawing the tracking element, then reading the coordinates of each hit in a FOR loop and using these coordinates to alter the center of the tracking element. This takes a great deal of time. The longer it takes to move the tracking element the slower the speed with which the tracking element can follow the light pen.

The VT11 graphics package makes available a much faster method of tracking the light pen. Since each light pen hit automatically alters the coordinates in LOC 0, making LOC 0 the center of the tracking element will cause the element to track the light pen automatically, as soon as each hit occurs, and without the intervention or supervision of the FOCAL program. To accomplish this, the program draws the tracking element in the LOCs immediately following LOC 0. LOC 1 usually contains an invisible vector to offset the image of the tracking element from its center. The next LOCs contain light pen sensitive vectors that comprise the tracking element. The LOC following the last LOC that comprises the tracking element contains an invisible absolute point, (usually at the lower left hand corner of the screen) so that graphics in subsequent locs will not move when the tracking element moves.

Now each time a hit occurs on the image formed by the light pen sensitive vectors following LOC 0, the coordinates of the visible or invisible vector (or point) in LOC 0 automatically alter to those of the light pen hit. Since the screen location produced by LOC 0 determines the center of the tracking element, the tracking element moves so that its center is the location of the light pen hit. If, for instance, the tracking element is drawn in the shape of a square, a hit on one of its sides will cause the square to move so that its center becomes the location of the hit.

The next example draws a tracking element consisting of a square and its diagonals, each diagonal drawn in two directions (When a light pen hit occurs, the hardware requires a certain delay before informing the display processor that the hit has occurred. This means the x and y coordinates of the hit as saved by the display processor are actually somewhat further along in the image than the actual hit coordinates. If the light pen sensitive diagonals in the tracking element were drawn in only one direction, it would impose a bias in that direction on each hit. Drawing the diagonals in two directions insures that on the average, the bias caused by the hardware delay cancels out.). Each time the user hits a carriage return on the keyboard, the program draws a line from the previous location of the tracking element to its current location thus enabling the user to draw a track of straight lines with the light pen.

Example:

```
1.10 X FVT(1,100);C: SET UP DISPLAY FILE,DISPLAY ON
1.20 X FMOV(0,500,500);D 2;C: SET UP LOC 0 AND DRAW TRACKER
1.25 X FDIS(0,5,0,0);C: SET FOR NON-LIGHT PEN SENSITIVE
1.30 S LOC= FSET(LOC,500,500);C: ABS POINT, CENTER SCREEN
1.40 S X1=500;S Y1=500;C: SET INITAL X AND Y COORDINATES
1.50 A "DRAW",A;C: WAIT FOR USER TO TYPE
1.55 I (A) 1.6,1.6;Q;C: WAIT FOR USER TO TYPE
1.60 S X=FXCO(0);S Y=FYCO(0);C: ELSE GET COORDS OF HIT
1.70 S LOC=FVEC(LOC,X-X1,Y-Y1)
1.71 C:DRAW VECTOR TO THERE FROM PREVIOUS HIT
```

```
1.80 S X1=X;S Y1=Y;C: RESET CURRENT COORDINATES
1.90 G 1.5;C: GO BACK FOR NEXT HIT

2.05 X FDIS(0,5,0,1);C: SET LIGHT PEN SENSITIVE
2.07 C: THEN DRAW THE TRACKER,A CROSSED SQUARE
2.10 X FVEC(1,30,30);X FVEC(2,-60,-60)
2.20 X FVEC(3,30,30)
2.30 X FVEC(4,30,-30);X FVEC(5,-60,60)
2.40 X FVEC(6,30,-30)
2.50 X FMOV(7,30,30)
2.60 X FVEC(8,0,-60);X FVEC(9,-60,0)
2.70 X FVEC(10,0,60);X FVEC(11,60,0)
2.80 S LOC=12;C: SET TO NEXT LOC AFTER TRACKER
```

FLP(N)

Function FLP returns the number of the LOC last hit by the light pen. Before any hits have occurred this number will be 0. As soon as a light pen hit occurs on light pen sensitive graphics, the current value will change to the number of the LOC hit. When the FOCAL program calls FLP with a positive argument, it returns the number of the LOC last hit by the light pen. If the program calls FLP with an argument of 0, FLP will wait for a light pen hit to occur before returning. It will then return the number of the LOC hit.

Example:

```
6.05 T "HIT THE CIRCLE WITH THE LIGHT PEN",!
6.10 S L=FLP(0);C: WAIT FOR HIT;SAVE ITS LOC
6.20 S DX=FXCO(L)-500;S DY=FYCO(L)-500
6.21 C:GET OFFSET OF POINT HIT FROM CENTER
6.25 X FDIS(2,5,0,0);C: SET FOR SHORTDASH
6.30 X FVEC(LOC+1,DX,DY);C: RE-DRAW THE VECTOR TO POINT TO THE
6.31 C: POINT HIT
6.40 S R=(L-1)*.1;C: CALULATE ANGLE OF HIT FROM LOC
6.41 C: THAT CAUSED THE HIT
6.50 T R," RADIANS",!!;C: TYPE IF FOR THE OPERATOR
```

These lines added to the main example wait for the operator to hit the circle with the light pen. When the hit occurs the coordinates of the hit are used to redraw the vector to extend to the point hit. The LOC of the hit as returned by FLP is used to calculate the angle of the hit. This value is typed out. Note that when the hit occurs, LOC 0 is not loaded with the hit's coordinates because LOC 0 has not been loaded with a vector or a point. If LOC 0 had been loaded with a vector or a point, the vector or the point would take on the coordinates of the hit. But this would not effect the location of the circle since the circle consists of absolute points which do not move when other graphics moves.

9.2.10 Altering Display of Terminal I/O

The FOCAL program may need to alter the terminal I/O displayed on the screen in order to make room for graphics. FSCR allows the FOCAL program to alter the number of lines, the intensity, and the starting y coordinate of the terminal I/O.

FSCR(L,I,Y)

By executing FSCR the FOCAL program alters the terminal I/O display. The first argument determines the number of lines displayed before scrolling occurs; to a maximum of 31 lines. The second argument determines the intensity of the terminal I/O display from 0 (dimmest) to 7. Setting the intensity to 0 effectively eliminates I/O from the screen when the brightness knob is set less than half-way to full intensity. The third argument determines the y-coordinate at which the first line of I/O displays. Setting y to approximately 740 sets I/O to the top of the screen. FSCR will have no effect on terminal I/O display until the next attempt to scroll takes place.

Example:

```
        1.05 X FSCR(1,4,740);C:SET SCROLLING TO 1 LINE ONLY
        1.07 A "TYPE <CR> TO START",A,!!;C: WAIT FOR USER
        1.95 X FSCR(30,4,740);C: SET SCROLLING TO ITS DEFAULT CONDITION
```

These lines added to the example prevent the scrolling from overlapping the image, then restore the proper scrolling when the program ends.

CHAPTER 10

FUNCTIONS FOR USING THE VT55

10.1 INTRODUCTION

The VT55 video-graphics terminal is a version of the VT50 video terminal that includes graphics capability. As an alphanumeric terminal it resembles in every way the VT50 except that the VT55 has 24 lines of text with 80 characters per line (See VT50 VIDEO TERMINAL PROGRAMMER'S MANUAL, DEC-00-0VT5A-A-D for a complete description of the VT50.) The VT55's graphics capabilities consist of the ability to display two graphs across the viewing area, each graph consisting of up to 512 points. The graphs may be programmed to display as either a series of points or as histograms, that is, with the vertical area under each point shaded. The program may selectively display or discontinue the display of any point in either graph and may emphasize the display of any point by displaying a short vertical line, called a graphic cursor, running through it. The program may also cause all points in either graph to display or disappear from the screen. In addition to the two graphs the VT55 can display a grid of up to 512 vertical and 236 horizontal lines. The program may turn the display of the grid on or off independently of the two graphs.

Transmission of graphic data to the VT55 from the processor is in the form of the normal printing ASCII characers. A program selectable mode setting on the VT55 determines whether the VT55 will interpret incoming characters as ASCII terminal output for alphanumeric display or as coded graphics commands. The graphics commands, each encoded in the lower six bits of the ASCII characters, have a simple format (described the the VT55 PROGRAMMER'S GUIDE, DEC-00-0VT5A-A-D). The user of FOCAL need never, however, encode the graphic data. The set of functions provided automatically convert the function calls to the required strings of ASCII characters.

The functions offer the user the ability to make use of all the VT55's graphics capabilities. They consist of:

FGRA which sets the mode of the VT55, alphanumeric or graphics,

FMD0 and FMD1 which allow the FOCAL program to clear the entire display, turn each graph on and off as either histograms or points, and turn the display of the vertical markers and the grid on and off.

FGRD which allow the user to cause display of an entire grid in a single function call.

FXY which enables plotting and erasing of individual points as well as the plotting of lines specified by their endpoints.

FMRK, which allows the program to selectively display each vertical marker.

Two additional functions increase the ease of access to the alphanumeric capabilities of the VT55:

FCUR allows the program to move the text cursor and to create vertical labels.

FALP allows the program to execute all available alphanumeric functions such as homing the cursor and erasing characters.

10.2 THE FUNCTIONS

The following sections describe the functions in detail. Each section begins with the function it describes in its most complete format. Some functions can have a variable number of arguments, the number determining the operation the function performs. The sections describing these functions are divided according to operation. Unless otherwise indicated all arguments to these functions are integers in the range -32568 to +32567. Coordinate arguments apply to a screen scaled from (0,0) at the lower left corner to (511,235) at the upper right. The values returned by the functions have no significance. Therefore, these functions may be invoked using the Xecute statement, and use of the Set statement will not prove useful.

10.2.1 Turning the Graphic Mode On and Off

FGRA(N)

When the VT55 receives a string of ASCII characters it interprets them as either alphanumeric terminal output or as encoded graphics commands depending on a program selectable mode setting. Execution of each of the graphics functions causes output of a string of ASCII characters. If the VT55 is in alphanumeric mode when it receives these characters, the characters will display on the screen rather than causing graphics operations to occur. The FOCAL program uses FGRA to set the VT55 to graphics mode. Upon its execution, the FOCAL program can perform calls to the graphics functions. Before doing any character output using the TYPE or WRITE commands, for instance- the FOCAL program should call FGRA with an argument of 0 to put the terminal back in alphanumeric mode. Otherwise, the VT55 will continue to interpret the characters in graphic mode rather than as text.

The user may wonder why each graphic function executed does not turn graphic mode on, send its commands, then turn alphanumeric mode off. Graphics mode is turned on and off by sending a pair of ASCII characters to the VT55. Sending all four of these characters for every graphic command executed would greatly increase the total number of characters sent, thus decreasing the maximum speed of operation.

Example:

```
X FGRA(1);C: TURN GRAPHIC MODE ON

X FGRA(0);C: TURN GRAPHIC MODE OFF
```

10.2.2 Setting Display Modes

The two functions FMD0 and FMD1 each require three arguments. Each allows the FOCAL program to set three characteristics of the display.

FMD0(D,P,H)

The first argument to FMD0 determines whether to turn the entire graphic display on or off. An argument greater than 0 means on, equal to 0 means off. The second argument determines whether to display either or both graphs as points. A value of 0 means display neither graph, a value of 1 means display graph 1 only, 2 means display graph 2 only, and 3 means display both graphs, where FOCAL graphs 1 and 2 correspond to the hardware manual graphs labelled 0 and 1. The third argument determines whether or not to display the graphs as histograms. The possible values of this argument, 0, 1, 2, 3, have the same meaning as they do for the second argument. A graph displayed as both a series of points and as a histogram, will display as a histogram with its uppermost point brightened. Any of these three arguments may take a negative value. An argument with a negative value causes no change in the current status of the corresponding mode. Thus, a first argument of -1 would leave the display either on or off, depending on its current state. The initial state of the display before any call to FMD0 is assumed to be all modes off (argument values of 0).

ARGUMENT	MODE SET	VALUES
1	DISPLAY ON/OFF	0=DISPLAY OFF 1=DISPLAY ON
2	POINT DISPLAY	0=DISPLAY NEITHER GRAPH AS POINTS 1=DISPLAY GRAPH 1 AS POINTS 2=DISPLAY GRAPH 2 AS POINTS 3=DISPLAY BOTH GRAPHS AS POINTS
3	HISTOGRAM DISPLAY	0=DISPLAY NEITHER GRAPH AS A HISTOGRAM 1=DISPLAY GRAPH 1 AS A HISTOGRAM 2=DISPLAY GRAPH 2 AS A HISTOGRAM 3=DISPLAY BOTH GRAPHS AS HISTOGRAMS

Example:

```
X FMD0(1,3,0);C: DISPLAY ON, BOTH GRAPHS AS POINTS

X FMD0(0,-1,-1);C: DISPLAY OFF
```

FMD1(I,L,M)

The first argument to FMD1 specifies whether to erase both graphs and all horizontal and vertical lines. Erasing the graphs consists of setting the value of each of the 512 points in both graphs to zero. An argument value greater than zero erases both graphs, while a value equal to zero leaves them unchanged.

The second argument determines whether or not to display the horizontal and vertical lines that the program has plotted. A value of 0 means display neither horizontal nor vertical lines, a value of 1 means display horizontal lines only, 2 means display vertical lines only, and 3 means display both horizontal and vertical lines. The value of the third argument determines whether or not to display the vertical markers the program may have plotted. The possible values (0,1,2,3) have the same meaning as those for the second and third arguments to FMD0.

Like the arguments to FMD0, the arguments to FMD1, if less than 0, cause no change to the corresponding mode setting. Values of 0 for each of the arguments are assumed to be their original settings.

ARGUMENT	MODE SET	VALUES
1	INITIALIZE	0=DON'T INITIALIZE 1=INITIALIZE BY ERASING
2	GRID LINES	0=DISPLAY NEITHER VERTICAL NOR HORIZONTAL LINES 1=DISPLAY HORIZONTAL LINES 2=DISPLAY VERTICAL LINES 3=DISPLAY BOTH HORIZONTAL AND VERTICAL LINES
3	MARKER	0=DISPLAY MARKERS ON NEITHER GRAPH 1=DISPLAY MARKERS ON GRAPH 1 2=DISPLAY MARKERS ON GRAPH 2 3=DISPLAY MARKERS ON BOTH GRAPHS

Example:

```
    X FMD1(1,1,2);C: CLEAR, ENABLE VERTICAL LINES AND GRAPH 2 MARKERS

    X FMD1(0,-1,0)C: TURN OFF MARKERS ON BOTH GRAPHS
```

10.2.3 Displaying Vertical and Horizontal Lines

The VT55 provides 512 vertical and 236 horizontal lines which the FOCAL program may selectively display using FGRD. FGRD may have from one to six arguments. The operation it performs depends on the number of arguments. The first three possible arguments refer to vertical lines; the last three to horizontal lines.

FGRD(V,SV,DV,H,SH,DH)

Called with six arguments FGRD causes display of a complete grid on the VT55 screen. The magnitude of the first argument determines the number of vertical lines in the grid, its sign determines whether to display or erase the lines specified by the function. A positive argument means display, a negative one means erase. The second argument specifies the coordinate of the first such line; and the third determines the spacing between lines. Thus, if the first three arguments are 10, 20, and 50 then ten vertical lines will display, with the first at x coordinate 20, the next at 70, the next at 120 and so on. The last three arguments similarly specify the horizontal line placement with reference to the y axis.

Example:

```
        X FGRD(10,100,50,5,100,25)
```

This example causes display of a grid consisting of 10 vertical and 5 horizontal lines. Both sets of lines start at coordinate 100 to leave room for labels and text. The horizontal grid is twice as dense as the vertical grid.

FGRD(V[,SV[,DV]]), V<>0

FGRD called with one to three arguments, the first not equal to zero, specifies vertical lines only. If called with one argument, the argument's value determines the number of vertical lines to display or erase. These will display (or erase) evenly spaced lines starting at x=0 and ending as close to x=512 as even spacing permits. If two arguments are specified, the number of lines specified by the first argument will plot (or erase) evenly spaced, the first line at the x coordinate specified by the second argument, the last line as close to x=512 as even spacing permits.

Called with three arguments, FGRD plots (or erases) the number of vertical lines specified by the first argument starting at the x coordinate specified by the second argument, with spacing specified by the third argument. In all cases, the sign of the first argument determines whether the lines are to be displayed (positive) or erased (negative).

Example:

```
        X FGRD(3);C:  PLOT VERTICAL LINES AT X=0,255, AND 511

        X FGRD(-3,100);C:  ERASE AT X=100, 306, AND 511

        X FGRD(3,100,50);C:  PLOT AT X=100,150, AND 200
```

FGRD(0,H[,DH[,DH]])

When called with a first argument of zero, FGRD interprets the next one to three arguments as specifying horizontal lines. These three arguments are interpretted exactly the same way as those for vertical lines but relative to a y axis extending to y=235. The sign of the second argument determines whether the lines are to be displayed (positive) or erased (negative).

Example:

```
        X FGRD(0,100);C:  PLOT 100 HORIZONTAL LINES

        X FGRD(0,-100,50);C:  ERASE 100 HORIZONTAL LINES

        X FGRD(0,100,50,5);C:  PLOT 100 HORIZONTAL LINES 5 APART
```

10.2.4 Drawing Points and Lines

The FXY function, like FGRD, may be called with a variable number of arguments, the number determining the operation it performs. It may be used to erase an entire graph, plot and erase points, and plot straight (non-vertical) lines.

Each graph, referred to as GRAPH 1 or GRAPH 2, can consist of up to 512 points, each point at a separate x coordinate. Neither graph can have more than one point at any given x coordinate. This means that each time a point is plotted on a graph at a given x coordinate, any point previously displayed on that graph at that x coordinate disappears. There is always a point plotted at each x coordinate. But points plotted with their y coordinate greater than 235 will plot off screen and will not display.

FXY(G,SX,SY,EX,EY[,DX])

Called with six or seven arguments, FXY plots a straight line on the graph whose number (1 or 2) the first argument specifies. The second and third argument specify the x and y coordinates respectively of the line's starting point. The fourth and fifth arguments determine the x and y coordinates, respectively, of the line's endpoint. If the call does not include a seventh argument, the line drawn includes all the x values between the starting and endpoints. For a line plotted between (50,100) and (60,200), the endpoint of the line should be to the right of its starting point. Otherwise, only the point indicated by the second and third arguments will plot. The optional seventh argument specifies the x distance between each point on the line. A seventh argument of two, for instance, only plots every second possible point. The line from (50,100) to (60,200) would then consist of the points at x coordinates 50,52,54,56,58, and 60. Plotting a line using every point will, due to the graphic coding method, cause a faster plot than a line plotted using every second point, and a plot as fast as one using every third point.

Example:

```
        X FXY(1,100,150,200,250,4)
```

This example draws a line at a 45 degree angle with every 4th point displayed, using graph 2.

FXY(G,SX,SY,EX)

FXY called with four arguments causes display of a horizontal line on the graph specified by the first argument. The second and third arguments determine the x and y coordinates respectively of the line's starting point. The fourth argument specifies the x coordinate to which the line extends.

Example:

```
        X FXY(2,100,100,400)
```

This example draws a horizontal line 300 long on GRAPH 2 starting at X=100, Y=100.

FXY(G,SX,SY)

FXY called with three arguments plots a point on the graph specified by the first argument, at the coordinates indicated by the second and third arguments.

Example:

```
        X FXY(1,100,130)C:  DRAW ON POINT ON GRAPH 1
```

FXY(G,XS)

FXY called with two arguments erases the point on the graph specified by the first argument at the x coordinate specified by the second argument. Since only one point per graph can be plotted at any given x coordinate, the x coordinate alone is sufficient to specify the point to erase. Erasure consists of setting the y coordinate of the point to 236, that is, off the screen's viewing area. This method of erasure will not erase a point displayed as a histogram since in histogram mode the entire area beneath the point displays. To erase such a point the program should plot it at y coordinate 0 using FXY with three arguments.

Example:

```
        X FXY(1,100);C:  ERASE THE POINT ON GRAPH 1
        C:  AT X COORDINATE 100.
```

FXY(G)

FXY called with one argument completely erases the graph specified by the argument by setting all its y coordinates to 236. To erase a graph displayed as a histogram, the program should use FXY with four arguments to plot a horizontal line starting at x=0 and y=0 and ending at x=512.

Example:

```
        X FXY(2);C:  ERASE GRAPH 2
```

10.2.5 Displaying Markers

Each of the 512 x coordinates on both graphs has an associated marker. The marker is a vertical line, 16 points high, which displays with its base at the x coordinate that is the nearest multiple of 16 to the point currently on display at that y coordinate. FMRK allows the FOCAL program to display any or all of the markers.

FMRK(G,X[,N])

The first argument to FMRK specifies the graph to which the next arguments apply. The second argument specifies an x coordinate. If there is no third argument, the marker at the specified x coordinate will display. The marker will also display for a third argument greater than 0. It will cease to display for a third argument equal to 0.

Examples:

```
        X FMRK(1,100);C:  DISPLAY THE MARKER AT X=100 ON GRAPH 1

        X FMRK(1,100,1);C:  DISPLAY THE MARKER AT X=100 ON GRAPH 1

        X FMRK(1,100,0);C:  ERASE THE MARKER AT X=100 ON GRAPH 1
```

10.2.6 Alphanumeric Cursor Control

The VT55's cursor may be moved to any of the 80 character positions on any of the 24 lines. Any TYPE command will always begin output at the current cursor position. All but the bottom line of characters displayed on the VT55 screen, displays upon the area of the screen used for graphics. All but the leftmost character on each line also displays on this area. Therefore, the graphic area of the screen has room for 1817 (23x79) characters. The x and y coordinates of the lower left corner of any characters may be located using the formulas below. In the formulas, the lowest line on the screen is line 0, the next line up, line 1, and so on. The leftmost character on the screen is character 0, the next character 1, and so on. The symbol L stands for line number; the symbol C for the character position.

```
X=(L-1)*12

Y=(C-1)*7
```

Thus, the lower left hand corner of the fourth character (character 3) in the third line from the bottom (line 2) displays at (14,24).

FCUR allows the FOCAL program to position the cursor to any character position. It also provides for vertical printing useful in labelling.

```
FCUR(C,L,A,B,...)
```

The first argument to FCUR specifies the number of character positions to move the cursor from its current position. A negative value means move left, a positive value means move right. The cursor will not move beyond the left or right edge of the screen regardless of the argument value. The second argument specifies the number of lines to move; a value greater than 0 means move down, a value less than 0 means move up. The cursor cannot go beyond the top or bottom of the screen regardless of the argument value. When the cursor attempts to move below the last line on the screen, scrolling occurs.

Any number of arguments may follow the first two arguments. Each of these arguments specifies a character. Values 0 through 32 correspond to ASCII characters of octal codes 100 to 177. Values 33 through 128 correspond to ASCII characters of octal codes 40 through 177. The characters thus specified will be printed vertically downward from the cursor position created by the first two arguments. If these characters would extend beyond the bottom of the screen, scrolling will occur to make room for them.

Example:

```
        X FCUR(1,-2,0L,0A,0B,5,0L)
```

This example plots 'LABEL' vertically after moving the cursor 1 character position to the right and 2 lines down.

```
FCUR(C,L)
```

FCUR called with two arguments moves the cursor but prints no characters.

Example:

```
        X FCUR(-5,10);C:  MOVE 5 LEFT AND 10 UP
```

10.2.7 Generating Control Commands

The VT55 like the VT50 responds to a number of commands in the form of an escape character (octal 33) followed by some printing character. The FOCAL program could generate such commands using FCHR to generate the two characters. FALP simplifies command generation by automatically generating the escape character.

FALP(N)

FALP sends an escape character followed by the value specified as its argument. The table below lists the operations peformed by each possible argument.

ARGUMENT		OPERATION
OCTAL	DECIMAL	
110	72	MOVE THE CURSOR TO THE HOME POSITION
112	74	ERASE ALL LINES FROM CURSOR TO BOTTOM OF SCREEN
113	75	ERASE FROM CURSOR TO END OF LINE
133	91	ENABLE HOLD MODE
134	92	DISABLE HOLD MODE
135	93	PRINT ENTIRE SCREEN INCLUDING GRAPHICS

Example:

```
        X FALP(@110);C: HOME THE CURSOR
```

The FOCAL program should perform the following commands using the FCHR function rather than FALP since they do not require the preceeding escape character.

ARGUMENT		OPERATION
OCTAL	DECIMAL	
7	7	RING THE BELL
10	8	BACKSPACE THE CURSOR
11	9	PERFORM TAB
12	10	MOVE CURSOR DOWN ONE LINE
15	13	PERFORM CARRIAGE RETURN

Example:

```
        X FCHR(7);C: RING THE VT55'S BELL
```

APPENDIX A

FOCAL-11 OPERATIONS AND THEIR SYMBOLS

Control Characters	Use
%	Output format delimiter
!	Carriage return and line feed (spacing)
↵	
#	Carriage return without line feed
$	Type variable symbol table
( )	Enclosures for mathematical expressions
[ ]	
< >	
" "	Text string
? ?	Trace feature
@	The following numbers are to be interpreted as octal.
' '	The enclosed expression should be evaluated and the resulting positive integers used as part of the file name in a LIBRARY command.

Terminators	Use
SPACE key	Names or numerical values
RETURN key	Lines
ALT MODE key	ASK statement
Comma	Expression
Semicolon	Multiple commands and statements
Line Feed	ASK statement

APPENDIX B

FOCAL-11 ERROR DIAGNOSTICS

Code	Explanation
?00	Manual restart from location 0 or by CTRL/C.(r)
?01	Illegal line number.
?02	Illegal variable or function name.
?03	Unmatching parentheses.
?04	Illegal command.
?05	Nonexistent line number.
?06	Nonexistent group or line number in DO.
?07	Illegal format in SET or FOR.
?08	Double or missing operators in expression.
?09	Stack overflow, nonexistent device, or bad address specification.
?10	Core filled by text or command line too long.(o)
?11	Core filled by variables or no room for variables.(o)
?12	Exponent range greater than E+38.(o)
?13	Disallowed bus address in "FX".(o)
?14	Division by zero attempted.(r)
?15	Attempt to exponentiate to a negative power or power too large. (r)
?16	Too many characters in input data.(r)
?17	Square root of negative number.(r)
?18	Input buffer overflow

?19	Subscript out of range (r)
?20	Invalid argument to function call (o)
?21	Bad argument to function call (o)
?22	Unable to perform specified interrupt linkage or unable to schedule desired routine. (o) (r)
?23	Symbol table shuffle error (i)(R)
?24	Memory allocation error (R)
?25	Intenal memory error (i)(R)
?26	Illegal RELM request (o)(R)
?27	General I/O error (r)(R)
?28	Insufficient resources (r)(R)
?29	File number out of range (o)(R)
?30	Illegal format code (o)(R)
?31	File specification syntax error (o)(R)
?32	Fatal write error encountered (unopened channel,etc) (o)(R)
?33	Attempt to read (or write) past EOF.(o)(R)
?34	File was not found (r)(R)
?35	Illegal library command encountered while performing a LIBRARY GET, RUN,or NEXT. (o)(R)
?36	Internal Virtual file error (i)(R)
?37	Illegal floating point call (i)
?38	Integer overflow error (range outside +32,767 to -32,768) (r)
?39	Interrupt linkage error (i)

(i) Internal FOCAL error
(o) Operational error
(r) A run-time error
(R) RT-11 version only

APPENDIX C

FOCAL-11 COMMAND AND FUNCTION SUMMARY

C.1 COMMANDS

Command	Abbreviation	Example of Form	Action
ASK	A	ASK M	Request input from the currrent input device.
		ASK "AGE",A	Output text (AGE) and store input as variable A.
COMMENT	C	COMMENT	Ignore the remainder of the line
DO	D	DO 4.1	Execute line 4.1; return to command following DO command.
		DO 4	Execute all group 4 lines; upon completion, return to command following DO command or when a RETURN is encountered.
	D A	DO ALL	Execute entire program as a subroutine.
	D v	DO var	Execute the line or group of lines defined by the variable (var).
ERASE	E	ERASE	Erase the symbol table.
		ERASE 2	Erase all group 2 lines.
		ERASE 2.1	Erase line 2.1.

	E A	ERASE ALL	Erase the entire program and clear all variables.
	E T	ERASE TEXT	Erase text only; do not erase symbol table.

FOR	F	FOR I=X,Y,Z;(commands)
		FOR I=X,Z; (commands)
			Where the command(s) is executed at each new value of I.
			
			X = initial value of I.
			Y = value added to I until I is incremented beyond Z. Y assumed=1 if omitted.

GO	G	GO	Starts program at lowest numbered line number.
		GO 3.4	Transfers control to line 3.4
		GO 3	Transfers control to lowest numbered statement in group 3.

IF	I	IF(X)L1,L2,L3
		IF(X)L1,L2; (commands)
		
		IF(X)L1; (commands)
			Where X is a defined variable, a value, or an expression followed by one to three line numbers. (FOCAL-11 also supprts group numbers) If X is less than zero, transfer control to the line number L1, if X is equal to zero, transfer control to the second line number, L2. If X is greater than zero, transfer control to L3. If the line number is not specified, proceed to the next sequential command.

KILL	K	KILL	Stop all I/O and reset I/O devices. Error code ?09 is printed.
LIBRARY	L	LIBRARY INPUT 1,TEST/T	Attempts to open the old file SY:TEST.FCL as ASCII file number 1. If the file does not exist FOCAL returns an error message.

LIBRARY OPEN 1,TEST/T
: Attempts to open the old file SY:TEST.FCL as ASCII file number 1. If the file does not yet exist it is created.

LIBRARY MAKE 1,TEST/T
: Creates a new file called SY:TEST.FCL as ASCII file number 1.

LIBRARY CLOSE 1
: Terminate all activity with file number 1.

LIBRARY TYPE 1,<type args>
: Type output to file 1.

LIBRARY WRITE 1,<write arg>
: Write output to file 1.

LIBRARY ASK 1,<ask args>
: Read ASK input from file 1.

LIBRARY RUN TEST<args>
: ERASE all text and variables, read in the FOCAL program saved in the file SY:TEST.FCL, and begin execution at the line or group specified. (If args are left out the first statement in the program is assumed.)

LIBRARY GET TEST
: Read in the file SY:TEST.FCL and merge it with the current program. This form of the command must be terminated by a carriage return.

LIBRARY NEXT TEST<arg>
: ERASE all text, leaving variables intact, and "RUN" the program saved in the file "TEST.FCL". Execution will continue at the start of the program or at the line/group number specified in arg.

LIBRARY SAVE TEST
: The entire program text is saved in the newly created file called SY:TEST.FCL. If another

			file by that name already exists, it is deleted.
		LIBRARY DELETE TEST	The RT-11 file SY:TEST.FCL is deleted.
MODIFY	M	MODIFY 1.15	Enable editing of line 1.15.
OPERATE			Selects the input and/ or output device for such commands as TYPE and ASK.
	O	OPERATE	Forces all pending output to the currently selected output device. The current output device is not altered. (RT-11 maintains a rather large output buffer to increase I/O efficiency.)
	O T	OPERATE T	Select terminal printer.
	O K	OPERATE K	Select terminal keyboard for input.
	O P	OPERATE P	Select high-speed paper tape punch for output.
	O R	OPERATE R	Select high-speed paper tape reader for input.
	O RP	OPERATE RP	Select both high-speed reader and punch for I/O.
	O TK	OPERATE TK	Select both terminal keyboard and printer for I/O.
	O L	OPERATE L	Select line printer for output.
QUIT	Q	QUIT	Return control to the user (command mode).
RETURN	R	RETURN	terminate DO subroutines, returning to the original sequence.
SET	S	SET A=5/B*C	Perform arithmetic assignment. The variable on the left side of the "=" is set equal to the value of the expression on the right.
TYPE	T	TYPE A+B+C	Evaluate expression and type "=" followed by result in current output format.

		TYPE A-B,C/E	Compute each expression and type the resultant values.
		TYPE "TEXT STRING"	Type text, may be followed by ! to generate carriage return/line feed, or # to generate only a carriage return.
		TYPE $	Type the symbol table. Must be terminated by a carriage return only.
WRITE	W	WRITE	Type out the entire program.
	W A	WRITE ALL	Same as WRITE.
		WRITE 1	Type out all group 1 lines.
		WRITE 1.1	Type out line 1.1.
XECUTE	X	XECUTE FSBR(5,ARG)	Call functions without need for a dummy SET statement.
(TRACE)	GO?		Starts at lowest numbered line and traces entire program until another ? or an error is encountered , or until completion of program.

C.2 FUNCTIONS

Function	Form	Action
FABS	FABS (expression)	Returns absolute (positive) value of expression.
FADC	FADC(channel)	Provides access to A/D channels.
FCHR	FCHR(arg)	Accepts and/or prints ASCII codes.
FCLK	FCLK()	Returns the value of the time elapsed.
FCOS	FCOS(angle)	Calculates the cosine of a specified angle in radians.
FERR	FERR(line or group)	Intercept a FOCAL error and perform a DO to the specified line or group.
FEXP	FEXP(arg)	Exponential function.
FINT	FINT(vector,group,priority,CSR address,mask)	This routine logically connects a line or group of the FOCAL user's program with a device and it's interrupt vector.
FITR	FITR(expression)	Provides the integer part of a number.
FLN	FLN(arg)	Natural logarithm function
FLOG	FLOG(arg)	Base ten logarithm
FPRM	FPRM(parameter,value)	Alter FOCAL internal parameters.
FQUE	FQUE(count,group,interval,delay,priority)	This function schedules a line or group of the user's program to be performed a specified number of times at regular time intervals at a specified software priority.

FRAN	FRAN()
	FRAN(1)
		Generates a random value between -1 and 1.

FSBR	FSBR(group,arg)	Calls program group specified as a subroutine.
FSGN	FSGN(arg)	Sign function.

FSIN	FSIN(angle)	Calculates the sine of the specified angle in radians.
FSQT	FSQT(expression)	Computes square root of expression.
FX	FX(func, UNIBUS-address, data)	Controls additional device options or non-standard peripherals or references core storage.

APPENDIX D

EXTENDED FUNCTIONS

D.1 EXAMPLE OF A RECURSIVE FUNCTION

```
1.1 SET N=5
1.2 TYPE FSBR(5,N);C-FACTORIAL FUNCTION
1.3 QUIT

5.1 IF (1-&)5.2;R
5.2 SET &=&*FSBR(5,&-1)
```

D.2 SOME TRANSCENDENTAL FUNCTIONS SERIES

```
11.01 C TAN:  FSBR(11,ARG)
11.10 I (&↑2-.01)11.2;S &=&/2;D 11;S &=2*&/(1-&↑2+1E-20);R
11.20 S &=&+&↑3/3 +&↑5/7.5+&↑7/315

12.01 C ASIN:FSBR(12,ARG);  ACOS:FSBR(12.3,ARG)
12.10 I (&↑2-.01)12.2;S &=&/(FSQT(1+&)+FSQT(1-&));D 12;S &=2*&;R
12.20 S &=&+&↑3/6+.075*&↑5+&↑7/22.4;R
12.30 D 12;S &=1.570796-&;R

13.01 C ATAN;  FSBR(13,ARG)
13.10 I (&↑2-.01)13.2;S &=&/(1/FSQT(&↑2+1));D 13;S &=2*&;R
13.20 S &=&-&↑3/3+&↑5/5-&↑7/7

14.01 C EXP:  FSBR(14,ARG)
14.10 I (&↑2-.01)14.2;S &=&/2;D 14;S &=&↑2;R
14.20 S &=1+&+&↑2/2+&↑3/6+&↑4/24+&↑5/120+&↑6/720

15.01 C LOG;  FSBR(15,ARG)
15.10 I (&↑2-2.04*&+1)15.2;S&=FSQT(&);D 15;5 &=2*&;R
15.20 S &=(&-1)/(&+1);S &=2*(&+&↑3/3+&↑5/5+&↑7/7)

16.01 C SINH:FSBR(16,ARC) COSH:FSBR(16.3,ARG)
16.10 I (&↑2-.01)16.2;S &=&/3;D 16:S &=3*&+4*↑3;R
16.20 S &=&↑3/6+&↑5/120;R
16.30 D 16;S &=FSQT(1+&↑2)
*
*
```

Comment lines contain format of call for subroutines shown.

D.3 EXAMPLE OF A DEVICE CONTROL FUNCTION (AND BINARY PRINTOUT)

```
*
21.01 C THIS PROGRAM PRINTS THE BIT PATTERN IN SWITCHES 15-00
21.02 C AND WAITS FOR SWITCH 15 TO BE CHANGED.
21.03 C EXIT WITH CTRL/C TWICE
21.05 X FPRM(8,1)
21.07 SET Z1=1
21.10 SET Z=FX(0,@177570,-1)
21.20 IF(Z1*(Z+1))21.5,21.1,21.1
21.50 SET Z1=Z;TYPE !;X FSBR(40,Z);TYPE " ",↑B(Z);G 21.1

40.10 SET N=15
40.15 IF(-&)40.2,40.2;S &=&-@100000+@77777+@1
40.20 IF(FITR(&)-FITR(2↑N))40.3;X FCHR(@61)
40.21 SET &=FITR(&-FITR(2↑N))
40.22 SET N=N-1
40.25 IF (N)40.4,40.2,40.2
40.30 X FCHR(@60);G 40.22
40.40 RETURN
*
*
*GO

1111011001001000    1111011001001000
0011111111111111    0011111111111111
1000011111111111    1000011111111111
0000000000000001    0000000000000001
1000000011111111    1000000011111111
0000000001111110    0000000001111110
1000000000000000    1000000000000000
0111111111111111    0111111111111111
1111111111111111    1111111111111111
```

D.4 EXAMPLE OF A TIMING ROUTINE

```
C:FOCAL-11S V1  (RT-11) 16-OCT-74

  1.10 I (A) 2,1.2,2
  1.20 T !"ENTER THE CODE TO BE TIMED IN GROUP 3."!
  1.30 T "THE CODE WILL BE EXECUTED 1000 TIMES, AND THE TIME OF THE"!
  1.40 T "CODE WILL BE PRINTED IN MILLISECONDS. TYPE 'GO' TO"
  1.41 T " PROCEED."!
  1.45 S A=1
  1.50 Q

  2.10 S X=FCLK()
  2.20 FOR I=1,1000;D 3
  2.30 S X=FCLK(X)
  2.40 S Y=FCLK()
  2.50 FOR I=1,1000;D 4
  2.60 S Y=FCLK(Y)
  2.70 T "THE TIME IS ",(X-Y)/60," MS."!!!;S A=0
  2.80 E 3
  2.90 G

  4.10 C

*G
```

```
ENTER THE CODE TO BE TIMED IN GROUP 3.
THE CODE WILL BE EXECUTED 1000 TIMES, AND THE TIME OF THE
CODE WILL BE PRINTED IN MILLISECONDS. TYPE 'GO' TO PROCEED.
*3.1 S A=3,R
*G
THE TIME IS =     3.25000 MS.

ENTER THE CODE TO BE TIMED IN GROUP 3.
THE CODE WILL BE EXECUTED 1000 TIMES, AND THE TIME OF THE
CODE WILL BE PRINTED IN MILLISECONDS. TYPE 'GO' TO PROCEED.
*
```

APPENDIX E

Loading FOCAL-11

Paper-tape Versions

The procedure for loading FOCAL-11/PTS requires loading the Bootstrap Loader, followed by the FOCAL-11 binary paper tape (DEC-11-LFOCB-A-PB). The details for loading the Bootstrap and Absolute loaders are described in Chapter 5 of the PDP-11 Paper Tape Software Programming Handbook (DEC-11-XPTSA-A-D). The FOCAL program is self-starting and the message ?00 AT 0.00 is printed to indicate the program is started.

NOTE

FOCAL-11 only uses memory between the Absolute Loader and the bottom of core. To make full use of memory, make certain that the Absolute Loader is loaded into the highest location possible.

Restarting FOCAL-11

If the user wants to restart FOCAL-11 and give it new commands, there are two methods which may be used: typing CTRL/C, (possibly twice if doing I/O) or performing manual restart. Any time the user types CTRL/C (accomplished by holding down the CTRL key and typing C), FOCAL prints the message

```
?00 AT 0.00
*
```

The asterisk indicates that FOCAL-11 is in command mode.

The procedure for restarting FOCAL-11 manually is as follows:

1. Press HALT if the run light is on
2. Set the switch register to 0000000
3. Press LOAD ADDR
4. Press START

FOCAL-11 prints:

```
?00 AT 0.00    The error code indicates manual restart.
*              The asterisk indicates that FOCAL-11
               is in command mode.
```

The resart feature would be useful, for example, if the user should detect that his program is not operating properly. He would then use one of the above restart methods, modify his program, and re-execute the program.

RT-11 Versions:

The RT-11 versions of FOCAL-11 are loaded into memory via the RT-11 Run command.

```
          .R FOCALS        (single precision 12K version)
          ?00 AT 0.00
          *↑C

          .R FOCALD        (double precision 12K version)
          ?00 AT 0.00
          *↑C

          .R FOCAL8        (8K version)
          ?00 AT 0.00
          *↑C

          .                (CTRL/C is used to exit from FOCAL)
```

In order to restart RT-11 versions of FOCAL, it is necessary to strike CTRL/C (possibley twice) and then type 'RE' (for REenter). For example:

```
          .R FOCALS
          ?00 AT 0.00
          *↑C

          .RE
          ?00 AT 0.00
          *1.1 G 1.1
          *GO
          ↑C
          ↑C

          .RE
          ?00 AT 0.00
          *
```

When this is done, all LIBRARY files have been released. The user's program and variables will be intact.

APPENDIX F

ASCII CHARACTER SET

(Octal)

Printing Character	7-bit ASCII	6-bit Trimmed ASCII	Printing Character	7-bit ASCII	6-bit Trimmed ASCII
@	100	00	Form Feed	014	
A	101	01	Carriage Ret.	015	
B	102	02	ALT MODE(ESC)	175	
C	103	03	Rubout	177	
D	104	04	(Space)	040	40
E	105	05	!	041	41
F	106	06	"	042	42
G	107	07	#	043	43
H	110	10	$	044	44
I	111	11	%	045	45
J	112	12	&	046	46
K	113	13	'	047	47
L	114	14	(	050	50
M	115	15	)	051	51
N	116	16	*	052	52
O	117	17	+	053	53
P	120	20	,	054	54
Q	121	21	-	055	55
R	122	22	.	056	56
S	123	23	/	057	57
T	124	24	0	060	60
U	125	25	1	061	61
V	126	26	2	062	62
W	127	27	3	063	63
X	130	30	4	064	64
Y	131	31	5	065	65
Z	132	32	6	066	66
[	133	33	7	067	67
←	134	34	8	070	70
]	135	35	9	071	71
↑	136	36	:	072	72
←	137	37	;	073	73
Null	000		<	074	74
Horizontal Tab	011		=	075	75
Line Feed	012		>	076	76
Vertical Tab	013		?	077	77

APPENDIX G

PAPER TAPE SYMBOL TABLE

Single precision version:

```
RT-11 LINK    V03-01    LOAD MAP
PFOC  .LDA              12-JAN-75
SECTION ADDR    SIZE    ENTRY   ADDR    ENTRY   ADDR    ENTRY   ADDR
. ABS.  000000  001000  ALOG    000000  ALOG10  000000  DEXP    000000
                        DLOG    000000  DLOG10  000000  EXP     000000
                        $DBL    000000  SORTJ   104600  SORTC   104602
                        PRINTC  104604  READC   104606  OUTCH   104610
                        INCH    104612  GETC    104614  PACKC   104616
                        TESTC   104620  GETLN   104622  FINDLN  104624
                        PRNTLN  104626  COPYLN  104630  START   104632
                        SPNOR   104634  ERASEV  104636  ERASET  104640
                        PRINT2  104642  DIGTST  104644  PARTST  104646
                        GROOVY  104650  SKPLPR  104652  SKPNON  104654
                        TASK    104656  EVAL.X  104660  FPMP    104662
                        FREAD   104664  FPRINT  104666  ITOA    104670
                        OTOA    104672  BTOA    104674  PATCH1  104676
                        PATCH2  104700  LPSCSR  170400  LPSBUF  170402
                        LPS     177514  PRS     177550  PPS     177554
                        TKS     177560  TPS     177564
        001000  000240  BEGIN   001000  PATCH   001000  PATCHB  001024
                        FBASE   001050  IOLIST  001140  FNTABL  001142
                        IOGO    001146  IPRS    001146  ITKS    001150
                        IPPS    001152  ITPS    001154  ILPS    001156
                        IOPATC  001160  CONFIG  001172  CFRS    001174
        001240  010740  TERMS   001510  TLIST   001540  PCF     001570
                        FLAC    001610  FSW     001620  SWITCH  001621
                        LINENO  001622  FISW    001624  INDEV   001632
                        OUTDEV  001634  BOTTOM  001652  PARAM   001656
                        STARTX  002166  TESTX   002576  SKPNOX  002634
                        SORTB   002650  SORTD   002702  GETLNX  002716
                        FINDX   003030  PRIN2A  003070  XTSTLP  003100
                        DIGTSA  003116  GROVX   003134  CHIN    003152
                        OUT     003272  XPRNTL  003372  SPNORX  003432
                        PACKX   003454  GETX    003570  PROC2   004024
                        COPYLX  004512  ERTX    004522  DO2     004546
                        EVALUX  005472  WHIPV   005664  ERVX    006050
                        PARTSA  006170  FPRM    006210  TASKX   006534
                        XI33    007012  XOUT    007064  $PRINT  007236
                        $READ   010264  FADC    010562  XABS    010622
                        XSGN    010632  XFCLK   010640  CLKT    010732
                        ITOAX   011220  XTOA    011234  OTOAX   011330
                        BTOAX   011336  XRAN    011402  XCHR    011474
                        XFSBR   011544  XEX     011610
        012200  001346  $FPMPX  012200  FINT    013202  XITR    013376
                        FSQT    013404  FSIN    013412  FCOS    013420
                        FLN     013426  FLOG    013434  FEXP    013442
        013546  000510  ADF$IS  013546  ADF$PS  013554  SUF$PS  013560
                        SUF$MS  013564  ADF$MS  013576  SUF$IS  013606
                        SUF$SS  013612  $SBR    013612  ADF$SS  013616
                        $ADR    013616  ADD$    013632
        014256  000116  AINT    014256  $INTR   014274
        014374  000354  COS     014374  SIN     014430
        014750  000306  DIF$PS  014750  DIF$MS  014754  DIF$IS  014764
                        DIF$SS  014770  $DVR    014770
        015256  000312  MUF$PS  015256  MUF$MS  015262  MUF$IS  015272
                        MUF$SS  015276  $MLR    015276
        015570  000174  SQRT    015570
        015764  000100  CCI$    015764  CDI$    015764  $IC     015764
                        $ID     015764  CFI$    016000  $IR     016000
        016064  000116  CIC$    016064  CID$    016064  CLC$    016064
                        CLD$    016064  $DI     016064  CIF$    016074
                        CLF$    016074  $RI     016074  CIL$    016174
                        CLI$    016200
TRANSFER ADDRESS = 011752
HIGH LIMIT = 016202
```

Double precision version:

```
RT-11 LINK      V03-01  LOAD MAP
PFOC8 .LDA              12-JAN-75
SECTION ADDR    SIZE    ENTRY   ADDR    ENTRY   ADDR    ENTRY   ADDR
. ABS.  000000  001000  $DBL    000001  MAXTSK  000010  SORTJ   104600
                        SORTC   104602  PRINTC  104604  READC   104606
                        OUTCH   104610  INCH    104612  GETC    104614
                        PACKC   104616  TESTC   104620  GETLN   104622
                        FINDLN  104624  PRNTLN  104626  COPYLN  104630
                        START   104632  SPNOR   104634  ERASEV  104636
                        ERASET  104640  PRINT2  104642  DIGTST  104644
                        PARTST  104646  GROOVY  104650  SKPLPR  104652
                        SKPNON  104654  TASK    104656  EVAL.X  104660
                        FPMP    104662  FREAD   104664  FPRINT  104666
                        ITOA    104670  OTOA    104672  BTOA    104674
                        PATCH1  104676  PATCH2  104700  LPSCSR  170400
                        LPSBUF  170402  LPS     177514  PRS     177550
                        PPS     177554  TKS     177560  TPS     177564
        001000  000650  BEGIN   001000  PATCH   001000  PATCHB  001024
                        FBASE   001050  IOLIST  001170  FNTABL  001172
                        IOGO    001176  IPRS    001176  ITKS    001200
                        IPPS    001202  ITPS    001204  ILPS    001206
                        IOPATC  001210  I.SLOT  001222  E.SLOT  001422
                        $QUEUE  001422  CONFIG  001602  CFRS    001604
        001650  011234  TERMS   002120  TLIST   002150  PCF     002200
                        FLAC    002220  FSW     002230  SWITCH  002231
                        LINENO  002232  FISW    002234  PTR0    002236
                        INTSW   002240  INTCHN  002242  INTPRI  002262
                        INDEV   002270  OUTDEV  002272  BOTTOM  002310
                        PARAM   002326  STARTX  003000  TESTX   003420
                        SKPNOX  003456  SORTB   003472  SORTD   003524
                        GETLNX  003540  FINDX   003652  PRIN2A  003712
                        XTSTLP  003722  DIGTSA  003740  GROVX   003756
                        CHIN    003774  OUT     004114  XPRNTL  004214
                        SPNORX  004254  PACKX   004302  GETX    004416
                        PROC2   004656  COPYLX  005350  ERTX    005360
                        DO2     005404  EVALUX  006330  WHIPV   006522
                        ERVX    006706  PARTSA  007026  FPRM    007046
                        TASKX   007372  XI33    007650  XOUT    007722
                        $PRINT  010074  $READ   011122  FADC    011420
                        XABS    011460  XSGN    011470  XFCLK   011476
                        CLKT    011570  ITOAX   012056  XTOA    012072
                        OTOAX   012166  BTOAX   012174  XRAN    012240
                        XFERR   012330  XCHR    012400  XFSBR   012450
                        XEX     012514
        013104  002034  INTREX  013104  ZAPINT  013372  $INTRP  013470
                        $FINT   013732  XFQUE   014204  QUESET  014744
                        QUECHK  015046
        015140  001434  $FPMPX  015140  FINT    016160  XITR    016360
                        FSQT    016366  FSIN    016374  FCOS    016402
                        FLN     016410  FLOG    016416  FEXP    016424
                        DINT    016432
        016574  001164  SUD$PS  016574  SUD$MS  016600  ADD$PS  016620
                        ADD$MS  016624  ADD$IS  016644  SUD$IS  016656
                        SUD$SS  016666  $SBD    016666  ADD$SS  016672
                        $ADD    016672
        017760  000560  DCOS    017760  DSIN    020036
        020540  000634  DEXP    020540
        021374  000742  DID$PS  021374  DID$MS  021400  DID$IS  021420
                        DID$SS  021430  $DVD    021430
        022336  000574  DLOG10  022336  DLOG    022342
        023132  000204  DSQRT   023132
        023336  000660  MUD$PS  023336  MUD$MS  023342  MUD$IS  023362
                        MUD$SS  023372  $MLD    023372
        024216  000116  CIC$    024216  CID$    024216  CLC$    024216
                        CLD$    024216  $DI     024216  CIF$    024226
                        CLF$    024226  $RI     024226  CIL$    024326
                        CLI$    024332
        024334  000152  $DINT   024334
        024506  000100  CCI$    024506  CDI$    024506  $IC     024506
                        $ID     024506  CFI$    024522  $IR     024522
        024606  000020  $POPR4  024606  $POPR5  024606  $POPR3  024620
TRANSFER ADDRESS = 012656
HIGH LIMIT = 024626
```

APPENDIX H

INTERNAL CODES

Octal Code	Character
200	space
201	+
202	-
203	/
204	*
205	↑
206	(
207	[
210	<
211	)
212	]
213	>
214	comma
215	semicolon
216	CR
217	equals

APPENDIX I

GENERATING FOCAL
(20K RT-11 minimum for assembly)

To generate FORLIB.OBJ for the various hardware arithmetic options, see Appendix L.

PAPER TAPE VERSION (4K):

```
.R MACRO
*PPUB,LP:/N:CND=PAPER,SMALL,PUBLIC
*PFOC,LP:/N:CND=PAPER,SMALL,FOCAL1
*PMAT,LP:/N:CND=PAPER,SMALL,SINGLE,FOCMAT
^C
.R LINK
*PP:,LP:=BIN:PPUB,PFOC,PMAT/F/L
^C
.
```

Paper tape version (8K double precision):

```
.R MACRO
*PPUB8,LP:/N:CND=PAPER,PUBLIC
*PFOC1,LP:/N:CND=PAPER,FOCAL1
*PFOC2,LP:/N:CND=PAPER,FOCAL2
*PMAT8,LP:/N:CND=PAPER,DOUBLE,FOCMAT
^C
.R LINK
*PP:,LP:=PPUB8,PFOC1,PFOC2,PMAT8/F/L
^C
.
```

RT-11 Version:

RT-11 8K Version (SJ Monitor)

```
.R MACRO
*PUB8,LP:/N:CND=SMALL,PUBLIC
*FOC81,LP:/N:CND=SMALL,FOCAL1
*FOC82,LP:/N:CND=SMALL,FOCAL2
```

```
*FMAT8,LP:/N:CND=SMALL,SINGLE,FOCMAT
^C
.R LINK
*FOCAL8,LP:=PUB8,FOC81,FOC82,FMAT8/F
^C
```

RT-11 12K and larger systems (SJ and FB Monitor)

```
.R MACRO
*PUBLIC,LP:/N:CND=TRAP,PUBLIC
*FOCAL1,LP:/N:CND=TRAP,FOCAL1
*FOCAL2,LP:/N:CND=TRAP,FOCAL2
*FMATSP,LP:/N:CND=TRAP,SINGLE,FOCMAT
*FMATDP,LP:/N:CND=TRAP,DOUBLE,FOCMAT
^C
.R LINK
*FOCALS,LP:=PUBLIC,FOCAL1,FOCAL2,FMATSP/F
*FOCALD,LP:=PUBLIC,FOCAL1,FOCAL2,FMATDP/F
^C
.
```

NOTE

/B:loc should be added after /F if additional stack space is required.

FOCALS.SAV - Single Precision FOCAL
FOCALD.SAV - Double Precision FOCAL

```
PAPER.MAC:                 $PAPER=0      ;PAPER TAPE VERSION
                           $TRAP=0       ;USE TRAPS

TRAP.MAC:                  $TRAP=0       ;USE TRAPS

SMALL.MAC:                 $SMALL=0      ;SMALL VERSION

SINGLE.MAC:                $DBL=0        ;SINGLE PRECISION PACKAGE

DOUBLE.MAC:                $DBL=1        ;DOUBLE PRECISION PACKAGE
```

NOTE

The file "TRAP" may be omitted from the 12K and larger versions. This will cause approximately 250-350 more words of memory to be used by the interpreter. An increase in speed of about 15% can be obtained by doing this.

APPENDIX J

GLOSSARY

Term	Definition
Address	A label, name, or number which designates a location where information is stored.
Algorithm	A prescribed set of well-defined rules or processes for the solution of a problem in a finite number of steps.
Alphanumeric	Pertaining to a character set that contains both letters and numerals, and usually other characters.
Argument	1. Variable or constant which is given in the call of a subroutine as information to it.
	2. A variable upon whose value the value of a function depends.
	3. The known reference factor necessary to find an item in a table or array(i.e. the index).
Bug	A mistake in the design or implementation of a program resulting in erroneous results.
Call	To transfer control to a specified routine.
ASCII	Abbreviation for American Standard Code for Information Interchange.
Binary	Pertaining to the number system with a radix of two.

Bootstrap	A technique or device designed to bring a program into the computer from an input device.
Bug	A mistake in the design or implementation of a program resulting in erroneous results.
Call	To transfer control to a specified routine.
Character	A single letter, numeral or symbol used to represent information.
Command	A user order to a computer system, usually given through a terminal keyboard.
Data	A general term used to denote any or all facts, numbers, letters and snt -30 Bug A mistake in the design or implementation of a program resulting in erroneous results. blank 1
Debug	To detect, locate and correct mistakes in a program.
Delimiter	A character that separates, terminates and organizes elements of a statement or program.
Digit	A character used to present one of the non-negative integers smaller than the radix; e.g., in binary notation, either 0 or 1.
Digital Computer	A device that operates on discrete data, performing sequences of arithmetic and and logical operations on this data.
Direct Command	A command entered without a line number which is executed immediately.
Dummy	Used as an adjective to indicate an artificial address, intruction, or record of information solely to fulfill prescribed conditions, as in a "dummy" variable.
EOF	End Of File. This is a logical terminating point for the end of the data placed in a file.
Execute	To carry out an instruction or run a program on the computer.
File	This is a collection of data usually on a mass storage device which can be collectively reffered to by a single name.

Fixed point	The position of the radix point of a number system is constant according to a predetermined convention.
Floating point	A number system in which the position of the radix point is indicated by one part of the number (the exponent) and another part of information inserted solely to fulfill prescribed conditions, as in a "dummy" variable.
Flowchart	A graphical representation of the operations required to n instruction or run a program on the computer.
	30 Function subprogram A subprogram which returns a single value result, usually in the accumulator.
Hardware	Physical equipment; e.g., mechanical, electrical or electronic devices.
Initialize	To set counters, switches and addresses to zero or other starting values at the beginning of, or at prescribed points in, a computer routine.
Interpreter	A program that translates and executes source language statements at run time.
Iteration	Repetition of a group of instructions.
Language, computer	A systematic means of communicating instructions and information to the computer.
Language, source	A computer language such as FORTRAN or FOCAL in which programs are written and which require extensive translation in order to be executed by the computer.
Leader	The blank section of tape at the beginning of the tape.
Line feed	The terminal operation which advances the paper by one line.
Line number	In source languages such as FOCAL, BASIC, and FORTRAN, a number which begins a line of the source program for purposes of identification. A numeric label.
List	1. A set of items. 2. To print out a listing on the line printer or terminal.
Load	To place data into internal storage.

Loop	A sequence of instructions that is executed repeatedly until a terminal condition prevails.
Machine language programming	In this text, synonymous with assembly language programming.This term is also used to mean the actual binary machine instructions.
Matrix	A rectangular array of elements. Any table can be considered a matrix.
Nesting	1. Including a program loop inside a loop.
	2. Algebraic nesting, such as (A+B*(C+D)), where execution proceeds from the innermost to the outermost level.
Octal	Pertaining to the number system with a radix of eight.
Off-line	Pertaining to equipment or devices not under direct control of the computer, or processes performed on such devices.
On-line	Pertaining to equipment or devices under direct control of the computer and to programs which respond directly and immediately to user commands.
Output	Information transferred from the internal storage of a computer to output devices of external storage.
Patch	To modify a routine in a rough or expedient way.
Peripheral equipment	In a data processing system, any unit of equipment distinct from the central processing unit which may provide the system with outside storage or communication.
PIC	Position Independent Code. (Refer to the Paper Tape Software Programming Handbook.)
Program	The complete sequence of instructions and routines necessary to solve a problem.
Pushdown list	A list that is constructed and maintained so that the next item to be retrieved is the item most recently stored in the list.
Radix	The base of a number system; the number of digit symbols required by a number system.

Read	To transfer information from an input device to core memory.
Recursive subroutine	A subroutine capable of calling itself.
Restart	To resume execution of a program.
Routine	A set of instructions arranged in proper sequence to cause the computer to perform a desired task. A program or subprogram.
Software	The collection of programs and routines associated with a computer.
Subroutine	A sequence of program lines that performs a particular operation and returns to the calling line.
Subscript	A number or set of numbers used to specify a particular item in an array.
Symbol table	A table in which symbols and their corresponding values are recorded.
Table	A collection of data stored for ease of reference, generally as an array.
terminal	A peripheral device in a system through which data can enter or leave the computer.
User	Programmer or operator of a computer.
Variable	A symbol whose value changes during execution of a program.
Virtual File	A term used to denote the storage of data in a file, which can be used as if the file were an array in memory.
Write	To transfer information from core memory to a peripheral device or to auxiliary core.

APPENDIX K

DIFFERENCES FROM PAST VERSIONS OF FOCAL-11

All features of past FOCAL-11 versions are supported by this release of FOCAL-11. In addition, several new features have been added:

1.	Numbers	Numerical constants may now allow Octal values.
		Values may be typed in Octal, Binary, or Decimal radix.
		The date is available for output in the RT-11 versions.
		Numerous extensions to the TYPE format (%) values. These include a variable length floating point notation, and removal of restrictions on the size of a format value.
2.	Variables	Variables are now examined for correct syntax structure.
		An extended subscripting mode is now available.
		Virtual files are fully implemented, and can be used with any variable.
3.	Line numbers	Expressions may now be used for line numbers.
		Both the GO and IF commands will accept group numbers as agruments.
4.	Trace	The trace feature has been implemented with an extended version which will allow the user to display the results of any SET or FOR command.

5.	Parameters	Internal FOCAL modifications performed in the past by patches to the FOCAL program are now available as a function which may be altered at run time.
6.	Library	Full library facilities are available for the RT-11 version of FOCAL.
7.	Double Precision	FOCAL-11 is now available in a double precision version which yields approximately seventeen (17) digits of accuracy.
8.	Error Handling	FOCAL errors can now be intercepted by the user program.
9.	Time Scheduling of FOCAL Routines	Up to eight (8) FOCAL routines can be scheduled based upon time. Up to eight (8) software priority levels are available to the user.
10.	Interrupt Processing	Up to (8) FOCAL routines can be scheduled by the detection of a device interrupt. The total number of routines scheduled by both time and interrupts may not exceed eight in the release version of FOCAL.
11.	Functions	Extended mathematical functions are available. These include: Sine, Cosine, Logarithms, and Exponential functions.
12.	UNIBUS Access	Both word and byte transfers are now available.

APPENDIX L

GENERATING FORLIB.OBJ FOR HARDWARE ARITHMETIC OPTIONS

For FORLIB Preparation refer to the appropriate section as listed below.

Section	Hardware Configuration
L.1	Bare machine
L.2	EIS
L.3	FIS
L.4	EAE
L.5	FPU

Underlined text is typed by the system; other text is typed by the user.

L.1 Building FORLIB for a bare machine

```
.R LIBR
*FORLIB=UNI,OTS/G

ENTRY POINT:
$ERRS
$ERRTB

*↑C
```

L.2 Building FORLIB for EIS option

```
.R LIBR
*FORLIB=UNI,EIS/G

ENTRY POINT:
$ERRS
$ERRTB

*↑C
```

L.3 Building FORLIB for FIS option

```
.R LIBR
*FORLIB=UNI,FIS/G

ENTRY POINT:
$ERRS
$ERRTB

*↑C
```

L.4 Building FORLIB for EAE option

```
.R LIBR
*FORLIB=UNI,EAE/G

ENTRY POINT:
$ERRS
$ERRTB

*↑C
```

L.5 Building FORLIB for FPU option

```
.R LIBR
*FORLIB=UNI,FPU/G

ENTRY POINT:
$ERRS
$ERRTB

*↑C
```

APPENDIX M

ASSEMBLING AMD LOADING THE FOCAL LAB EXTENSIONS

The source file LABFNS.MAC contains all the FOCAL lab functions. The user may conditionally assemble it to produce object code for some or all of the functions by defining certain symbols at assembly time. For the AR11/LPS functions, symbols consist of the name of the function less the initial 'F'.

SYMBOL DEFINED	FUNCTION(S) PRODUCED
TIC	FTIC
	FDLY
TOI	FTOI
	FTIC
	FDLY
CRT	FCRT
	FTIC
	FDLY
FRM	FFRM (RT-11 VERSION ONLY)
	FCRT
	FTIC
	FDLY
SAM	FSAM
	FBUF
FIL	FFIL (RT-11 VERSION ONLY)
	FSAM
	FBUF
DMA	FDMA
	FSAM
	FBUF
LED	FLED
BUF	FBUF
FNS	FFNS
BIT	FBIT

As the table indicates, defining certain symbols produces code for more than one function. This occurs when operation of the function named requires existence of some other function or functions. FCRT, for instance, can produce no display without the FTIC function to start the AR11/LPS clock.

The listing of the FOCAL extensions includes comments describing the internal workings of the lab extension functions such as the AR11/LPS display mechanism and character set coding.

The FTOI function leaves room for eight interrupt times unless the symbol 'TIMES' is defined. If defined, its value indicates the number of interrupt times FTOI can save.

LABFNS.MAC may be assembled to produce code usable by either the RT-11 or Paper Tape versions of FOCAL. Defining the symbol '$PAPER' at assembly time produces the Paper Tape object code. When '$PAPER' is defined, setting the symbol 'CORE' to some value indicates the memory size in bytes. Leaving 'CORE' undefined sets memory size to 8k (words). Leaving '$PAPER' undefined at assembly time produces code usable by the RT-11 Linker to produce a version of FOCAL/RT-11 for use with the AR11/LPS.

For distribution on cassette, LABFNS.MAC has been broken into three files. These must be concatenated with PIP before proceeding. With the cassette distribution:

```
    .R PIP
    *LABFN1.MAC=CT:LABFN1.MAC
    *LABFN2.MAC=CT:LABFN2.MAC
    *LABFN3.MAC=CT:LABFN3.MAC
    *LABFNS.MAC=LABFN1.MAC,LABFN2.MAC,LABFN3.MAC
    ^C
```

To produce a 16K Paper Tape version of FOCAL which includes all the functions available for handling the AR11/LPS, proceed as follows:

First, produce object code using the assembler:

```
    .R MACRO
    *LABFNP,LP:=TT:,DK:LABFNS
    ^$PAPER=0
    CORE=100000
    FRM=0
    DMA=0
    TOI=0
    FNS=0
    BIT=0
    ^Z^Z
```

This produces the object module, LABFNP.OBJ.

Link and output to the paper tape punch:

```
    .R LINK
    *PP:,LP:=LABFNP/L
```

This produces a paper tape for overlaying a 16k Paper Tape FOCAL.

To load the paper tape:

1. LOAD 8K PAPER TAPE FOCAL.
2. INSERT THE TAPE CONTAINING THE FUNCTIONS IN THE READER
3. LOAD THE START ADDRESS OF THE ABSOLUTE LOADER AND PRESS 'START'.
4. LOAD ADDRESS 0 AND PRESS START.

This will start FOCAL with all the functions loaded.

To produce a version of FOCAL/RT-11 that includes the AR11/LPS functions:

Assemble LABFNS.MAC:

```
.R MACRO
*LABFNS,LP:=TT:,RK:LABFNS
^FRM=0
ARLPS=0
DMA=0
TOI=0
FNS=0
BIT=0
^Z^Z
```

This produces the object module LABFNS.OBJ.

Next, modify the source file, PUBLIC.MAC, to contain in its function list the names of each function assembled. This involves adding several statements of the form 'FUNCT FXXX,FXXX' starting in location 'FBASE'. Using EDIT:

```
.R EDIT
*EBPUBLIC.MAC$$
FFBASE:$VA$$
FBASE:  FUNCT
I       FUNCT   FCRT,FCRT
        FUNCT   FFRM,FFRM
        FUNCT   FSAM,FSAM
        FUNCT   FBUF,FBUF
        FUNCT   FFIL,FFIL
        FUNCT   FDMA,FDMA
        FUNCT   FLED,FLED
        FUNCT   FTIC,FTIC
        FUNCT   FTOI,FTOI
        FUNCT   FFNS,FFNS
        FUNCT   FBIT,FBIT
$$
*EX$$
```

This adds all the AR11/LPS functions to PUBLIC.MAC.

Reassemble PUBLIC.MAC, LINK THE OBJECT MODULE WITH THE FOCAL language files using either the single precision (FMATSP) or double precision (FMATDP) math package:

```
.R LINK
*FOCAL,FOCAL=PUBLIC,FOCAL1,LABFNS,FOCAL2/F/C [/B:N]
*FMATSP
```

This produces a save file, FOCAL.SAV.

To run FOCAL, type 'R FOCAL'.

CHANGING DEVICE REGISTER AND INTERRUPT VECTOR ADDRESSES

The two symbols in LABFNS.MAC 'REG' and 'VEC' define respectively the base address of AR11/LPS device registers and the first AR11/LPS interrupt vector addresses. If undefined at assembly time, they take on the values 170400 and 340 respectively. For hardware configurations that use different addresses, the user should define one or both symbols at assembly time along with the other symbol definitions. For instance, if the base address of the AR11/LPS device addresses is 170440, the user would set 'REG' equal to 170440 when assembling.

VT11

The VT11 functions will assemble only for RT-11. To include all of the VT11 functions, define 'VT11' at assembly time. When assembling LABFNS.MAC to include the VT11 functions, include in the command string the name of the RT-11 graphics file, VTMAC.

```
.R MACRO
*LABFNS,LABFNS=TT:,SY:VTMAC,LABFNS
^VT11=0
^Z^Z
```

Modify PUBLIC.MAC to include in its function table the names of all VT11 funtions: FVT, FVEC, FMOV, FPT, FSET, FTXT, FSPC, FDIS, FSTA, FSKP, FCLR, FXCO, FYCO, FLP, FSCR.

When linking, include the file of RT-11 graphics routines, VTLIB.

```
.R LINK
FOCAL,FOCAL=PUBLIC,FOCAL1,LABFNS,FOCAL2/F/C
*FMATSP,VTLIB
```

For a hardware configuration with the VT11 device register at an address other than 177200, define the symbol 'VTREG' and set it equal to the value of the VT11 device register address at assembly time. For a base vector other than 320, define 'VTVEC' and set it equal to the new value when assembling.

VT55

The VT55 functions will assemble for both RT-11 and Paper Tape FOCAL. To include them, proceed exactly as described for the AR11/LPS functions but define the symbol 'VT55' instead of (or in addition to) the AR11/LPS related symbols.

For RT-11 modify PUBLIC.MAC to include in its function table the names of the VT55 functions: FGRA, FXY, FGRD, FMRK, FMD0, FMD1, FCUR, FALP.

The user may assemble a version of FOCAL that includes any of the lab extension functions available under RT-11. The most complete form of FOCAL would include all of the AR11/LPS functions, the VT11 functions, and the VT55 functions. Under Paper Tape FOCAL the AR11/LPS functions or the VT55 functions may be included. However, due to size limitations in the Paper Tape FOCAL's function table, AR11/LPS and VT55 functions can not both be included at the same time.

APPENDIX N

SUMMARY OF FOCAL EXTENSION FUNCTIONS

N.1 AR11/LPS FUNCTION SUMMARY

AR11/LPS CLOCK

FTIC(M[,N])

FUNCTION FORMAT: FTIC(N,M), N>0
OPERATION: START/STOP THE CLOCK; SET THE TIMER

FUNCTION FORMAT: FTIC(N), N>=0
OPERATION: RETURN TIMER VALUE, LESS N

FDLY(N[,M])

FUNCTION FORMAT: FDLY(N), N>0
OPERATION: WAIT N TICKS

FUNCTION FORMAT: FDLY(N,FN), N>0
OPERATION: EVALUATE FN UNTIL FN BECOMES >=1 OR UNTIL N TICKS HAVE EXPIRED, WHICHEVER COMES FIRST.

FUNCTION FORMAT: FTIC(0,M)
OPERATION: RETURN TICKS ELAPSED IN PREVIOUS WAIT FUNCTION.

FTOI(V[,N])

FUNCTION FORMAT: FTOI(V), V>0
OPERATION: SAVE TIME OF INTERRUPTS THAT OCCUR THROUGH VECTOR AT ADDRESS V.

FUNCTION FORMAT: FTOI(V,N)
OPERATION: RETURN AR11/LPS TIME OF LAST INTERRUPT THROUGH VECTOR AT ADDRESS V, LESS N.

FUNCTION FORMAT: FTOI(V),V<0
OPERATION: STOP SAVING TIME OF INTERRUPTS THROUGH VECTOR AT ADDRESS V.

ANALOG TO DIGITAL CONVERSIONS

FSAM(N[,M])

FUNCTION FORMAT: FSAM(N), N>=0
OPERATION: RETURN CURRENT VALUE ON AR11/LPS A/D CHANNEL N.

FUNCTION FORMAT: FSAM(N), N<0
OPERATION: WAIT FOR -N EXTERNAL EVENTS BEFORE RETURNING.

FUNCTION FORMAT: FSAM(N,A,B,C,...),N>0
OPERATION: SAMPLE N VALUES, 1 PER CLOCK TICK, INTO THE A/D BUFFER STARTING WITH CHANNEL A, THEN B, THEN C, FOR UP TO 8 CHANNELS, REPEATING THE SAMPLING SEQUENCE AFTER EACH PASS THROUGH THE ARGUMENTS.

FUNCTION FORMAT: FSAM(N,A,B,C,...), N<0
OPERATION: SAMPLE N VALUES, 1 PER EXTERNAL EVENT, INTO THE A/D BUFFER STARTING WITH CHANNEL A, THEN B, THEN C, FOR UP TO 8 CHANNELS REPEATING SAMPLING SEQUENCE AFTER EACH PASS THROUGH THE ARGUMENTS.

FUNCTION FORMAT: FSAM(0,N)
OPERATION: RETURN THE NTH VALUE FROM THE A/D BUFFER.

FBUF(N)

FUNCTION FORMAT: FBUF(N), N>0
OPERATION: ALLOCATE N WORDS AS BUFFER AREA; RETURN BASE ADDRESS OF AREA ALLOCATED

FUNCTION FORMAT: FBUF(N), N=0
OPERATION: DEALLOCATE BUFFER AREA; RETURN 0

FFIL(N,M) (RT-11 ONLY)

FUNCTION FORMAT: FFIL(N,M)
OPERATION: STARTING WITH SAMPLE M, WRITE SAMPLES AS INTEGER VALUES TO THE I/O CHANNEL SPECIFIED BY N. CONTINUE OUTPUT UNTIL AN UNTAKEN OR OVERWRITTEN SAMPLE IS REACHED. IF AN UNTAKEN SAMPLE IS REACHED, RETURN ITS NUMBER. IF AN OVERWRITTEN SAMPLE IS REACHED, RETURN -2.

FDMA(N,M,L)

FUNCTION FORMAT: FDMA(N,M,L)
OPERATION: DO DIRECT MEMORY TRANSFER OF N VALUES FROM A/D CHANNEL M USING THE MODE INDICATED BY THE VALUE IN L

GRAPHICS

FCRT(N[,M<,L>])

FUNCTION FORMAT: FCRT(0,N)
OPERATION: ALLOCATE N LOCS FOR DISPLAY BUFFER

FUNCTION FORMAT: FCRT(N), N>=0
OPERATION: TURN DISPLAY ON (N>0) WITH INTENSITY N OR OFF (N=0).

FUNCTION FORMAT: FCRT(L,N), L<>0
OPERATION: LOAD THE CHARACTER SPECIFIED BY N INTO THE LOC SPECIFIED BY THE ABSOLUTE VALUE OF L. FOR L<0, N REPRESENTS A SPECIAL CHARACTER, 0 TO 4.

FUNCTION FORMAT: FCRT(L,X,Y), L<>0
OPERATION: LOAD THE POINT SPECIFIED BY X AND Y INTO THE LOC WHOSE VALUE IS SPECIFIED BY THE ABSOLUTE VALUE OF L. X AND Y EACH SPECIFY

EITHER AN ABSOLUTE OR RELATIVE COORDINATE. A VALUE PRECEEDED BY A + OR - SIGN INDICATES A RELATIVE COORDINATE. FOR L<0 AND AN ABSOLUTE Y COORDINATE, LOAD THE POINT SPECIFIED BY X AND Y AS AN INVISIBLE POINT.

FUNCTION FORMAT: FCRT(N), N<0
OPERATION: SET CHARACTER SCALING TO SIZE N.

FFRM(N,M) (RT-11 ONLY)

FUNCTION FORMAT: FFRM(N,M)
OPERATION: FOR M>0 SAVE FRAME M IN THE FILE ON CHANNEL N. FOR M<0 RESTORE THE FRAME SPECIFIED BY THE ABSOLUTE VALUE OF M FROM THE FILE ON CHANNEL N.

FLED(N,M)

FUNCTION FORMAT: FLED(N,M) (LPS ONLY)
OPERATION: LOAD THE NUMERIC DISPLAY WITH VALUE M, DISPLAYED WITH N PLACES TO THE RIGHT OF THE DECIMAL POINT.

RAPID FUNCTION EXECUTION

FFNS(F1,F2,...)

FUNCTION FORMAT: FFNS(F1,F2,...)
OPERATION: EVALUATE THE ARGUMENTS SEQUENTIALLY.

16-BIT LOGICAL OPERATIONS

FBIT(V,C1,V1,C2,V2,...)

FUNCTION FORMAT: FBIT(V,C1,V1,C2,V2,...)
OPERATION: TAKE V AS OPERAND, PERFORM THE OPERATION SPECIFIED BY C1 BETWEEN IT AND THE OPERATOR IN V1. REPLACE THE RESULTING VALUE AS OPERAND. CONTINUE THE SAME PROCEDURE FOR ALL CODE/OPERATOR PAIRS THAT FOLLOW.

N.2 VT11 FUNCTION SUMMARY

FUNCTION	OPERATION
FVT(N), N>0	TURN DISPLAY ON
FVT(N), N=0	TURN DISPLAY OFF
FVT(N,L)	SET UP DISPLAY FILE OF AT LEAST L LOCS; TURN DISPLAY ON OR OFF DEPENDING ON THE VALUE OF N; RETURN ACTUAL NUMBER OF LOCS ALLOCATED
FVEC(L,X,Y)	LOAD VECTOR (X,Y) INTO LOC L; RETURN L+1
FMOV(L,X,Y)	LOAD INVISIBLE VECTOR (X,Y) INTO LOC L; RETURN L+1
FPT(L,X,Y)	LOAD POINT (X,Y) INTO LOC L RETURN L+1
FSET(L,X,Y)	LOAD INVISIBLE POINT (XY) INTO LOC L; RETURN L+1

FTXT(L,A,B,C,...)	STARTING IN LOC L LOAD THE CHARACTERS SPECIFIED BY A,B,C..., FOUR CHARACTERS PER LOC; RETURN THE FINAL LOC LOADED PLUS ONE.
FSPC(L,A[,B])	LOAD THE SPECIAL CHARACTER SPECIFIED BY A AND, FOR THREE ARGUMENTS, THE SPECIAL CHARACTER SPECIFIED BY B, INTO LOC L; RETURN L+1
FDIS(T,I,B,S)	SET THE FOUR GRAPHICS MODES THAT WILL PERTAIN TO THE NEXT LOC LOADED
FDIS(L), L>=0	RETURN THE FIRST WORD OF LOC L
FDIS(M), M<0	SET THE FOUR MODES THAT WILL PERTAIN TO THE NEXT LOC LOADED TO THOSE SPECIFIED BY THE VALUE OF M
FSTA(L,C)	LOAD LOC L WITH THE THREE DISPLAY CHARACTERISTICS SPECIFIED BY C
FSKP(LL)	LOAD LOC L WITH A JUMP TO THE END OF THE FILE
FSKP(L1,L2), L2>=0	LOAD LOC L1 WITH A JUMP TO LOC L2
FSKP(L1,L2), L2<0	LOAD LOC L1 WITH A JUMP TO THE START OF THE
FCLR(L)	STARTING AT LOC L, CLEAR THE DISPLAY FILE
FCLR(L1,L2)	CLEAR THE DISPLAY FILE FROM LOC L1 TO LOC L2
FXCO(L)	RETURN THE X COORDINATE OF LOC L; FOR LOC L CONTAINING CHARACTERS, JUMP, OR NULL, RETURN -4095
FYCO(L)	RETURN THE Y COORDINATE OF LOC L; FOR LOC L CONTAINING CHARACTERS, JUMP, OR NULL, RETURN -4095
FLP(N), N>0	RETURN THE LOC OF THE LAST LIGHT PEN HIT
FLP(N), N<=0	WAIT FOR THE NEXT LIGHT PEN HIT TO OCCUR THEN RETURN THE LOC OF THE HIT
FSCR(L,I,Y)	SET TERMINAL SCROLLING FORMAT: L, LINES ON THE SCREEN; I, INTENSITY; Y, TOP Y COORDINATE

N.3 VT55 FUNCTION SUMMARY

FUNCTION	OPERATION
FGRA(N), N=0	TURN ALPHANUMERIC MODE ON
FGRA(N), N=1	TURN GRAPHIC MODE ON
FMD0(D,P,H)	SET DISPLAY ON/OFF, POINTS ON/OFF, HISTOGRAMS ON/OFF
FMD1(I,L,M)	INITIALIZE, TURN HORIZONTAL AND VERTICAL LINES ON/OFF, TURN MARKERS ON/OFF
FGRD(V,SV,DV,H,SH,DH)	PLOT COMPLETE GRID
FGRD(V[,SV[,DV]]), V>0	PLOT VERTICAL LINES
FGRD(0,H[,SH[,DH]])	PLOT HORIZONTAL LINES
FXY(G,SX,EX,EY[,DX])	PLOT A LINE
FXY(G,SX,SY,EX)	PLOT A HORIZONTAL LINE
FXY(G,SX,SY)	PLOT A POINT
FXY(G,SX)	ERASE A POINT
FXY(G)	ERASE A GRAPH
FMRK(G,X[,N])	PLOT OR ERASE A MARKER
FCUR(C,L,A,B,...)	MOVE CURSOR, PRINT VERTICAL LABEL
FCUR(C,L)	MOVE CURSOR
FALP(N)	OUTPUT ESCAPE SEQUENCE

INDEX

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form

Did you find errors in this manual? If so, specify by page.

____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name ________________ Date ________

Organization ____________________

Street ____________________

City ________ State ________ Zip Code or Country ________

If you require a written reply, please check here. ☐

-- Fold Here --

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

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

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

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

UPDATE NOTICE #1

FOCAL-11

User's Manual

DEC-11-LFOCA-F-DN1

March 1976

Insert this Update Notice page in the manual as a means of maintaining an up-to-date record of changes to the manual.

NEW AND CHANGED INFORMATION

This addition provides instructions for using FORLIB with the hardware of a particular system.

INSTRUCTIONS

The following pages are to be placed in FOCAL-11 User's Manual, DEC-11-LFOCA-G-D as replacements for, or additions to, current pages.

Old Page	New Page
N/A	I-3

FOCAL uses FORLIB (FORTRAN library) to process all its math functions. FOCALS.SAV, FOCALD.SAV and FOCAL8.SAV are versions that use the no hardware FORLIB. To use FORLIB with the hardware configuration of a particular system, rename the FORLIB with the extension of that particular option to "FORLIB.OBJ". The FORLIB.OBJ supplied in the kit is the no hardware version.

Example:

```
.R PIP
*FORLIB.NHD=FORLIB.OBJ/R          ;SAVE THE NO HARDWARE FORLIB
*FORLIB.OBJ=FORLIB.EIS/R          ;FORLIB FOR EIS
```

NOTE

The FORLIB OBJ's supplied in the FOCAL/RT-11 distribution are special copies with only the modules included necessary for FOCAL to run. Do not confuse them with the standard RT-11 FORTRAN library.

Please cut along this line.

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form.

Did you find errors in this manual? If so, specify by page.

____
____
____
____
____
____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

____
____
____
____
____
____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

____
____
____
____
____
____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name ______________________________ Date ____________________

Organization ____

Street ____

City ____________________ State __________ Zip Code or Country ____________

If you require a written reply, please check here. ☐

-- Fold Here --

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

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

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

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754