TECHNICAL SPECIFICATIONS

FOCAL

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

CONTENTS (Cont)

CONTENTS (Cont)

CHAPTER 1
INTRODUCTION

FOCAL[†] is a service program for the PDP-8 family of computers, designed to help scientists, engineers, and students solve numerical problems.

FOCAL(TM) language is designed to be used as a tool in a conversational mode; that is, the user creates his problem step by step, while sitting at the computer; as soon as the steps of the problem have been completed, they can be executed and the results checked. Steps can be quickly changed, added, or deleted.

One great advantage of a computer is that once a problem has been formulated, the machine can be made to repeat the same steps in the calculation over and over again. Until now, the job of generating the program was costly, time-consuming, and generally required the talents of a specialist called a programmer. For many modest jobs of computation, a person unfamiliar with computers and programming would use a desk calculator or slide rule to avoid the delays, expense, and bothersome detail of setting up his problem so that the programmer could understand it.

FOCAL circumvents these difficulties by providing a set of simplified techniques that permit the user to communicate directly with the computer. The user has the advantages of the computer put at his disposal without the requirement that he master the intricacies of machine language programming, since the FOCAL language consists of imperative English statements and standard mathematical notation is used.

The FOCAL language is flexible; commands may be abbreviated, and some may be concatenated within the same line. Each input string or line containing one or more commands is terminated by a carriage return.

A great deal of power has also been put into the editing properties of the command language. Normally, deletions, replacements, and insertions are taken care of by the line number which indicates where this line should go or what line is to be replaced. However, if single characters are to be changed within a FOCAL command line, it is not necessary to retype the entire string. The changes may be executed by using the MODIFY command. Thus, complex command strings may be modified quite easily.

In operation, the program indicates that it is ready to receive input by typing an asterisk. On-line command/input may be either direct (to be executed immediately) commands or indirect (to be stored and executed later). An example of a direct command is

```
*TYPE   5*5*5,1    user
        125.000    PDP-8
*
```

---

† Formulating On-Line Calculations in Algebraic Language.

(TM) Trademark of the Digital Equipment Corporation, Maynard, Mass.

The final asterisk indicates that FOCAL is ready for its next command. All commands may be given in immediate mode.

Text input requires that a numerical digit, in the form ab.cd and within a range of 1.01 to 15.99 follow the * . The number to the left of the period is called the group number. The nonzero number to the right is called the specific line or step number. While keying in command/input strings, the rubout key and the left arrow may be used to delete single characters or to kill the entire line, respectively.

Since the command decoder is table driven, FOCAL could be modified by a small binary tape to understand commands in foreign languages.

FOCAL is written especially for the educational market and is intended to be used as a student's problem solving tool. It attempts to give quick and concise reinforcement, to minimize turnaround time, and to provide an unambiguous printed record.

It is also an extremely flexible, high accuracy, high resolution, general purpose desk calculator and demonstration program.

CHAPTER 2
USAGE

2.1 REQUIREMENTS

Any 4K PDP-8 family computer with Teletype may be used with FOCAL.

2.2 LOADING PROCEDURE

a. The RIM or Read-In-Mode Loader must be in memory. (See RIM Loader Manual for a thorough discussion.)

b. The RIM Loader is used to load the Binary Loader. (See the Binary Loader Manual for a complete description.)

c. The Binary Loader is used to load FOCAL.

d. Upon halting, press the CONTINUE key, since the program is loaded in three sections for additional checksum protection.

e. Place 200, the starting address of FOCAL, into the Switch Register when the complete tape has been loaded.

f. Press the LOAD ADDRESS key.

g. Press the START key.

h. The initial dialogue will begin.

2.3 INITIAL DIALOGUE

The program will identify which of the six DEC 12-bit computers you are using and make appropriate corrections to itself. It will then permit you to reject the extended functions to provide extra space, if desired.

FOCAL is ready for your commands when it types * .

2.4 OPERATION

2.4.1 Restart Procedure

There are two possible methods of restarting the system.

Method 1 - Type the character control/C at any time; (FOCAL acknowledges this by typing ?01.00).

Method 2 -

a. Put 200 into the Switch Register.

b. Press the LOAD ADDRESS key.

c. Press the START key.

d. The program will then type ?00.00 indicating a manual restart, and an asterisk indicating it is ready to receive input.

2.4.2 Error Recovery

If an error is made while typing commands to FOCAL, one of the following two methods may be used to recover.

a. Use the RUBOUT key on the teletype keyboard to erase the preceding character. The RUBOUT key echoes \ when typed for each character removed.

Example:

```
*2.70 SETS\SINE = TEMP
*WRITE 2.70
02.70 SET SINE = TEMP
```

b. Use the MODIFY command with the modify control characters to search the command string for any character in error and alter or delete that character. Example is shown in the Command List. Note that the RUBOUT key has the same function while in the modify command mode.

2.4.3 Saving FOCAL Programs

To save a FOCAL text type * WRITE ALL, turn on the punch, type @ marks for leader-trailer, and type carriage return. When all of the program has been typed out, type additional @ marks for more leader-trailer, turn off the punch, and continue your conversation with the computer.

2.4.4 Terminators

Any of the three types of parenthetical pairs may be used in alphanumeric expressions: parentheses (( )), angle brackets (< >), and square brackets ([ ]). The program checks to see whether or not the proper matching terminator has been used at the correct level. Use of these terminators in different configurations should provide additional clarity in reading alphanumeric expressions, especially those which must contain many parenthetical expressions. The only place where normal parentheses must be used is around the expression in the IF command.

2.4.5 Trace Feature

As a further aid to diagnosing or debugging difficulties in a program,

a. a trace feature may be used to find where your errors are, to follow program control, and to create special formats. To operate the trace feature, insert a question mark into a command string at any point other than as the left most character. Each succeeding character will then be typed out as it is interpreted until another question mark is encountered.

2.4.6 Variable Names and Functions

A variable name consists of one or two alphanumeric characters of which the first must be a letter. Additional characters are ignored.

Function names are easily distinguished from variable names because they start with the letter F:

FSIN, FCOS, FATN, FLOG, FEXP, FSQT, FADC,
FDXS, FDIS, FRAN, FSGN, FABS, FITR, FNEW

2.4.7 Error Diagnostics

The error diagnostic printouts are intended to be efficient and yet informative on both a general and explicit level. By using these in conjunction with the trace feature, errors may be pinpointed precisely.

The printout is in the form ?XX.YY. The XX is a category number, and the YY is a specific number derived from the core address of the error call. The categories are:

00 - Console restart by manual control
01 - Interrupt by control - C
02 - Storage or number exceeded
03 - Miscellaneous or illegal character
04 - Format error
05 - Non-existent function or bad format

2.4.8 Arithmetic Priorities

↑
*
/
+ -

Operations of equal priority are executed from left to right (e.g., T 2↑3↑2=+16).

CHAPTER 3
COMMANDS

3.1 TYPE AND ASK STATEMENTS

The TYPE and the ASK statements are used for output and input of literals and alphanumeric calculations. Formatting of input/output is done within the statement itself. The simplest form of the TYPE statement is a command such as TYPE A*1.4. This will cause the program to type = , evaluate the expression, and type out the result. Several expressions of this kind may be typed from the same statement if the expressions are each ended by commas. The ASK statement is similar in form except that only single variable names may be used between commas, and the user types in the values.

3.1.1 Literals

For output of literals, the user may enclose characters between quotation marks. A carriage return will automatically generate closing quotation marks. One unusual character that one might wish to imbed in quotes is the bell, but it may only be inserted during initial input.

3.1.2 Print Positions

Carriage returns are not automatically supplied at the termination of a typeout. In order to supply carriage returns within a TYPE or ASK statement, the exclamation mark (!) is used. This is similar to the use of the slash in FORTRAN format statements.

Occasionally, it is desirable to return the carriage and type out again on the same line without giving a line feed. A number sign (#) returns the print mechanism to the left hand margin but does not feed the paper forward. This feature might be used in plotting another variable along the same coordinate.

3.1.3 Symbol Table

The contents of the symbol table may be typed out to see what the current values are and which variables have been created by TYPE $. The symbol table is typed with subscripts and values in chronological order. The routine then returns as though a carriage return had been encountered in the TYPE statement, thereby terminating the TYPE command. Both the TYPE and the ASK statements may be followed by ; and other commands, unless a $ is in the string.

3.1.4 Output Formats

There is a symbol to change the output format within a TYPE statement: %X.YY, where X and YY are positive integers less than or equal to 19. X is equal to the total number of digits to be output and YY is equal to the number of digits to the right of the decimal point.

On output, leading 0s are typed as spaces. If the number is larger than the field width shows, Xs will be typed. E format is specified by % alone or by %.0X for X decimal points in the E format. (Floating-point decimal: ± 0.XXXXXXXE ± Y where E means "10 to the Yth power.") The current output format is retained until explicitly changed.

3.1.5 Special Characters

The exclamation point (!), percent (%), dollar sign ($), and the number sign (#) may be used after the occurrence of quotation marks or by themselves. They cannot be used to terminate alphanumeric expressions. They may be used in either TYPE or ASK commands.

The TYPE statement precedes its numerical typeouts with an equal sign (=) before beginning the output conversation process. The ASK statement types a colon (:) when it is ready to receive keyboard data.

If the user wishes an expression typed before its results, he may bracket the expression by question marks. This is a special use of the trace feature.

```
*TYPE  ?A*5.2?
A*5.2=+10.40
*
```

3.1.6 Terminators

In the ASK statement, arguments are scanned by the GETARG Recursive Routine and may therefore be terminated by any legitimate terminating character (e.g., space, comma, * , etc.). In the TYPE statement, arguments are scanned by the EVAL Recursive Routine and must therefore be terminated by comma, semicolon, or carriage return. In either, command arguments may be preceded by format control characters # ! ".

3.1.7 Input Formats

Keyboard responses to the ASK inputs may

a. have leading spaces

b. be immediately preceded by + or – sign if desired or required

c. be in any fixed point or floating point format

d. be terminated by any terminating character, carriage return, or ALTMODE. However, it is recommended that the space be adopted as the conventional and general purpose input terminator. The ALTMODE is a special nonprinting terminator that may be used to synchronize the program with external events. For example, if you wish to insert special paper in the teletype before executing the program, type Ask A; GO and RETURN, then load your paper, and hit ALTMODE.

3.1.8 Alphanumerics

Input data that is in response to an ASK command may take any format, may be signed or unsigned, and must be terminated by a legitimate terminating character (space, CR, comma, /, etc.). This means that alphabetic input may also be accepted by an ASK input command. This is done by a simple hash-coding technique so that the program can recognize keyboard responses by a single compare. See example under the IF command for an illustration of how to program the recognition of the user reply "WAIT".

3.1.9 Off-Line Tapes

To prepare data tapes off-line, type the data word, the terminating space, and the "here-is" key. Use backspace and rubout to remove characters off-line. (See technical specs for alternate use without interrupts.)

3.1.10 Corrections

For editing of input to an ASK command before the input has been terminated, the left arrow (←) is used.

3.1.11 Roundoff

Numbers to be typed out are rounded to the last significant digit to be printed (i.e., the rightmost digit of the requested format) or to the sixth significant digit, whichever is smaller.

3.2 THE DO COMMAND

The DO command is used chiefly to form subroutines out of single lines, groups of lines, or of the entire text buffer. Thus, the instruction DO 3.3 makes a subroutine of line 3.3. For a single line subroutine, control will be returned when the end of the line is encountered or when the line is otherwise terminated (such as by a RETURN statement, or in the case of TYPE, with the $).

One of the most useful features of a command language of this type is the ability to form subroutines out of entire groups. Thus, the statement DO 5 calls all of group 5 as a subroutine beginning with the first group 5 line number. Control will then proceed through the group numbers going from smaller to larger. A RETURN or EXIT is generated from this type of subroutine by using the word RETURN, or by encountering the end of that group, or by transferring control out of that group via a GOTO or IF command. Similarly, the entire text buffer may be used as a recursive subroutine by simply using DO or DO 0.

The DO statement may be concatenated with other legitimate commands by terminating it with a semicolon. Thus, a single line could contain a number of subroutine calls. In this way, several forms of complex subroutine groupings may be tested from the console.

The number of DO commands which may be nested linearly or recursively is limited only by the amount of core storage remaining after inclusion of the text buffer and the variable storage.

NOTE

> When a GOTO or IF statement is executed within a DO subroutine, control is transferred immediately to the object line of the GOTO command. That line will be executed and return made to the DO processor. If the next line number is within the group (if this is a group subroutine) it will be executed. If, however, a line number outside of that group is about to be executed, then a return will be made from the DO subroutine and the remainder of the DO command line, if any, will be processed.

3.3 EDITING AND TEXT MANIPULATION FACILITIES

Line numbers which have already been used and are used again in a new input will cause the new input to replace the line that previously had that number. Insertions are made at the appropriate point in a numerically ordered string of lines. For example, line number 1.01 (the smallest line number) will be inserted in front of (or above) line number 1.1. The largest line number is 15.99.

Removal of a single line may be made by using the ERASE command. For example, ERASE 2.2 will cause line 2.2 to be deleted. No error comment will be given if that line number does not exist. The command ERASE 3 or 3.0 will cause all of group 3 to be erased. To delete all of the text, one must type the words ERASE ALL. This insures that all text is not erased accidentally.

ERASE, used alone, has the function of merely removing the variables. This may also be thought of as initializing the values of the variables to zero.

In order to examine the contents of a line, one may type WRITE 3.3. This will cause line 3.3 to be typed out with its line number on the Teletype. WRITE 4.0 will cause all of group four to be written on the Teletype. The WRITE ALL will cause all of the text to be printed on the Teletype, left justified with title and line numbers in numerical order. The WRITE and ERASE commands may not be followed by any other commands.

Often only a few characters need be changed in a particular line. To facilitate this job, so that the entire line does not have to be replaced, we have included the properties of the MODIFY command. Thus, to modify characters in a line, one would type MODIFY 5.41, in order to modify the characters of line 5.41. This command is terminated by a carriage return, whereupon the program waits for the user to type that character at which he wishes to make changes or additions. After he has done so, the program will type out the contents of that line until the search character is typed. (The search character is not echoed when it is first keyed in by the user.) The program will now accept input.

At this point, the user has seven options. These are

a. to type in new characters in addition to the ones that have already been typed out.

b. to type a form-feed. This will cause the search to proceed to the next occurrence, if any, of the search character.

c. type a bell which allows him to change the search character just as he did when first beginning to use the MODIFY command.

d. use the rubout key to delete characters going to the left.

e. type a left arrow to delete the line over to the left margin.

f. type a carriage return to terminate the line at that point and move the text to the right.

g. type line-feed to save the remainder of the line.

The ERASE ALL and MODIFY commands are generally used only in immediate mode since they return to command mode upon completion. The reason for this is that internal pointers may be changed by these commands.

During command/input the left arrow will delete the line numbers as well as the text. During the MODIFY command the left arrow will not delete the line number.

When the rubout key is struck a backslash (\ ) is typed for each character that is deleted.

Any modifications to the text will cause the variables to be deleted as if an ERASE command had been given. This is caused by the organization of our data structure. It is justified by the principle that a change of program probably means a change of variables as well.

3.4 THE FOR STATEMENT

This command is used for convenience in setting up program loops and iterations. The general format is: FOR A = B,C,D;---. The index A is initialized to the value B, then the command string following the semicolon is executed. When the carriage return is encountered, the value of A is incremented by C and compared to the value of D. If A is less than or equal to D, then the command string after the semicolon is executed again. This process is repeated until A is greater than D.

A must be a single variable; B, C, and D may all be expressions, variables, or numbers. The computations involved in the FOR statement are done in floating point arithmetic. If comma and the value C are omitted, then it is assumed that the increment is one.

Example: SET B = 3; FOR I= 0,10 ; TYPE B↑I, !

3.5 THE CONDITIONAL IF STATEMENT

In order to provide for transfer of control after a comparison, we have adopted the IF statement format from FORTRAN. The normal form of the IF statement contains the word IF, space, a parenthesized expression, and three line numbers separated from each other by commas. The program will GOTO the first line number if the expression is less than zero, the second line number if the statement has a value of zero, and the third line number if the value of the expression is greater than zero.

Alternative forms of the IF command are obtained by replacing the comma between the line numbers by a semicolon. In this case, if the condition is met which would normally cause the program to transfer to a line number past that position, then the remainder of the line will be executed. Thus, if one desires only a two way match, you may say "IF (expression) line number; other command".

Example:

```
IF (REPLY - 1WAIT + 10000) 6.4,5.01;RETURN
IF (REPLY - 1YES + 19000) 6.3,5.02;6.3
```

3.6 THE GOTO COMMAND

This command causes control of the program to be transferred to the indicated line number. A specific line number must be given as the argument of the GOTO command. If command is initially handed to the program by means of an immediately executed GO, control will proceed from low numbered lines to higher numbered lines as is usual in a computer program. Control will be returned to command mode upon encountering a QUIT command, the end of the text or a RETURN at the top level.

The operation of the GOTO is slightly more complicated when used in conjunction with a FOR or a DO statement. Its operation is perfectly straightforward when used with any other statement.

3.7 THE RETURN COMMAND

The RETURN command is used to exit from DO subroutines. It is implemented by merely setting the current program counter to zero. When this situation is encountered by the DO statement it exits. (Refer to the DO command, Section 3.2.)

3.8 THE QUIT COMMAND

A QUIT causes the program to return immediately to command/input mode, type * , and wait.

3.9 THE COMMENT STATEMENT

Beginning a command string with the letter "C" will cause the remainder of that line to be ignored so that comments may be inserted into the program.

3.10 THE CONTINUE STATEMENT

This word is used to indicate dummy lines. For example, it might be used to replace a line referenced elsewhere without changing those references to that line number.

3.11 THE SET STATEMENT

The SET command for arithmetic substitution is used for setting the value of a variable equal to the result of an expression. The SET statement may contain function calls, variable names, and numerical literals on the right hand side of the equal sign. All of the usual arithmetic operations plus exponentiation, may be used with these operands. The priority of the operators is a standard system: +-/*↑ . These, however, may be superseded by the use of parenthetical expressions. The SET statement may be terminated by either a carriage return or a semicolon, in which case it may be followed by additional commands.

```
SET AA=B*(5+<6+CONST>*ALPHA/[5/BETA])
```

CHAPTER 4
PROGRAM SPECIFICATIONS

4.1 MACHINE REQUIREMENTS

The minimum hardware configuration necessary to run this program is a 4K PDP-8 family or PDP-5 computer with ASR-33.

EAE for speed, scope and an additional 4K memory for text storage are potential options.

4.2 DESIGN SPECIFICATIONS

4.2.1 Design Goals

This is a JOSS * -like or FORTRAN-like conversational language and operating system for a basic PDP-8. It is designed to provide ease and power for on-line editing and execution of symbolic programs.

4.2.2 Input

Either the keyboard or the low-speed reader is used for input of program text. The keyboard is also used for typing commands to be executed immediately. Keyboard input is single buffered internally.

4.2.2.1 Input Format- See the description of the commands in Chapter 3 for format information.

4.2.2.2 Character Set - Input and output characters are in ASCII teletype code.
Interpretive operations are also done internally in ASCII.
The text buffer is packed two characters to a word as follows.

number = represented as: prints as
300 = not packed = ignored: @
301 - 336 = 01 - 36: A - Z
337 = not packed - edit control, kill line: ←.
240 - 276 = 40 - 76: symbols
277 = 37: ?.
340 - 376 = 7740 - 7776 (extended codes): non-printing

---

* JOSS is a copywrited name of the RAND Corporation.

377 = not packed - edit control, delete preceding character; if a character is deleted, \ (backslash) is typed.

200 = not packed - ignored: leader-trailer

210 - 237 = 7701 - 7737: control characters

000 = not packed - ignored: blank tape.

4.2.3 Output

4.2.3.1 Output Format - See the TYPE and WRITE statements for format of output. The output character set is the same as that for input.

4.2.3.2 The Input/Output and Interrupt Processor - The purpose of the interrupt handler and the I/O buffers is to permit input and output to proceed asynchronously with calculations. This allows an optimal use of the computer time. When the interrupt handler finds that the teletype output flag has been raised, it clears that flag and looks to see whether there are any additional characters in the teletype output buffer to be printed. If there are, it takes the next character from the buffer, prints it, clears that location in the buffer, and moves the pointers. Separate pointers are maintained for both the interrupt processor and for the program output subroutine (XOUTL). If the interrupt handler finds that there are no more characters to be output on the Teletype, it will clear a teletype in-progress-switch called TELSW. If it does output another character it sets TELSW to a nonzero value.

When the program desires to place characters in the buffer for the interrupt processor to print, it makes a call to XOUTL. This routine first checks to see whether or not TELSW has been set. If TELSW is zero, then no further interrupts are expected by the interrupt processor so the output routine immediately types the character itself and sets TELSW to a nonzero value. Otherwise, if the interrupt processor is in motion, then the output routine places the character into the buffer and increments the pointer. If there is no room in the buffer for additional characters, the low speed output routine waits until there is. The keyboard input processors are similar in organization to the output routines except that no in-progress-switch is needed and the input is only double buffered.

Another advantage of using the interrupt system is that it enables you to stop program loops from the keyboard by typing Control C. The recovery routine will then reset the I/O pointers, type out the message code ?Ø1.ØØ, and return to command mode. Manual restart via the console switches also goes to the recovery routine, resets the pointers, and types out message code ?ØØ.ØØ. In fact, all error diagnostics go to the recovery routine. Error printing is withheld until prior printing is complete. Otherwise, on occasion, a full buffer could be dumped and the error message could be printed as many as 16 characters before it should have otherwise occurred. This would be misleading when using the trace mode to discover specific errors within a character string.

The recovery routine may also be called by the interrupt processor if it discovers that there is no more room in the keyboard buffer. This could occur for example, if the user continued to type on the keyboard while the program was making computations. He should notice something unusual because his characters would not be echoed back as he typed.

This error could also occur when reading a paper tape program into the text buffer. If the output hardware were slower than the input hardware, more text would be read in than was being read out of the buffer with the result that the program would not empty the reader buffer as quickly as it was being filled up, since the program synchronizes the reading of the characters with sending them into the buffers. In other words, the program synchronizes its side of the I/O buffers, but the interrupt side of the I/O buffers proceeds at a rate determined by the hardware. To guard against incurring this type of error with long input tapes, which were prepared off line, carriage returns may be followed by some blank tape which is ignored by the input routines, thereby giving the output routine time to catch up.

4.2.4 Organization

4.2.4.1 The Internal Structure

a. Part 1 - Arithmetic Package - The arithmetic is done in the floating point system. The three-word floating point package allows six digits of accuracy plus the extended functions. The program will also be able to use four words without the trigonometric functions. The largest of the floating point packages occupies locations 4600 - 7577. Both packages have an exponential range of approximately ten to the six hundredth.

The four-word floating point system creates ten digits of accuracy, including roundoff. It does, however, require more storage for variables and for push-down-list data.

b. Part 2 - Storage - The major components of the program occupy locations 1 - 3220. The remaining storage 3220 - 4600 is used for text storage, variable storage, and push-down storage, in that order. The text occupies approximately two characters per register. The variables occupy either five or six locations per variable depending on whether the three or four-word option is utilized. Remaining storage is allocated to the push-down list. Overflow will occur only when this push-down list exceeds the remaining storage. This could happen in the case of complex programs which have multiple levels or recursive subroutine calls.

The push-down list contains three kinds of data. One of these is a single location for push-jump and pop-jump operations. The content of the accumulator is also pushed into the same list in a single register. The third type of push-down storage is floating point storage.

This storage allocation scheme permits flexibility in the trade off of text size, number of variables, and complexity of the program, rather than restricting the user to a fixed number of statements or characters, or to a fixed number of subroutine calls, or to a limited number of variables.

4.3. HARDWARE ERRORS

The 8/S will halt at location EXIT +2 if a parity error occurs.

4.4. INTERNAL ENVIRONMENT

4.4.1 Floating-Point Arithmetic System

The FOCAL system was designed to be easily interfaced for new hardware such as LAB-8, multiplexed ADC's real-time clocks, or to software such as a nonlinear function.

The information given below, the symbol table, the various lists, and a core layout are sufficient for all required modifications and patches. This symbolic approach ensures greater flexibility and compatibility with DEC modifications to FOCAL, other user's routines, and assembly via PAL III on a PDP-8.

Example: Suppose we had a scope routine to display characters at a given point on a scope. We will call this routine from FOCAL as function by FNEW (X, Y, SHOW). Here X and Y are expressions to be used as display coordinates for the start of SHOW.

First we patch the function branch table:

```
*FNTABF + 12
XFNEW
```

When control arrives at XFNEW the X has already been evaluated:

```
XFNEW,   JMS    I    INTEGER     / make 12-bits
         TAD         FLAC + 1
         DXL                     / set X -coor.
         CLA
```

Now we should test for the possibility of another argument;

```
         SPNOR                   / ignore spaces
         TAD         CHAR
         TAD         MCOMMA
         SZA         CLA
         JMP    I    EFUN3I      / no more
```

Move past the separating comma;

```
        GETC
```

Test for the end of the parentheses;

```
        TAD             CHAR
        TAD             RPAR
        SNA             CLA
        JMP     I       EFUN3I          / exit
```

Evaluate the second argument;

```
        PUSHJ
                        EVAL
        JMS     I       INTEGER
        TAD             FLAC + 1
        DYS; CLA        /Set Y and intensify
        SPNOR
        TAD             CHAR
        TAD             MCOMMA
        SZA             CLA
        JMP     I       EFUN3I
```

Now we are ready to pick up the single letters for display until the end of the function is reached.

```
DCHR,   GETC
        TAD     CHAR
        TAD     RPAR
        SNA     CLA
        JMP  I  EFUN3I
```

Char. display routine called

```
        JMS     DCHR
```

We now need a few definitions from the symbol table.

```
        FLAC = 44
        EFUN3I = 106
        CHAR = 152
        SPNOR = 4527
```

<u>Summary</u>:

a. User defined functions must leave their value, if any, in FLAC and return by a JMP I EFUN3I.

b. FLAC is converted to an integer in FLAC + 1 by a JMS I INTEGER.

c. The floating point arithmetic interpreter is entered by JMS I 7.

d. The address of the user's function is placed by him in the FNTABF list.

e. Location BOTTOM contains the address of the last location to be used for storage. If BOTTOM is made to contain 4277, for example, then the user has from 4300 to 4577 for storage of his function processor. The user is requested to achieve his function implementations using the information given here and in the symbol table without needing the actual listing so that changes made by different users may be compatible and so that they might also be relocated easily should any changes be made by DEC.

f. The argument following the function name is evaluated and left in FLAC before control is transferred to the particular function handler. Since evaluation is terminated by either , or a right parenthesis, a special function could have more than one argument.

Only in the case of multiple arguments does a user need to worry about saving his working machine language storage for a possible recursive use of his function. The contents of the AC are saved by PUSHA and restored by POPA for this purpose. If there is another argument, it may be evaluated by PUSHJ ; EVAL.

4.4.2 Internal Subroutine Conventions

4.4.2.1 Calling Sequences - The (AC) = Ø unless it contains information for the subroutines. Upon returns (AC) = Ø unless it contains data.

There are six types of routines and subroutines used in the implementation of this program.

a. Normal subroutines called by an effective

```
        JMS         SUBR1
```

which contain zero at their entry point

```
        SUBR1, Ø
```

and a return by a

```
        JMP    I    SUBR1
```

b. New instructions called by

```
        PRNTLN          / (to print a line number)
```

and usually defined by

```
        PRNTLN = JMS  I.
        XPRNT
```

where XPRNT is the entry point for a normal subroutine. These new instructions may have multiple returns and/or multiple arguments:

```
SORTJ                          /call;
            LIST6-1            /data list minus one;
            INLIST-LIST 6      /increment to branch table
                               /return if CHAR is not in LIST 6
```

These new instruction subroutines often have implied arguments, e.g., GETC, READC, PACKC, TESTC, and SORTC all use the variable CHAR as their argument. The new instructions SORTJ and PRINTC use CHAR only if the AC is zero. If the AC is nonzero, then that value is used. Still others use only the AC for their argument: RTL6, TSTLPR, PUSHA, and TSTGRP.

c. Recursive routines called by

```
        PUSHJ           /call
        EVAL            /address
        '               /return
```

Where the address contains the first instruction of the routine. The return address is kept in the push-down list, and exit is made by use of

```
        POPJ            /exit subroutine.
```

Such routines may call each other or themselves in any sequence and/or recursively by saving data on the push-down list. Others are EVAL, DELETE, PROCESS, PROC, and GETVAR.

d. Command processor routines to handle specific command formats are called by

```
    SORTJ                       /go to command
            CONLST-1
            COMGO-COMLST
    ERROR 3                     /illegal command
```

The individual command routines use only new instructions and recursive routines. They may exit in one of three possible ways:

(1) POPJ - if C.R. is encountered or

(2) transfer to another command routine or

(3) transfer to START.

e. Floating point groups of interpretive instructions similar to the following format:

```
FINT                            /enter floating interpreter,
FGET        FLARG
FMPY    I   PT1
:
EPUT        FLARG
FXIT                            /leave floating interpreter.
```

f. Main processor to handle text input and keyboard commands. This routine could be "locked out by an instructor to protect and execute a stored program repeatedly.

```
    IBAR, JMP GONE + 11
```

Similarly, selected commands are easily deleted by the instructor by placing zero in the appropriate locations in COMLST.

Line number input and explicit replacements are "short-circuited" by

```
    INPUTX + 4, NOP
```

4.4.2.2 Subroutine Organization - Figure 4-1 illustrates the internal use of various subroutines.

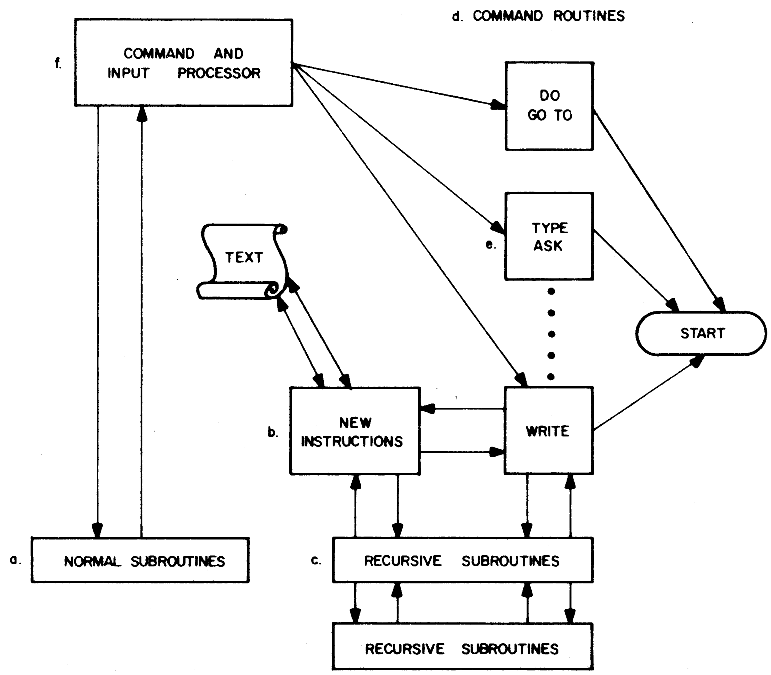


Figure 4-1

4.4.3 Character Sorting

If a program must contend with a number of different characters (or 11-bit items) each of which can initiate different responses, we simply look up the address of the action that corresponds to a given symbol or bit pattern. If the symbols do not form a continuum, the programmer must find the most efficient method for determining the corresponding address.

The method used in FOCAL is the table sort and branch. This method uses a subroutine to match up an input character with one member of a list of characters. The call to the subroutine is followed by

a. the address minus one of the list and

b. the difference between that list and a second list. The latter list contains the corresponding addresses. Thus if a match is found in the first list, the difference is added to the address of that match to computer the address in the second list which contains the name of the action to be performed.

c. The next instruction to be executed if a match is not found.

In addition to being simple and concise, although more time consuming than other methods, this technique has another advantage that is especially useful in a PDP-8: the tables may be placed at page boundaries to take up the slack that often occurs at the end of a page. This results in a more efficient use of available core storage.

4.4.4 Language

The program is written in PAL III with floating point commands plus program defined commands implemented as subroutine calls.

APPENDIX A

FOCAL COMMAND SUMMARY

Command	Abbr	Example of Form	Explanation
TYPE	T	TYPE FSQT (AL ↑ 3+FSQT (B))	Evaluates expression, types out =, and result in current output format
		TYPE "TEXT STRING"!	Types text. Use ! to generate carriage return line feed.
WRITE	W	WRITE ALL	FOCAL prints the entire indirect program.
		WRITE 1	FOCAL types out all group 1 lines.
		WRITE 1.1	FOCAL prints line 1.1
IF	I	IF (X) 1.2,1.3,1.4;	Where X is identifier or expression.

Control is transferred to the first, second, or third line number if (X) is less than, equal to, or greater than zero respectively. If the semicolon is encountered prematurely then the remainder of the line is executed.

MODIFY	M	MODIFY 1.15	Enables editing of characters on line 1.15

The next character typed becomes the search character. FOCAL will position itself after the search character; then the user may

a. type new text, or

b. form-feed to go to the next occurence, or

c. bell to change the search character, or

d. rubout to delete backwards, or

e. left arrow to kill backwards, or

f. carriage return to end the line, or

g. line-feed to save the rest of the line.

QUIT	Q	QUIT or * or control-C	Returns control to user.
RETURN	R	RETURN	Terminates DO subroutines
SET	S	SET A = 5/B * SCALE(3)	Substitution statement
ASK	A	ASK ALPHA (I + 2 * J)	FOCAL types a colon for each variable; the user types a value to define each variable.
COMMENT CONTINUE	C C	COMMENT C	If a line begins with the letter C, the remainder of the line will be ignored.

Command	Abbr	Example of Form	Explanation
DO	D	DO 4.14	Execute line 4.14; return
		DO 4	Execute all group 4 lines, return when group is expanded or when a RETURN is encountered.
		DO ALL	Execute entire indirect text as a subroutine.
ERASE	E	ERASE	Erases the symbol table.
		ERASE 2	Erases all group 2 lines.
		ERASE 2.1	Deletes line 2.1.
		ERASE ALL	Deletes all user text.
FOR	F	FOR I = x,y,z; TYPE I	The command string following the semicolon is executed for each value. x,y, z are constants, variables, or expressions. x = initial value of I, y = value added to I until I is greater than z. y is assumed = 1 if omitted.
GO	G	GO	Starts indirect program at lowest numbered line number.
GOTO	G	GOTO 3.4	Starts indirect program at or line 3.4

C - The Fourteen (14) Functions are

FSQT ( ) - Square Root
FABS ( ) - Absolute Value
FSGN ( ) - Sign Part of the Expression
FITR ( ) - Integer Part of the Expression
FRAN ( ) - A Noise Generator
FEXP ( ) - Natural Base to the Power
FSIN ( ) and - FCOS ( ), FATN ( ) - Trig Functions
FLOG ( ) - Naperian Log
FDIS ( ) and - FDXS ( ) - Scope Functions
FADC ( ) - Analog to Digital Input Function
FNEW ( ) - User Function

ASK/TYPE CONTROL CHARACTER TABLE

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

APPENDIX B

ERROR DIAGNOSTICS

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

NOTE

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

APPENDIX C

TO SAVE BINARY OF INITIAL DIALOGUE

1. Load FOCAL and FLOAT;
2. start at 2ØØ;
3. type CNTL-C and "Erase All";
4. read in init-dialogue program (Dialog);
5. load JR46;
6. start at 46ØØ;
7. type T;
8. turn on the punch (low speed);
9. wait for leader-trailer;
10. stop computer, turn punch off;
11. restart at 46ØØ;
12. type 144; 144P;
13. turn punch on, hit continue;
14. when punching stops, turn punch off;
15. type 165; 165P;
16. turn punch on, hit continue;
17. when punching stops, turn punch off;
18. type 3240; 4276P;
19. turn punch on, hit continue
20. when punching stops, turn punch off;
21. type "E";
22. turn punch on, hit continue;
23. When some leader-trailer has been punched, stop the computer:
24. You have punched the binary of the initial dialogue.

    i.e., C(BUFR), C(LASTV), and C(FRST to C(BUFR)).

    For generating the Error Diagnostic Codes

    NOP-location CHINX-1 (2475)

APPENDIX D

FOCAL CORE LAYOUT-USAGE

Free	Used	Mnemonics	What
	0	ZERO	
	173		
2	0200	START	FOCAL PROPER
	3251		
Ø	3252	BUFBEG	BUFFER AREA
	4377		
Ø	4400	BEGIN	INITIAL DIALOGUE
	4577		
	4600	FEXP	EXTENDED FUNCTIONS
	4775		
2		(BET 2+ 3)	
	5000	ARTN	
	5166		
11		(FLAG 3 +1)	
	5200	FCOS	
	5365		
12		(FLOA + 11)	OUTPUT CONVERSION
	5400		
	5576		
1		(TEMPO + 1)	
	5600	DECONV	
	5752		
25		(INFIX + 5)	INPUT-OUTPUT ROUTINES
	6000	FLOUTP	
	6175		
2		(OUTOG+4)	
	6200	FLINTP	
	6315		
62		(P43+1)	FLOATING-POINT INTERPRETER
	6400	FPNT	
	6576		
1			
	6600	ACMINS	
	7355		
22		(RAR1+1)	
	7400	DNORM	
	7556		
21		(BUFFER + 10)	LOADERS
	7600	BINARY	
	7777	(RIM)	

FOCAL CORE LAYOUT - DETAILED

Page 0 - Field 0

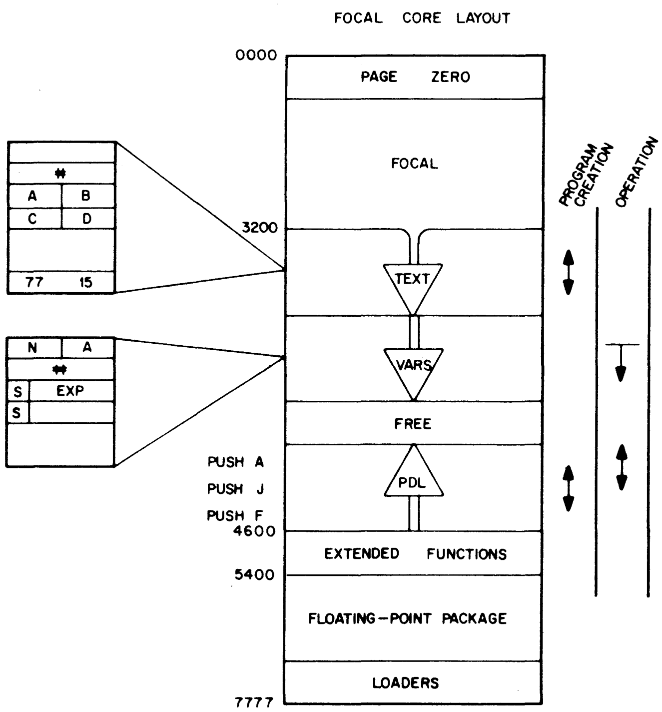
* 001	Miscellaneous
	Numbers
	Floating-Point Working Area
	Constants
	New Instruction Pointers
	Variables
* 200	
	START
	Command/Input
"GETLN -	Line Read Routine
* 400	'DO' Routine
	Push-POP Routines
* 600	'GOTO' and WRITE ' and Misc.
*1000	'IF', "SET", 'FOR' and Misc.
*1200	'ASK', 'TYPE', 'MODIFY'
*1400	
	"GETARG" - Recursive Routine
	"SPNOR", "TESTN", "POPJ"
	'RETRUN'
*1600	"EVAL" - Recursive Routine
	OPNEXT - read operator
	ARGNXT - read operand
	ETERM - evaluate terminator
	FLOP - floating operations called
	ENUM - number processor
	EFUN - function processor
*2000	ELPAR - left parens processor
	EFUN3 - function returns
*2100	"DELETE" - Recursive Routine
	DOK - group delete
	DONE - garbage collection
*2200	"FINDLN" - Normal Routine
	Find exact match or next larger
	'ERASE' command processor
*2300	"GETC" - unpack text and trace
*2400	"ENDLN", "PRNTLN"
	I/O Subroutines
	Command Buffer
*2600	Interrupt Processor
	ERROR Processor
*3000	"PACKC" - pack text
	Rubout routine

*3220		I/O Buffer
*3240		Text Buffer Begin
T		
E		
X	*4400 -	Once-Only Code
T		SELF-START
/		TEST ARITH
V		TEST EAE
A		TEST X-MEM
R		CLEAR ALL FLAGS
I		TYPE MESSAGE
A		
B		
L		
E		
S		
/.		
./		
P		
U		
S		
H		
D	*3600	ODT-JR (for X-FUN)
O	*4000	ODT-JR (standard)
W	*4600	ODT-JR (for dialogue)
N		
L		
I		
S		
T		

Floating Point Routines

*4600	Extended Functions
*5400	I/O Controller
*6400	Basic
*7600	Binary Loader or 8-SYS LIB Bootstrap or Disk Bootstrap
*7756	Rim Loader
End of Field	ZERO
Field ONE	Extended Text Storage

FOCAL CORE LAYOUT - DYNAMIC STORAGE



APPENDIX E

SYMBOL TABLE

AUD	0145	EFUN	1736	FLTONE	2366	LCON	0373
ALIST	1446	EFUN2	1747	FLTXR	0015	LIBRAR	2737
ARGNXT	1716	EFUN3	2040	FLTZER	2370	LINENO	0153
ASK	1214	EFUN3I	0106	FNPT	4550	LISTGO	1402
ATES	4476	ELPAR	1766	FNTABF	0374	LISTN	1566
ATLIST	1164	END	0104	FNTABL	2165	LIST3	0070
ATSW	0141	ENDLN	4525	FOR	1041	LIST6	0064
AXIN	0010	ENDT	0105	FOUTPU	0100	LOOKUP	4571
AXOUT	0017	ENUM	1725	FPNT	6400	LPRTST	2061
BEGIN	4400	EPAR	1703	FRST	3240	L2	1321
BELL	0066	EPAR2	1770	FRSTX	3250	L8A	4544
BFX	4553	ERASE	2203	GEND	2340	L8AX	4547
BFXX	4552	ERG	2224	GETARG	1417	L8AY	4546
BOTTOM	0027	ERL	2221	GETC	4514	L8B	4545
BOX	2516	ERROR2	4534	GETLN	4523	MASK	0026
BUFBEG	3252	ERROR3	4535	GETVAR	1423	MBREAK	2607
BUFR	0144	ERROR4	4536	GET1	2334	MCR	0073
CCR	0070	ERROR5	2737	GET3	2351	MEQ	1136
CDF	7000	ERR2	2743	GEXIT	0357	MF	0604
CFRS	0103	ERR3	2723	GFND1	1521	MFLT	0074
CHAR	0152	ERR4	2731	GINC	0154	MINSKI	0053
CHIN	2467	ERR5	2735	GLIST	1413	MINUSA	0035
CHINX	2476	ERV	2216	GONE	0234	MODIFY	1301
CLF	0067	ERVX	2237	GOTO	0605	MOV1	3113
CNTR	0142	ER5	4551	GRPTST	0745	MPER	1546
COL	1266	ESCA	3110	GSERCH	1442	M100	0072
COMBOT	0225	ETERM	1642	GS1	1461	M11	0076
COMBUF	0102	ETERMN	1637	GS2	1501	M12	0014
COMEIN	2522	ETERM1	1616	GS3	1463	M137	2365
COMGO	1175	ETERM2	1650	GS4	1476	M140	3047
COMLST	0757	EVAL	1602	GTEM	0021	M2	0034
COMMEN	0616	EXIT	2652	IBAR	0211	M20	0025
COMOUT	2605	EXTR	2317	IF	1011	M240	1545
CSTAR	0224	FCONT	1102	IF1	1037	M340	3076
C100	0006	FEND2	2271	IF2	1015	M40	2364
C140	3056	FEND3	2272	IF3	1026	M5	0075
C200	0016	FEXP	4600	IGNOR	0216	M77	0023
C260	0063	FINCR	1066	ILIST	1002	NAGSW	0151
C77	0077	FINDLN	4524	ILISTA	0777	NOTIN	2377
DCONT	0463	FINUN	2247	ILIST1	1005	ONE	0455
DDTJR	0004	FINFIN	1137	INBUF	2666	OOUT	4540
DEBGSW	0161	FINPUT	0101	INDEV	0150	OP	3176
DECON	5622	FISW	0051	INLIST	0571	OPNEXI	2060
DECP	0143	FLAC	0044	INPUTX	0256	OPNEXT	1611
DELETE	2073	FLARG	0171	INSUB	0030	OPTABL	1724
DGRP	0424	FLARGP	0170	INTEGE	0052	OPTABS	2021
DGRP1	0433	FLARG2	1142	INTRPT	2610	OPTRI	2665
DMPSW	0162	FLIMIT	1076	IOBUF	3220	OPTRO	2664
DO	0417	FLINTP	6200	IRETN	0226	OPTR0	2663
DOK	2113	FLIST1	0601	I33	2516	OUT	2477
DONE	2131	FLIST2	0576	LASTLN	0160	OUTCR	2512
ECHOLS	2374	FLOP	1667	LASTOP	0140	OUTDEV	0147
EFOP	0141	FLOUTP	6000	LASTV	0165	OUTL	2157

OUTX	2511
O1	4557
O2	4562
O3	4564
O4	4444
O5	4565
O6	4566
P	ØØØØ
PACBUF	3Ø57
PACKC	4515
PACKST	Ø163
PACX	31Ø6
PC	Ø155
PCHK	Ø51Ø
PCK1	3113
PC1	Ø616
PDLXR	ØØ13
PDP	4563
PDP8I	4571
PD2	Ø535
PD3	Ø555
PEO	1265
PER	ØØ22
POPA	1413
POPF	4513
POPJ	551Ø
PRINTC	452Ø
PRNT	2433
PRNTLN	4522
PRNT2	3175
PROC	Ø613
PROCES	Ø612
PT1	Ø164
PUSHA	4511
PUSHF	4512
PUSHJ	45Ø7
P13	ØØØ5
P17	ØØ31
P177	ØØ26
P2	ØØ62
P277	ØØ32
P3	ØØ33
P337	ØØ64
P377	3Ø55
P4Ø	3Ø54
P7	4567
P76ØØ	ØØ24
P77ØØ	ØØ72
RANPT	3217
READC	4521
RECOVR	2757
RETURN	156Ø
ROT	313Ø
RTL6	4526
RUB1	3Ø1[illegible]
RUB2	3Ø44
RUB3	3Ø33
RUB4	3Ø42
SAVAC	26Ø5
SAVLK	26Ø6
SBAR	133Ø
SCONT	1316
SET	1Ø41
SEX	1373
SEXC	Ø741
SFOUND	1334
SGOT	134Ø
SIN	2723
SJ	Ø116
SORTR	1347
SORTC	4517
SORTCN	Ø137
SORTJ	4516
SPLAT	3Ø53
SPNOR	4527
SRETN	1342
SRNLST	1376
START	Ø177
STARTV	Ø144
SUBS	1533
T	ØØØØ
TASK	1216
TASK4	1257
TCRLF	1255
TCRLF2	1261
TDUMP	3135
TELSW	2662
TERMS	2ØØ1
TESTA	Ø323
TESTC	4533
TESTN	453Ø
TEXTP	Ø317
TGRP2	Ø471
THISLN	Ø156
THISOP	Ø157
TINT	2633
TINTR	1267
TLIST	1414
TLIST2	142Ø
TLIST3	14Ø4
TQUOT	1246
TR1	31Ø2
TSTGRP	4532
TSTLPR	4531
TYPE	1215
TYPE2	124Ø
T1	Ø166
T2	Ø167
UTE	23Ø2
UTO	2311
UTRA	23ØØ
UTX	2322
VAL	2467
WALL	Ø667
WORDS	ØØØ3
WRITE	Ø636
WTESTG	Ø672
WTEST2	Ø656
WX	Ø674
XABS	2Ø35
XADC	32Ø3
XCT	ØØ2Ø
XCTIN	Ø146
XDXS	1152
XDYS	1146
XENDLN	24Ø1
XF	4561
XFIND	2242
XGETLN	Ø3Ø4
XINT	1156
XI33	2667
XM	456Ø
XOUTL	2677
XPOPJ	1563
XPRNT	2421
XPUSHA	Ø477
XPUSHJ	Ø521
XRAN	32Ø5
XRT	ØØ11
XRTL6	Ø412
XRT2	ØØ12
XSGN	2Ø27
XSORTC	Ø722
XSPNOR	1533
XTESTC	Ø742
XTESTN	1547
XT3	Ø72Ø
XW	4554
XW1	4555
XW2	4556
XYZ	2442

APPENDIX F

FOCAL SYNTAX IN BACKUS NORMAL FORM

<immediate command> : : = <program statement> C.R.
<indirect command> : :=<line #> <program statement> C.R.

<line #> : : = <group no.> · <line no.> | <variable>*

<group no.> : : = 1-15 | 01-15

<line no.> : : = 01-99 | 1-9

<program statement> : : = <command> |

<command> <space> <arguments> | <command string> |

<program statement>;<program statement>

<command> : : = WRITE | DO | ERASE | GO | GOTO

<arguments> : : = ALL | <line#> | <group no.>

<command string> : : = <type statement> | <Library statement> |

<Ask statement> | <If statement>
<Modify statement> | <Set statement>
<For statement> | QUIT | RETURN | COMMENT | CONTINUE

<Set statement> : : = SET <space> <variable> = <expression>

<For statement> : : = FOR <space> <variable> = <expression>,

<expression>, <expression>; <program statement> |

FOR <space> <variable> = <expression>, <expression>;
<program statement>

<If statement> : : = IF <space> (<expression>) <line #>; |

IF <space> (<expression>) <line #>, <line #>; |

IF <space> (<expression>) <line #>, <line #>, <line #>

<Ask statement> : : = ASK <space> <Ask arguments>

<Ask arguments> : : = <operand>, <Ask arguments> |

! <Ask arguments> | #<Ask arguments> | % <format code>,<Ask arguments> |

" <character string> " <Ask arguments> | <null> |

<operand> <space> | $

<format code> : : = <line#> | <null> | <group no.>

<Library statement> : : =

LIBRARY <space> <Library Command>

<space> <file descriptions>

<Library Command> : : = CALL | SAVE | DELETE | LIST

* Not yet implemented.

<file description> : : =

DATA <space> <data list> |

FILE <space> <File name> |

FILE <space> <File name> ; <program statement> |

SYSTEM <space> <File name> |

SYSTEM <space> <File name> : <program statement> |

DATA FILES SYSTEMS

<File name> : : = <character string>

<data list> : : = <variable> | <variable>, <data list>

<Type statement> : : = TYPE <space> <Type arguments>

<Type Arguments> : : = <Ask arguments> | <expression>

<Type Arguments>,< Type arguments >

<Modify statement> : : = MODIFY <space> <line #>

This command is then followed by keyboard input characters defined as <search character>

plus

<null> | <character string> | <control character> |
<character string> <control character>

<control character> : : = <search character> |

[bell] | [form] | [line-feed] | C.R. |

[C] | ← | [rub-out]

<Variable> : : = <letter> | <letter> <character>

<Variable> <not space> <subscript>

<Subscript> : : = <left paren > <expression> <right paren>

<operand> : : = <variable> <constant> | <subscript> | <function>

<left paren> : : = < | (| [

<right paren> : : = > |) |]

<expression> : : = <unary> <operand> <operand>

<expression> <operator> <expression>

<unary> : : = + | -

<operator> : : = ↑ |* | / | + | -

<Function> : : = F <function code> <subscript>

<function code> : : = SIN | COS | LOG | ATN | EXP |
SQT | ADC | DIS | DXS | ITR |
ABS | SGN | RAN | NEW |

<character string> : : = <null> |<character> <character string>

<character> : : = a-z |<digit> | <special symbols>

<digit> : : = 1-9 | Ø

<terminator> : : = <space> | , | ; | C.R.

<not space> : : = <null> | <character>

<special symbols> : : = & | ' | : | @

<leader-trailer> : : = @ | [200] |<null>

<space> : : =

Note: spaces are ignored except when required.

APPENDIX G

NOTES: EXPLANATION OF NAGSW

G.1 NOT ALL OR GROUP SWITCH

Since LINENO may be modified, a record is needed of whether a specific line number was given by

XX.YY

Where XX and YY are nonzero or whether a group was indicated by

XX or XX. or XX.YY

Where YY = Ø

or whether "all" text was indicated by either zero, less than one, or a non-numeric argument:

	NAGSW =
For one line	4000
For a group	0000
For all text	0001

Code for testing NAGSW:

Skip if

		or	or
ONE	SNA	SNA	
ALL	SPA SNA	SPA	
GROUP	SZA		SNA SZA

G.2 EXPLANATION OF DATA INACCURACIES

"From the inequality $10^8 < 2^{27}$, we are likely to conclude that we can represent 8-digit decimal floating-point numbers accurately by 27-bit floating-point numbers. However, we need 28 significant bits to represent some 8-digit numbers accurately. In general, we can show that if $10^p < 2^{q-1}$, then q significant bits are always enough for p-digit decimal accuracy. Finally, we can define a compact 27-bit floating-point representation that will give 28 significant bits, for numbers of practical importance."[1]

[1] Goldberg, B. "8-Digit Accuracy",
Communications of the ACM
Vol. 10, No. 2, February, 1967

APPENDIX H
FUNCTIONS

H.1 THE STANDARD FUNCTIONS

The functions are provided to give extended arithmetic capabilities and the potential for expansion to additional input/output devices. There are basically three types of functions. The first group contains integer parts, sign part, square root, fractional, and absolute value functions. The second group has the input/output for scope and analog/digital converter functions. The third group has extended arithmetic computations of trigonometric and exponential functions.

A function call consists of four letters beginning with the letter F and followed by a parenthetical expression: "FSGN (A-B *2)". This expression is evaluated before transferring to the function process itself.

The function FADC( ) is used to take a reading from an analog to digital converter. The value of the function is an integer reading. Additional versions of the ADC function could be designed and incorporated in the program to provide for synchronization by a clock or other means.

*SET A= FADC ( ) *5

The scope functions FDYS (expression) and FDXS (expression) are used to set and display an X-Y coordinate on a model 34 scope and scope interface. The DXS function only sets the value of the X-coordinate to the integer part of the expression in parentheses. The DYS function sets the Y-coordinate value and intensifies the point. This makes it convenient for the programmer to set an X value and then display as many Y points along that coordinate as desired. The value returned for each of these functions is the integer part of the expression in parentheses. This expression is called the function's argument.

The extended arithmetic functions are retained at the option of the user. They consume approximately 800 characters worth of his text storage area. These arithmetic functions are adapted from the extended arithmetic functions of the three word floating point package and are described in the pertinent document.

An unorthodox distribution is provided in the basic package for a random number generator: FRAN ( ). It uses the program itself as a table of random numbers. An expanded version could incorporate the random number generator from the DECUS library.

H.1.1 Trigonometric Functions

All arguments are in radians

FSIN ( )	- the sine functions
FCOS ( )	- the cosine function
FATN ( )	- the arctangent

From these the user may compute all other trigonometric functions.

Logrithmic Functions

FLOG( )	- log to the base e or Naperian base.
FEXP ( )	- e to the power

Arithmetic Functions

FSQT ( )	- the square root
FSGN( )	- one (1) with the sign of the argument
FABS ( )	- the absolute value
FITR ( )	- the next smaller integer part maximum of 1024

LOG_{10} (ARG) = LOG_e (ARG) *LOG_{10}(e)

LOG_{10} (e) = Ø.434295

LOG_e (10) = 2.3Ø258

e = 2.71828

1 degree = .0174533 radians

1 radian = 57.2958 degrees

H.1.1.1 Using The Arctangent - An arctan function cycles between + $\pi/2$ and - $\pi/2$. Thus to get a correct range for 0-2π radians from the expression FATN(Y/X) we must use the signs of X and Y.

X	Y	FATN (Y/X)
+	+	0-PI/2
-	+	PI/2 - PI
-	-	PI - 3*PI/2
+	-	3*PI/2 - PI*2

```
13.01     IF (X) 13.1, 13.02, 13.1
13.02     SET X = 1E-200
13.1      SET THETA = FATN (FABS <Y/X> )
13.2      SET PI = 3.14159
13.3      IF (Y) 13.4; if (x) 13.5;
13.4      IF (X) 13.6; R
13.5      SET TH= PI-TH;R
13.6      SET TH= PI + TH; R
13.7      SET TH= - TH; R
```

H.2 NEW FUNCTIONS (proposed)

These functions will be available as optional patches.

H.2.1 For LAB-8

FDIS - for display

FORM: "SET Z = FDIS (X,Y)"

Where Z is a dummy variable

FUNCTION:

Setup X - Coordinate with X - value;
Setup Y - Coordinate with Y - value;
Intensify the point;
Return zero.

FADC - for analog to digital converter

FORM: "SET Z = FADC (X)"

FUNCTION:

FOR X.GE.Ø

Set Multiplexor to A/D channel number X;
Convert and return conversion value;
Disable auto-convert flip-flop.

FOR X = -1

Enable RC clock and auto-convert;
Wait for ADC done flag;
Then read converter and return value.

FOR X = -2

Enable external clock and auto-convert.

FSEL - for clock, relay, SR selection and control.

FORM: "SET Z = FSEL (x_1 x_2 x_3 x_4)

Where Y is an expression, and x; are digits

FUNCTION:

FOR Y = Ø, AND Y,
FOR x_3 EQUAL 1, 2, 4

Select clock:
x_3: 1 = RC, 2 = Crystal, 4 = external;
Return number of clock interrupts since last call;
Zero clock count

FOR x_2 EQUAL 1, 2, 4

Select relays to turn on (microprogramable):
x_2: 1 = R1, 2 = R2, 4 = R4

FOR X_1 EQUAL 1 turn all relays off.

FOR X_1 EQUAL 2 output pulse on SØ

FOR Y NOT ZERO:

The number x_1, x_2, x_3, x_4, (Octal) is masked (AND) with SR bits and results returned in decimal.

H.2.2 For Display VD 8/I (Techtronics 611)

FDIS - for display control

FORM: "SET Z = FDIS (X, Y , L)"

FUNCTION: at X and Y execute the display function L:
The X and Y are coordinate values, and
L is a letter plus arguments, if appropriate:

A	-	Absolute reference	
I	-	Incremental	
C	-	Circle (full)	
S	-	Segment,	ANGLE (in 1/16ths)
T	-	Text display,	T*E*X*T
R	-	Reset to zero	
E	-	Erase screen	
☐	-	no change	

FCUR - for cursor control

FORM: "SET Z = FCUR(X)"

FUNCTION:

Return current coordinate position. (i.e., the last position at which the button was pushed).

Range is +511 to -511.

FOR X EQUAL 1 return X-coordinate

FOR X EQUAL 0 return Y-coordinate

H.3 NEW COMMAND FOR FOCAL WITH DF32 DISK

a. Form.

"LIBRARY a b c "

Where a = operation to be done:

SAVE	(create a disk item)
CALL	(use a disk item)
DELETE	(remove a disk item)
LIST	(print names of disk items)

and b = type of file or data:

FILE	(program text)
SYSTEM	(in-progress core image)
DATA	(variables)

and c = file name or description:

four letter name for a
FILE or SYSTEM, and
a list of variables for DATA

b. Examples

```
LIBRARY CALL DATA NAME; A1, B2, C(2)...
LIBRARY CALL FILE NAME
LIBRARY CALL SYSTEM NAME
          or
L  C  S  NAME

LIBRARY DELETE DATA NAME A1, B2, C(2)...
LIBRARY DELETE FILE NAME
LIBRARY DELETE SYSTEM NAME
          or
L  D  S  NAME
```

For a FILE or a SYSTEM in a SAVE command, the command string, if any, that follows the semicolon is placed in the command buffer to be executed as a direct command when the program has been loaded via a CALL.

```
LIBRARY SAVE DATA NAME; AL(I + 1)...
LIBRARY SAVE FILE NAME; GOTO 3.4
LIBRARY SAVE SYSTEM NAME;
```

To list all files of type n

```
LIBRARY LIST DATA
LIBRARY LIST FILES
LIBRARY LIST SYSTEMS
          or
L  L  S
```

Only LIBRARY SAVE n may be followed by ;.

c. Elucidation

These command features will permit optimum usage of available disk storage. It will be compatible with the disk "Monitor".

When a new program is called, the old one in core is erased and control is transferred to the command buffer, thereby automatically starting the program, if desired. Thus programs may link together and branch out in complex sequences.

Common variables may be referenced by establishing common names. Those variables saved by the LIBRARY command are stored in a FOCAL Scratch Area and may not be referenced by PIP. FILES and SYSTEMS are saved as .USER files.

A program may also save itself by some conventional name such as MAIN; GO before calling another program. That program could then return control to the original routine with LIBRARY CALL SYSTEM MAIN. Thus, programs may be used as subroutines.

d. Raw Date

FILE (1)

144 = BUFR, R
165 = LASTV, X/changed to R
2522 = COMEIN
↓
2577 = COMOUT-6
3240 = FRST
↓
R

SYSTEM (2)

Page ZERO
2522 = COMEIN
↓
2577
3200 = FRST - 40
↓
X

DATA (3)

N	A
SUBSCRIPT	
S EXPONENT	
S MANTISSA (11)	
MANTISSA (12)	

e. Loading Procedure

(1) Load and build the Disk Monitor.
(2) Add FOCAL to the system. Add LIBRARY to the system.
(3) Load FOCAL DISK SYSTEM tape
(SA = 200)
(4) Start it.
(5) The following files will be created.

FOCA.USER	MAIN + DIALOG
.FL.X	Floating Point Section
.LT.X	Library Command Section
FCON.SYS	Latest Program
REEN.SYS	Reentrant Program
.VR.X	Variables

(6) It will then commence the initial dialog.

f. Control - C

When FOCAL (disk version) is given a control - C it will save itself as FCON.SYS and return control to the Disk Monitor. He could then resume where he left off by typing

.FCONT ↵

and the program will continue. If he wishes to restart FOCAL, retain his FOCAL text and to go into command-input mode, he may type

.REENTER ↵
*

g. Currently available, on an experimental basis only, are an 8K version of FOCAL, a two user system, a patch to utilize the CalComp plotter, and a patch to utilize the high speed reader.

The latter is implemented as a command: **; or ** (return). When the * command is executed the interrupts are disabled, echo is disabled, and <u>all</u> input is taken from the high speed reader. This input may be commands, program text, or data. All output is presented to the high speed punch.

An * command on the tape will cause all interrupts, echos, and input device pointers to be restored. Out of tape condition will generate the same result. A user without a high speed reader will, therefore, not get into trouble by using the * command. This also means that several programs may be linked together via the reader.

A user without a high speed punch will get hungup!

APPENDIX I

PROGRAM LISTS

```
/ NEW INSTRUCTIONS:

PUSHJ = JMS  I .        /RECURSIVE SUBROUTINE CALL
          XPUSHJ
POPA = TAD  I POLXR /RESTORE AC
POPJ = JMP   I .        /SUBROUTINE RETURN
          XPUPJ
PUSHA = JMS I .         /SAVE AC
          XPUSHA
PUSHF = JMS  I  .       /SAVE GROUP OF DATA
          PD2
POPF = JMS   I  .       /RESTORE GROUP
          PD3
GETC = JMS  I  .        /UNPACK A CHARACTER
          UTRA
PACKC  = JMS I  .       /PACK A CHARACTER
          PACBUF
SORTJ = JMS   I .       /SORT AND BRANCH ON AC OR CHAR
SJ,       SORTB
/ NUMERICAL LIST - 1
/ ADDRESS LIST - NUMERICAL LIST
SORTC = JMS  I   .      /SORT CHAR
          XSORTC
PRINTC = JMS  I   .     /PRINT AC OR CHAR
          OUT
READC = JMS   I  .      /READ ASR - 33 INTO CHAR AND PRINT IT
          CHIN
PRNTLN = JMS I .        /PRINT C (LINENO)
          XPRNT
GETLN = JMS   I   .     /UNPACK AND FORM A LINENUMBER
          XGETLN
FINDLN = JMS  I   .     /SEARCH FOR A GIVEN LINE
          XFIND
ENDLN = JMS   I  L      /INSERT LINE POINTERS
          XENDLN
RTL6 = JMS       I  .   /ROTATE LEFT SIX
          XRTL6
SPNOR = JMS  I   .      /IGNORE SPACE AND LEADING ZEROS
          XSPNOR
TESTN = JMS   I   .     /PERIOD:  OTHER:  NUMBER
          XTESTN
TSTLPR = JMS   I   .    /SKIP IS 5 < SORTCN < 11 (I.E. AN L-PAR)
          LPRTST
TSTGRP = JMS  I   .     /SKIP IF G(AC) = G (LINENO)
          GRPTST
TESTC = JMS    I  .     /TERM; NUMBER; FUNCTION; LETTER
          XTESTC
```

```
ERROR2 = JMS  I  .     /EXCESS SOMETHING ERROR
         ERR2
ERROR3 = JMS  I  .     /MISCELLANEOUS ERROR
         ERR3
ERROR4 = JMS  I  .     /FORMAT ERROR
         ERR4
```

digital

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS