

Digital Equipment Corporation  
Maynard, Massachusetts

**digital**

# ADVANCED FOCAL

TECHNICAL SPECIFICATIONS

# **ADVANCED FOCAL TECHNICAL SPECIFICATIONS**

For additional copies order No. DEC-08-AJBB-DL from Program Library, Digital Equipment Corporation, Maynard, Mass.

Price: \$5.00

Copyright © 1969 by Digital Equipment Corporation

The following are registered trademarks of Digital  
Equipment Corporation, Maynard, Massachusetts:

DEC  
FLIP CHIP  
DIGITAL

PDP  
FOCAL  
COMPUTER LAB

## CONTENTS

Page

### CHAPTER 1 INTRODUCTION

### CHAPTER 2 COMMANDS

2.1	Type, Ask	2-1
2.1.1	Literals	2-1
2.1.2	Numerical Input Formats	2-1
2.1.3	Alphanumeric Input Formats	2-1
2.1.4	Special Characters	2-2
2.1.5	Print Positions	2-2
2.1.6	Symbol Table	2-2
2.1.7	Output Formats	2-3
2.1.8	Terminators	2-3
2.1.9	Off-Line Data Tapes (c.f., Section 4.5.3)	2-3
2.1.10	Corrections	2-3
2.1.11	Roundoff	2-3
2.2	DO	2-4
2.3	Editing and Text Manipulation Facilities	2-4
2.3.1	Command-Input	2-4
2.3.2	ERASE	2-5
2.3.3	MODIFY	2-5
2.4	FOR	2-6
2.5	IF	2-6
2.6	GOTO	2-7
2.7	RETURN	2-7
2.8	QUIT	2-7
2.9	COMMENT	2-7
2.10	CONTINUE	2-7
2.11	SET	2-7
2.12	High-Speed Reader	2-8
2.12.1	General	2-8
2.12.2	Other Rules	2-8
2.13	The Functions	2-10

## CONTENTS (Cont)

		Page
2.13.1	General	2-10
2.13.2	Analog to Digital	2-10
2.13.3	Extended Functions	2-11
2.13.4	Random Numbers	2-11
2.13.5	Standard Functions	2-11
2.13.6	Using the Arctangent	2-12
2.13.7	Boolean Functions	2-13
2.13.8	FNEW - A User Functions	2-13
2.14	The Library Command	2-13
2.14.1	L-Command For Single User System	2-13
2.14.2	LIBRA Command Specifications for Multi-User Systems	2-14
2.14.3	DF32 FOCAL FILE STRUCTURE	2-15
2.15	Write	2-16

## CHAPTER 3 FOCAL USAGE

3.1	Requirements	3-1
3.2	Loading Procedure	3-1
3.3	Initial Dialogue	3-1
3.4	Operation	3-1
3.4.1	Restart Procedure	3-1
3.4.2	Keyboard Error Recovery	3-2
3.4.3	Parentheses	3-2
3.4.4	Trace Feature	3-2
3.4.5	Variables, Functions and Numbers	3-2
3.4.6	Error Diagnostics	3-2
3.4.7	Arithmetic Priorities	3-3
3.4.8	ASCII data	3-3
3.4.9	Indirect Commands	3-3
3.5	Saving Focal Programs	3-4
3.5.1	Paper Tape	3-4
3.5.2	LINC Tape	3-4
3.5.3	Disk Monitor System	3-4
3.5.4	Disk System and Extended Functions	3-5

## CONTENTS (Cont)

		Page
3.5.5	Disk System and Extended Memory	3-6
3.5.6	For 4-user FOCAL SAVE command, see Section 4.6.6	3-7
3.5.7	EAE Patch for FOCAL, 1969	3-7

## CHAPTER 4 PROGRAM SPECIFICATIONS

4.1	Machine Requirements	4-1
4.2	Design Specifications	4-1
4.2.1	Design Goals	4-1
4.2.2	Input	4-1
4.2.2.1	Input Format	4-1
4.2.2.2	Character Set	4-1
4.2.3	Output	4-2
4.2.3.1	Output Format	4-2
4.2.3.2	The Input/Output and Interrupt Processor	4-2
4.2.4	Organization	4-3
4.2.4.1	Arithmetic Package	4-3
4.2.4.2	Storage	4-3
4.3	Hardware Errors	4-4
4.4	Internal Environment	4-4
4.4.1	Adding a User's Function; FNEW(Z)	4-4
4.4.2	Internal Subroutine Conventions	4-6
4.4.2.1	Calling Sequences	4-6
4.4.2.2	Subroutine Organization	4-7
4.4.3	Character Sorting	4-7
4.4.4	Language	4-8
4.5	Notes	4-9
4.5.1	Core Utilization	4-9
4.5.2	Extended Functions	4-9
4.5.3	Error Printouts	4-10
4.5.4	No Interrupts	4-10
4.5.5	Operating HS Reader Without Interrupts	4-10
4.5.6	Non-Typing of Program Tapes During Loading	4-10
4.5.7	Explanation of NAGSW (Not All or Group Switch)	4-10

## CONTENTS (Cont)

	Page
4.5.8 Data Inaccuracies	4-11
4.5.9	4-11
4.5.10 Estimating the Length of User's Program	4-11
4.6 FOCAL Systems	4-12
4.6.1 FOCAL Systems Assembly	4-14
4.6.2 FOCAL Binary Paper Tapes	4-15
4.6.3 FOCAL Listings	4-15
4.7 FOCAL Segments	4-15
4.7.1 8K Single User Overlay - 8K	4-15
4.7.2 Extended Precision Overlay - 4Word	4-15
4.7.2.1 Double Precision Multiply in Four-Word FOCAL	4-16
4.7.3 Four User Overlay - QUAD	4-16
4.7.3.1 Four User Loading and Operating Procedure	4-16
4.7.3.2 Swapping	4-17
4.7.3.3 Workload and Timing	4-17
4.7.3.4 Special Controls	4-17
4.7.3.5 Dialogue	4-17
4.7.4 Graphics for Circles and Lines - CLIN	4-18
4.8 FOCAL Demonstrations	4-22
4.8.1 One-Line Function Plotting	4-22
4.8.2 How to Demonstrate FOCAL's Power Quickly	4-23
4.9 FOCAL Versus BASIC	4-23

## CHAPTER 5 ADDITIONAL FOCAL APPLICATIONS

5.1 FOCAL for the LAB-8	5-1
5.1.1 Standard	5-1
5.1.2 Additions! (Possible) FOCAL Functions for AX-08	5-1
5.2 FNEW for Data Arrays	5-3
5.2.1 Storage Requirements	5-3
5.2.2 Usage	5-3
5.2.2.1 Loading	5-3
5.2.2.2 Calling Sequence	5-4
5.2.3 Recursive Calling	5-4

## CONTENTS (Cont)

	Page
5.2.4            Restrictions	5-4
5.2.5            Description	5-4
5.3              Dynamic Interrupt Processing via FOCAL, 1969	5-5
5.4              Simultaneous Equations' Solutions	5-6
5.5              Fast Fourier Transforms Programs	5-6
5.6              Travel Voucher to Expense Voucher Conversion Program	5-8
5.7              Twins Demo	5-10

### APPENDIX A FOCAL COMMAND SUMMARY

### APPENDIX B ERROR DIAGNOSTICS

### APPENDIX C EXPLANATION OF NEW INSTRUCTIONS

### APPENDIX D FOCAL CORE LAYOUT

### APPENDIX E SYMBOL TABLE AND OTHER TABLES/LISTS

### APPENDIX F FOCAL SYNTAX

### APPENDIX G ILLUSTRATIONS

### ILLUSTRATIONS

4-1              Figure 4-1	4-8
D-1              FOCAL Core Layout Dynamic Storage	D-4
G-1 (Sheet 1) Arithmetic Evaluation	G-1
G-1 (Sheet 2) Arithmetic Evaluation	G-2
G-1 (Sheet 3) Arithmetic Evaluation	G-3
G-1 (Sheet 4) Arithmetic Evaluation (Analysis of Functions)	G-4
G-2              Command/Input	G-5
G-3              Main Control and Transfer	G-6
G-4              DO Command	G-7
G-5 (Sheet 1) Input/Output Commands	G-8
G-5 (Sheet 2) Input/Output Commands	G-9



## ILLUSTRATIONS (Cont)

	Page
G-6 Iteration Control	G-10
G-7 Conditional Branch Command	G-11
G-8 Character Editing	G-12
G-9 (Sheet 1) ERASE and Delete	G-13
G-9 (Sheet 2) ERASE and Delete	G-14
G-10 (Sheet 1) Interrupt Handler	G-15
G-10 (Sheet 2) Interrupt Handler	G-16
G-11 Variable Look-up and Enter	G-17
G-12 Character Unpacking	G-18
G-13 "FINDLN" Routine	G-19

## TABLES

4-1 FOCAL Source Segments	4-13
4-2 Allowable FOCAL Systems	4-13
4-3 Variations for FOCAL Systems	4-14
B-1 Error Diagnostics of FOCAL, 1969	B-1
C-1 New Instructions	C-3
D-1 FOCAL Core Layout Usage	D-1
D-2 Detailed FOCAL Core Layout	D-2
F-1 Syntax in Backus Normal Form	F-1
F-2 FOCAL Commands in French	F-2

## CHAPTER 1

### INTRODUCTION

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

The FOCAL<sup>T.M.</sup> language is used as a tool in a conversational mode; that is, the user creates his problem step by step, while sitting at the computer; when 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 in standard mathematical notation.

FOCAL 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 the replacement or repositioning of lines. 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) or indirect (to be stored and executed later) commands. An example of a direct command is

```
*TYPE  5*5*5,1  (User)
      = 125.000* (PDP-8)
```

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

---

<sup>†</sup>Formulating On-Line Calculations in Algebraic Language (or FORMula CALculator)

<sup>T.M.</sup> Trademark of the Digital Equipment Corporation, Maynard, Mass.

Text input requires that a numerical digit, in the form ab.cd and within a range of 1.01 to 31.99, 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 can be modified by a small binary tape to understand foreign languages commands. (See Appendix F-2)

FOCAL is written especially for the educational and engineering markets and is intended to be used as a problem solving tool. It gives quick and concise reinforcement, minimizes turnaround time, and provides an unambiguous printed record.

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

This document describes the language, operating procedures for Disk Monitor and FOCAL; use of High Speed reader; addition of user function FNEW; and many other details of interest. Symbol tables, lists, and flow-charts are included.

There are also descriptions of the 10-digit overlay, 4 user overlay, and the complete graphics function.

## CHAPTER 2

### COMMANDS

#### 2.1 TYPE, ASK

The TYPE and the ASK statements are used for output and input of literals, alphanumeric calculations, and formats. The simplest form of the TYPE statement is a command (e.g., 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 to the TYPE statement in form, but only single variable names can be used instead of expressions, and the user types in the values.

##### 2.1.1 Literals

For output of literals, the user may enclose characters in quotation marks. The carriage return will automatically generate closing quotation marks. The bell may only be inserted during initial input, not via the MODIFY command.

##### 2.1.2 Numerical Input Formats

Keyboard responses to ASK inputs may

- a. have leading spaces
- b. be 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. It is recommended, however, 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, to insert special paper in the teletype before executing the program, type Ask A; GO and RETURN, then load the paper, and hit ALTMODE. The value of the variable used remains unchanged.

##### 2.1.3 Alphanumeric Input Formats

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 (see 3.4.9). This is done by a simple hash-coding technique so that the program can recognize keyboard responses by a single comparison. See example under the IF command for an illustration of how to program the

recognition of the user reply "WAIT". This is possible because the leading zero causes a character string to be interpreted as a number. (e.g.,

\*TYPE 0ANSWER = 0.26130E+22\*).

Any literal word containing the letter "E" twice in one input will cause the ASK statement to be terminated as the program interprets this letter as an exponent.

#### 2.1.4 Special Characters

The exclamation point (!), percent (%), dollar sign (\$), and the number sign (#) may be used next to 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.

To type an expression before its results, the user may enclose the expression in question marks. This is a special use of the trace feature.

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

#### 2.1.5 Print Positions

Carriage returns are not automatically supplied at the termination of a typeout. 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 may be used to plot another variable along the same coordinate.

#### 2.1.6 Symbol Table

TYPE.\$ (dollar sign) causes the contents of the symbol table to be typed out with the current values of all variables created. 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 a semicolon (;) and other commands, unless a \$ is in the string.

### 2.1.7 Output Formats

The output format may be changed within a TYPE statement by %X.YY, where X and YY are positive integers less than 31. 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.

During output, leading zeroes are typed as spaces. If the number is larger than the field width indicates, FOCAL will convert to E format. E format is also specified by % alone. (Floating-point decimal:  $\pm 0.XXXXXXXXXE\pm Y$ , where E means "10 to the Yth power".) The current output format is retained until explicitly changed. If a number is too large for the current format, the E format is used temporarily.

### 2.1.8 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 the TYPE or ASK statement, command arguments may be preceded by format control characters # ! ". Example:

```
*ASK ?A B C ?  
A :5, B :6 C :7) *
```

All commands except WRITE, RETURN, MODIFY, QUIT and ERASE may be combined on the same line if separated by a semicolon.

### 2.1.9 Off-Line Data Tapes (c.f., Section 4.5.3)

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.

### 2.1.10 Corrections

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

### 2.1.11 Roundoff

Numbers to be typed out are rounded-off 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.

## 2.2 DO

The DO command is used chiefly to form subroutines from single lines, groups of lines, or from 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 (e.g., 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 an 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 the 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 ALL.

The DO statement may be concatenated with other legitimate commands by terminating it with a semicolon. Thus, a single line may 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 if any of the DO command line remains, it will be processed.

## 2.3 EDITING AND TEXT MANIPULATION FACILITIES

### 2.3.1 Command-Input

A line number which has already been used and is reused 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.

### 2.3.2 ERASE

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.

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.

To examine a single line, type WRITE followed by the line number. For example, WRITE 3.3 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. WRITE ALL will cause all of the text to be printed on the Teletype, left justified, with title and line numbers in numerical order.

### 2.3.3 MODIFY

When only a few characters of a particular line must be replaced, the MODIFY command is used to avoid replacing the entire line. For example, to change characters in line 5.41, type MODIFY 5.41. This command is terminated by a carriage return, and the program waits for the user to type that character at which he wishes to make changes or additions. The program will then 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:

- a. type in new characters in addition to the ones that have already been typed out;
- b. 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, as these commands 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 typing 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.



## NOTE

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 the data structure. It is justified by the principle that a change of program probably means a change of variables as well.

### 2.4 FOR

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 at least once. 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.

Naturally, A must be a single variable; but 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. For example:

SET B = 3; FOR I = 0, 10; TYPE B ↑ I, ! (power of 3)

### 2.5 IF

To provide 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, followed by a 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.

Example:

ASK REPLY  
IF (REPLY - 0WAIT) 6.4, 5.01; RETURN  
IF (REPLY - 0YES) 6.3, 5.02; 6.3

## NOTE

The IF command could occasionally fail to take the = 0 branch due to internal computation and truncation errors.

### 2.6 GOTO

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.

### 2.7 RETURN

The RETURN command is used to exit from DO subroutines. It is implemented internally by 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.).

### 2.8 QUIT

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

### 2.9 COMMENT

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

### 2.10 CONTINUE

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.

### 2.11 SET

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:  $+ - / * \uparrow$ . 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. For example:

```
SET AA=B(5+<6+CONST>*ALPHA/[5/BETA]);GOTO 3.2
```

## 2.12 HIGH-SPEED READER

### 2.12.1 General

The asterisk (\*) is also used as a flip-flop control over the selection of the input device to be used by a FOCAL program. (See the examples that follow.) An out-of-tape condition will return to low-speed reader input and change the status of the \* flip-flop. An error condition, however, does not change that \* flip-flop (see notes below).

For example, typing:

```
**  
_ )
```

will read in a program tape or a series of immediate commands.

```
** ;ASK ABCDZ  
_
```

will fill AB with data from tape. If tape is empty, control will return to command mode.

```
*1.1* ; FOR I=1, 5; ASK AX(I)  
*DO 1.1
```

If the tape contains fewer than 5 pieces of data, then remaining items are taken from keyboard. (See c below.)

### 2.12.2 Other Rules

a. \* as a command may be concatenated with other processes [JMP (PROC):

(e.g., 01.30\* ; ASK A, B;\*)

b. If an out-of-tape condition is encountered while reading commands, then the input device is switched to keyboard and all is returned to normal. (This occurs when the user has no reader.) It is equivalent to receipt of a left arrow. [JMP (IBAR)].

c. If an out-of-tape condition occurs while executing an ASK command, then FOCAL responds as if the end of the command line (carriage return) has been reached. [ISZ PDLXR; POPJ]

Thus,

```
** ; ASK A,B,C,D  
_
```

produces:::(out of tape on C): and the user is back to normal mode.

However,

```
*ERASE
**; for I=1, 20; ASK A(I); TYPE I,!.
_
: = 1.0000
: = 2.0000
: = 3.0000
: (out of tape for I=4)
: (now accepting from keyboard) 123, = 5.0000
: 345, = 6.0000
: ?01.00 (Control-C typed)
* TYPE $
I@ (00) = 7.0000
A @ (01) = (data from tape)
A @ (02) = (data from tape)
A @ (03) = (data from tape)
A @ (04) = .0000
A @ (05) = 123.0000
* A @ (06) = 345.000
```

d. When an error occurs from the reader (illegal command, etc.), the code will be typed out and input device control returned to the low-speed device. However, the device flip-flop (HSPSW) will still indicate that the reader is active. Consequently, it will be necessary to give two asterisks before the reader will be activated again.

```
**
*****?12.83 (Buffer full)
**
**
(reader now active again).
```

e. It is necessary to have a fairly long timing loop to detect the out-of-tape condition (slow readers, restart delays, etc.). As a result, the user of a PDP-8/S may encounter long delays if there is no high speed reader or when the reader is out of tape. However, the initial dialogue makes a correction for this when an 8/S is being used.

f. Since the reader operates with the interrupt on, one may use Control-C to return at once to keyboard input mode. A manual interrupt via Control-C (?01.00) or a console restart (?00.00) gives the same effect.

g. All commands, including "\*" may be executed in immediate mode from the high speed reader. This has several beneficial results:

(1) Program tapes may be composed that are self-protecting and self-starting

```
ERASE ALL (protection)
01.10 ASK "Power of 2?"REPLY (input indirect program)
01.30 TYPE 2 REPLY,!,GOTO 1.1
(etc)
GOTO 1.1 (starting)
5, 3, 1 (data)
```

This particular program is an infinite loop and must be stopped by a Control-C from the keyboard.

- (2) Programs may chain themselves together.

ERASE ALL

3.4 TYPE "NUMBER 1"!!!; ASK A

3.5 \* (indirect command)

\*; GO (device restored to low speed and program started)

The printout from this tape will be:

\*\* (START READER)

\*\*\*\*\* NUMBER 1

(Three lines accepted)

(Erase processed)

: (waiting for keyboard input) ↵ (user)

(execution of 3.5 \* at this point will reactivate the high speed reader).

- (3) Immediate mode commands on the tape allow maximum storage for variables.

- (4) If the interrupts are disabled by the patches shown in Section 4.5.3, then two tapes may be merged from both high- and low-speed readers by a resident FOCAL program.

## 2.13 THE FUNCTIONS

### 2.13.1 General

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 no more than four letters beginning with the letter F and followed by a parenthetical expression (e.g., FSGN (A-B \*2)). This expression is evaluated before transferring to the function process itself.

### 2.13.2 Analog to Digital

#### a. Input

The function FADC(X) is used to take a reading from an analog-to-digital converter. The value of the function is a 12-bit integer reading. The argument "X" is the channel member (AX08) in decimal. Additional version of the ADC function could be designed to provide for synchronization by a clock or other means. (c.f., Chapter 5)

\*SET A=FADC ( ) \*5

#### b. Output

The scope function FDIS (expression, expression) is used to set and display an X-Y coordinate on a Model 34 Scope and scope interface. The value returned for each of these functions is the integer part of the second expression.

\*SET Z = FDIS(X,X43/50)

#### 2.13.3 Extended Functions

The extended arithmetic functions (FEXP, FLOG, FATN, FCOS, FSIN) are retained at the option of the user. They consume approximately 800 characters of text storage area. These arithmetic functions are adapted from the extended arithmetic functions of the three-word, floating point package.

#### 2.13.4 Random Numbers

A simple random number generator is provided in the basic package as FRAN(!). An expanded version could incorporate the random number generator from the DECUS library.

Functions for other devices are provided as overlay tapes (see Appendix H).

#### 2.13.5 Standard Functions

##### a. Trigonometric Functions

All arguments are in radians

FSIN ( ) - the sine functions

FCOS ( ) - the cosine function

FATN ( ) - the arctangent

From these functions, the user may compute all other trigonometric functions. (See FOCAL User's Manual)

##### b. Logarithmic Functions

FLOG ( ) - log to the base e or Napierian base

FEXP ( ) - e to the power

##### c. 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

$\text{LOG}_{10}(\text{ARG}) = \text{LOG}_e(\text{ARG}) * \text{LOG}_{10}(e)$

$\text{LOG}_{10}(e) = 0.434295$

$\text{LOG}_e(10) = 2.30258$

$e = 2.71828$

where:

1 degree = .0174533 radians

1 radian = 57.2958 degrees

## 2.13.6 Using The Arctangent

An arctan function cycles between  $+\pi/2$  and  $-\pi/2$ . Thus, to get a correct range for  $0-2\pi$  radians from the expression  $\text{FATN}(Y/X)$ , we must use the signs of X and Y.

<u>X</u>	<u>Y</u>	<u>FATN(X/Y)</u>
+	+	$0-\pi/2$
-	+	$\pi/2 - \pi$
-	-	$\pi - 3\pi/2$
+	-	$3\pi/2 - \pi$

\*60

INDEX	X	Y	FUNCTION	COMPUTED
= 0.00=	1.00=	0.00=	0.000000	= 0.000000
= 0.30=	0.96=	0.30=	0.300000	= 0.300000
= 0.60=	0.83=	0.57=	0.600000	= 0.600000
= 0.90=	0.62=	0.78=	0.900000	= 0.900000
= 1.20=	0.36=	0.93=	1.200000	= 1.200000
= 1.50=	0.07=	1.00=	1.500000	= 1.500000
= 1.80=-	0.23=	0.97=-	1.341600	= 1.800000
= 2.10=-	0.51=	0.86=-	1.041600	= 2.100000
= 2.40=-	0.74=	0.68=-	0.741595	= 2.400000
= 2.70=-	0.91=	0.43=-	0.441595	= 2.700000
= 3.00=-	0.99=	0.14=-	0.141595	= 3.000000
= 3.30=-	0.99=-	0.16=	0.158403	= 3.300000
= 3.60=-	0.90=-	0.44=	0.458402	= 3.600000
= 3.90=-	0.73=-	0.69=	0.758402	= 3.900000
= 4.20=-	0.49=-	0.87=	1.058400	= 4.200000
= 4.50=-	0.21=-	0.98=	1.358400	= 4.500000
= 4.80=	0.09=-	1.00=-	1.483200	= 4.800000
= 5.10=	0.38=-	0.93=-	1.183200	= 5.100000
= 5.40=	0.64=-	0.77=-	0.883196	= 5.400000
= 5.70=	0.84=-	0.55=-	0.583195	= 5.700000
= 6.00=	0.96=-	0.28=-	0.283198	= 6.000000
= 6.30=	1.00=	0.02=	0.016802	= 0.016802
= 6.60=	0.95=	0.31=	0.316803	= 0.316803
= 6.90=	0.82=	0.58=	0.616800	= 0.616800

C-FOCAL , 8/68

```

01.05 T !!!!!" INDEX X Y FUNCTION COMPUTED
01.10 FOR I=0,.3,7: TYPE !,Z4.02,I;D 2
01.20 TYPE !!!!!;WRITE ALL
01.30 QUIT

02.10 SET Y=FSIN(I); SET X=FCOS(I)
02.20 TYPE X,Y,Z8.06,FATN(Y/<X+1E-10>); DO 13: TYPE " " TH;

13.10 IF (X)13.3,13.2,13.3
13.20 SET X=1E-100
13.30 SET TH=FATN(FABS<Y/X>)
13.40 SET PI=3.141596
13.50 IF (Y) 13.6; IF (X) 13.7; RETURN
13.60 IF (X) 13.8;SET TH=PI+PI-TH; RETURN
13.70 SET TH=PI-TH; RETURN
13.80 SET TH=PI+TH; RETURN
*
```

### 2.13.7 Boolean Functions

TRUE is +1  
FALSE is -1

```
*D 15
  A B  AND  OR  NOR  XOR  CARRY  SUM
--1=-1  --1= -1   1=   1   --1=-1
--1= 1   --1=   1   1= -1   --1= 1
= 1=-1   --1=   1   1= -1   --1= 1
= 1= 1   = 1     1   -1=   1   = 1=-1
```

XOR is  $A*B$   
NOR is  $FSGN(-A-B)$   
OR is  $FSGN(A+B)$   
AND is  $FSGN(A+B-1)$   
NOT(A) is  $-A$

The result of adding A and B is

CARRY =  $FSGN(A+B-1)$   
SUM =  $-A*B$

\*

```
*WRITE 15
15.05 TYPE " A B  AND OR NOR XOR CARRY  SUM"!
15.10 FOR A=-1,2,1; FOR B=-1,2,1;TYPE A,B, "    "; DO 15.2
15.15 QUIT
15.20 TYPE FSGN(A+B-1),FSGN(A+B),FSGN(-A-B),A*B, "    "FSGN(A+B-1),-A*B,!
*
```

### 2.13.8 FNEW - A User Function

This function name may be used to call a machine language routine for any reason.

(See Section 4.4.1)

## 2.14 THE LIBRARY COMMAND

The form and usage of this mass storage command will vary with the computer and FOCAL system used. (c.f., 4.6)

### 2.14.1 L-Command For Single User System

The command may be given in either direct or indirect mode. Execution of this command first causes the octal typeout of the contents of four FOCAL pointers: CFRS, BUFR, LASTV, and BOTTOM, respectively. The second action is to type out whatever characters follow the "L" to serve as operating instructions for the user. The third action is to turn off the interrupts and transfer to the Disk Monitor or 8-Library System by jumping to 7600.



The four octal numbers represent:

- a. the start of text buffer,
- b. the end of text buffer,
- c. the end of the variable list,
- d. the bottom of the push-down list.

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

After debugging a program, a typical user will execute ERASE and LIB. (This causes B and C to be equal in the 4K system.) He will then save the program and restart or call another program. (See Section 3.4.12)

Manual Chaining may also be done. For example, when a program reaches line 12.3, it may need to call another routine (as in a series of teaching programs, demos, or math subroutines). The user, however, must be given instructions on how to proceed:

```
12.30LIB .CALL LES2
```

For example, execution of 12.3 may produce:

```
3206
3345
3401
4407
.CALL LES2
.CALL LES2           [User types this]
.START
*
```

In the 8K Version, the text and variables are stored independently. For this reason, the 8K version can have different programs operating on the same data. (See Section 3.4.14)

#### 2.14.2 LIBRA Command Specifications for Multi-User Systems\*

Four modifiers of the LIBRARY command are implemented to allow automatic program storage, retrieval, and management in multi-user FOCAL. This extension to the FOCAL system is implemented under the segment name LIBRA and requires at least an 8K PDP-8 with one DF32.

The LIBRARY command and its variations are:

- a. To save a program on disk,

```
LIBRA SAVE name )
```

Where "name" is a 1 to 4 character identifier and ) is described in the FOCAL language specifications.

\*Not completed

Errors:

- (1) A program with an identical name has been found in the directory list
- (2) Name missing from command
- (3) Disk I/O error (non-recoverable)

b. To call a program on disk,

LIBRA CALL name ↵

Errors:

- (1) No such program on directory list
- (2) Name missing from command
- (3) Disk I/O error (non-recoverable)

c. To delete a program from disk,

LIBRA DELETE name ↵

Errors:

- (1) No such program name in directory list
- (2) Name missing from command
- (3) Disk I/O error

d. To list the directory

LIBRA LIST ↵

Errors:

- (1) Disk I/O error

#### NOTE

This command will destroy any program by an effective "ERASE ALL".

The directory is printed ten across for as many lines as necessary.

### 2.14.3 DF32 FOCAL FILE STRUCTURE

Programs are stored in blocks 1600<sub>8</sub> words long. This allows 36 blocks of storage on one DF32 and a directory of 512 words or 256 entries. This directory is sufficient for the maximum DF32 configuration allowable on a PDP-8.

1. Disk 36 blocks
2. Disk 72 blocks
3. Disk 110 blocks
4. Disk 146 blocks

The directory is a linear list with a maximum size of 512 words (with 2 words/entry). Word position in the list corresponds to the block position on the disk. The blocks begin at location 1000<sub>8</sub> from the end of the directory and extend in increments of 1600<sub>8</sub> to the end of the disk. The end of the list is an entry of ones. Unused blocks are indicated by entries of all zeroes.

The LIBRARY functions swap users in the multiple user system. This diminishes the total number of blocks by the maximum number of allowed users. A disk program is required to clear the directory, and to set the maximum number of blocks available.

## 2.15 WRITE

The WRITE command is used to list the entire indirect program (WRITE ALL or W), specified groups, or single lines. When all text is printed, a leader-identifier is given at the top of the listing. This identifies which major version is being used for the particular indirect program. (FOCAL, 1969; 8K FOCAL @ 1969; 4-word @ 1969).

### NOTE

The WRITE command disables the trace.

## CHAPTER 3

### FOCAL USAGE

#### 3.1 REQUIREMENTS

Any 4K PDP-8 family computer with Teletype may be used with FOCAL: PDP-5, PDP-8, PDP-8/S, PDP-8/I, PDP-8/L, LAB-8, LINC-8, TSS-8, PDP-12.

#### 3.2 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 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 two sections.
- 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.

#### 3.3 INITIAL DIALOGUE

The program will identify the DEC 12-bit computer you are using and make appropriate corrections to itself. If the user determines that extra space is required, the program will permit rejection of extended functions.

FOCAL is ready for commands when it types \*.

#### 3.4 OPERATION

##### 3.4.1 Restart Procedure

There are two methods to restart 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 STOP key
- c. Press the LOAD ADDRESS key
- d. Press the START key
- e. The program will then type ?00.00 indicating a manual restart, and an asterisk indicating it is ready to receive input.

### 3.4.2 Keyboard Error Recovery

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

- a. Use the RUBOUT key on the teletype keyboard to erase the preceding character. The RUBOUT key echoes \ for each character removed.
- 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.
- c. Use Left Arrow to delete over to the left margin.
- d. Use Left Arrow to delete input data.

### 3.4.3 Parentheses

The following parenthetical pairs may be used in any alphanumeric expression: parentheses, angle brackets ( < > ), and square brackets ( [ ] ). The program checks to see whether the proper matching terminator has been used at the correct level. Use of these terminators in different configurations provides additional clarity in reading alphanumeric expressions.

### 3.4.4 Trace Feature

A trace feature may be used to detect errors, follow program control, and create special formats. To implement the trace feature, insert a question mark into a command string at any point. Each succeeding character will then be typed out as it is interpreted until another question mark is encountered or until the program returns to command-input mode.

### 3.4.5 Variables, Functions and Numbers

A variable name consists of one or two alphanumeric characters, of which the first must be a letter. The second character may be A-Z, 0-9, ", '. Additional characters are ignored.

Function names are easily distinguished from variable names because they start with the letter F. A number always begins with a digit 0-9.

### 3.4.6 Error Diagnostics

Programming errors are indicated by an error diagnostic. The printout is in the form ?XX.XX @ GG.SS. The first number is a specific error number derived from the core address of the error call. The GG.SS is the number of the line, if any, of the text which contains the error.

The error diagnostic printouts are intended to be efficient yet informative and explicit. Used in conjunction with the trace feature, these will pinpoint errors precisely. (See Appendix B).

Example:

```
*DO 2.35?  
SET A=5/C + ?28.72 (Divide by zero, C=0)  
*
```

#### 3.4.7 Arithmetic Priorities

↑  
\*  
/  
+-

Operations of equal priority are executed from left to right (e.g.,  $T 2 \uparrow 3 \uparrow 2 = 64$  not 512).

#### 3.4.8 ASCII data

ASCII input of A-Z has the values of 1-26 per digit per letter respectively, thus,

```
*ASK A; TYPE A  
:Z=26.00  
*A A; T A  
:AZ = 36.00
```

This is also true for internal numerical constants like 0NO, 0YES, etc.

(See the IF command for an example of this feature.)

The technique may also be used to create a kind of associative memory:

```
*ASK A; ASK GRADE (A) 2  
:DICK : 95  
  
*ASK A;TYPE GR(A) 2  
:DICK =95
```

#### 3.4.9 Indirect Commands

If a Teletype line is prefixed by a line number, that line is not executed immediately, but is stored for later execution. Line numbers must be in the range 1.01 to 31.99. The numbers 0.0, 1.00, 2.00, 3, etc., are illegal line numbers and are used to indicate the entire group. The number to the left of the point is called the group number; the number to the right is called the step number. Execution of indirect commands is begun by an immediate GOTO or DO command. The GOTO command causes FOCAL to start the program by executing the command at a specified line number (e.g., GOTO 1.3). The GO command causes FOCAL to go to the lowest numbered line to begin executing the program and continues until it runs out of program text. FOCAL can automatically cross group boundaries.

### 3.5 SAVING FOCAL PROGRAMS

#### 3.5.1 Paper Tape

To save a FOCAL symbolic 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. (To save a FOCAL binary program, see Appendix C.1.)

#### 3.5.2 LINC Tape (see Section 2.14.1; TC01 via 8-LIBRARY SYSTEM; PDP-12)

On LINC tape, load FOCAL program as follows:

- a. Load FOCAL binary tape, execute Initial dialog, and call UPDATE.

```
NAME:      START
SA (OCTAL): 200
MEM LOCATIONS: <4600, 7577 >;
```

- b. Call UPDATE again.

```
NAME:      FOCAL
SA (OCTAL): (none)
MEM LOCATIONS: <0, 3377 >;
```

- c. Calling Sequence:

```
FOCAL
START
*
```

- d. Write the desired FOCAL routine.

- e. Give an "L" command. Four octal numbers will be printed, and control will return to the Library System.

```
UPDATE
NAME:      (user's choice)
SA (OCTAL): (none)
MEM LOCATIONS: <0 ><(A), (B) >;
    Where "(A)" and "(B)" mean the first and second octal numbers.
```

- f. To call a program:

```
FOCAL
(user's choice)
START
*
```

#### 3.5.3 Disk Monitor System (see Section 2.14.1)

- a. Build the Disk System.

- b. Load FOCAL into field zero.  
(If the computer has 8K, use the binary loader in field 1.)  
Alternate procedure: Use PIP to place the binary on disk. Then, use LOAD on the disk file. (This procedure is faster for a teletype, but uses more disk space.)
- c. Load Address 200, START, and complete the initial dialogue.
- d. Load Address 7600 and START.
- e. Initialize the disk as follows:
  - .SAVE START!4600-7577;200
  - .SAVE FOCAL!0-3377;
- f. Run FOCAL.
  - .)FOCAL
  - .)START
  - (Create Program)
- g. Save program; return to disk Monitor by giving an L command.
  - .SAVE (name);0,(A) - (B) [note saving page zero]
- h. Run a program (after doing either step f or g).
  - .FOCAL )
  - .CALL (name) )
  - .START ) [linefeed will not occur]
  - \*(FOCAL ready)
- i Steps g and h may be repeated.

#### 3.5.4 Disk System and Extended Functions

To cope with configurations involving deletion of extended functions, proceed as follows:

- a. Load FOCAL and start at 7600;
  - .SAVE START!4600-7577;200
  - .SAVE INIT:0,3200-4577; [note saving page zero]
  - .CALL INIT
  - .START
  - [Dialogue, answer YES]
  - \*L
  - .SAVE FOCAL!0-3377;
- b. To reinitialize a system without some extended functions, type
  - .FOCAL
  - .CALL INIT
  - .START
  - [Dialogue, answer NO, YES, i.e., keep sine and cosine]
  - \*L
  - .SAVE STNY!5200-7577;200
  - .



- c. To create a system without any extended functions, type

```
.FOCAL
.CALL INIT
.START

[Dialogue, answer NO, NO]
*L
.SAVE STNN!5400-7577;200
```

- d. Be sure to use the correct START command with each user program.

(1)

[to use no exponential function version]

```
.FOCAL
.CALL NEXP
.STNY
*
```

(2) or

[to use no cosine function version]

```
.FOCAL
.CALL NCOS
.STNN
*
```

### 3.5.5 Disk System and Extended Memory (see section 2.14.1)

Follow these operations to set up an 8K version of FOCAL on the disk:

```
[Build Disk System]
[Load FOCAL]
[Start at 200]
[Dialogue, answer questions.]
*L
0100 (A)
0121 (B)
3217 (C)
XXXX (D)
.SAVE ST8K! (D) -7577;200
.SAVE FCL8! 0 - 3177;
.SAVE NUL8: 10100; 10113
.
```

The SAVE command for a finished 8K FOCAL program is

```
.SAVE CODE:1(A) - 1(B); 10113
```

where (A) and (B) are the first and second four digit numbers typed out by the L-command. These are the field one bounds of the program text. The value of (D) will depend on the functions retained.

The variables, however, are in field zero. To save a set of data, type:

```
.SAVE DAT8:0;3200-(C);    [note saving page zero, field zero]
```

To set up a null program with a particular data set, type:

```
.FCL8  
.CALL DAT8  
.CALL NUL8  
.ST8K
```

3.5.6 For 4-user FOCAL SAVE command, see Section 4.6.6.

3.5.7 EAE Patch for FOCAL, 1969

7203	3206	DCA	+.3
7204	1256	TAD	MP2
7205	7425	MQL	MUY
7206	0	0	
7207	3253	DCA	MP5
7210	7501	MQA	
7211	3255	DCA	MP3
7212	5227	SNP	+.15

## CHAPTER 4

### PROGRAM SPECIFICATIONS

#### 4.1 MACHINE REQUIREMENTS

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

Scope, an additional 4K memory, and high-speed reader and punch are available options. Additional PT08s are added for extra users.

#### 4.2 DESIGN SPECIFICATIONS

##### 4.2.1 Design Goals

FOCAL is a conversational language and operating system for a basic PDP-8. It is designed to facilitate on-line editing and execution of symbolic programs. (For BNF description, see Appendix F.)

##### 4.2.2 Input

The keyboard, low-speed reader, or high-speed reader may be used for input of program text and for commands to be executed immediately. Keyboard input is double buffered.

4.2.2.1 Input Format - See description of the commands in Chapter 2 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 expanded 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
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 the teletype in-progress-switch (TELSW). If the interrupt handler does output another character, it sets TELS 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 if TELS has been set. If TELS is zero, no further interrupts are expected by the interrupt processor, and the output routine immediately types the character itself and sets TELS 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 room is available. 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 the interrupt system is that it enables the user 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 ?01.00, 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 ?00.00. 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. For example, this could occur if the user continues to type on the keyboard while the program is making computations. Physical evidence of the error is indicated by failure of the computer to echo characters as the user types.

## NOTE

This error could also occur when reading a paper tape program into the text buffer via the low-speed reader. If the output hardware is slower than the input hardware, more text is read in than is being read out of the buffer, resulting in failure of the program to empty the reader buffer as quickly as it is 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 prevent 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. This is essentially a hardware problem since the program is unable to stop the low-speed reader.

### 4.2.4 Organization

4.2.4.1 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 eventually use four words as an option. The exponential range is approximately ten to the six hundredth. Internal accuracy during computations is 6.924 decimal digits.

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.

4.2.4.2 Storage - The major components of the program occupy locations 1-3200. The remaining storage 3200 - 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 one of these lists 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 (see Appendix D).

This important 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 +6 if a parity error occurs.

#### 4.4 INTERNAL ENVIRONMENT

##### 4.4.1 Adding a User's Function; FNEW(Z) (c.f., Section 5.2)

The FOCAL system was designed to be easily interfaced for new hardware such as LAB-8, multiplexed ADCs 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 intended to be 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.

- a. First, patch the function branch table.

```
*FNTABF + 15
XFNEW
```

- b. When control arrives at XFNEW, the X has already been evaluated.

```
XFNEW,      JMS      I      INTEGER      /make 12 bit integer
                                         in AC
                                         /set X - coord.
           DXL
           CLA
```

- c. Now, test for the possibility of another argument.

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

- d. Move past the separating comma.

```
           GETC
           SPNOR
```

- e. Evaluate the second argument.

PUSHJ			/this FNEW is
		EVAL	/not recursive
JMS	I	INTEGER	
DYS;CLA		/set Y and intensify	
SPNOR			
TAD		CHAR	
TAD		MCOMMA	
SZA		CLA	
JMP	I	EFUN3I	

- f. Now, pick up the single letters for display until the end of the function is reached.

DCHR, GETC		
TAD		CHAR
TAD		MRPAR
SNA		CLA
JMP	I	EFUN3I

Char. display routine called here; (for Tektronics Y002, it is simply PRINTC)

JMP	DCHR
-----	------

- g. Definitions from the symbol table are available in Appendix E.

Summary:

- User defined functions must leave their value, if any, in FLAC and return by a `JMP I EFUN3I`.
- The contents of FLAC is converted to an integer in FLAC and in the AC by a `JMS I INTEGER`.
- The floating point arithmetic interpreter is entered by `JMS I 7`.  
(FOCAL uses its own version of the floating point package.)
- The address of the user's function is placed by him in the FNTABF list.
- 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 the function processor. The user should achieve his function implementations using the information given here and in the symbol table without using the actual listing so that changes made by different users may be compatible and so that they may also be relocated easily should any changes be made by DEC. (see Section 4.5.1 for Core Utilization List)
- 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 a comma (,) 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. Doing a PUSHJ; EVAL-1 is equivalent to

GETC;PUSHJ;EVAL.

#### 4.4.2 Internal Subroutine Conventions

4.4.2.1 Calling Sequences - The (AC)=0 unless it contains information for the subroutines. Upon returns (AC)=0 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

SUBR1,0

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/multiple arguments:

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

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, (see Appendix G).

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, PROCESS, PROC, and GETVAR.

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

SORTJ	/go to command
COMLST-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 (i.e., JAS I7)
FGET	FLARG
FMPY I	PT1
.	
EPUT	FLARG
FXIT	/leave floating interpreter

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

IBAR, INPUT X

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

GONE + 11, error 3

4.4.2.2 Subroutine Organization - Figure 4-1 illustrates the internal use of various subroutines. (c.f., Flow Charts in Appendix G).

#### 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, 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 compute 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.

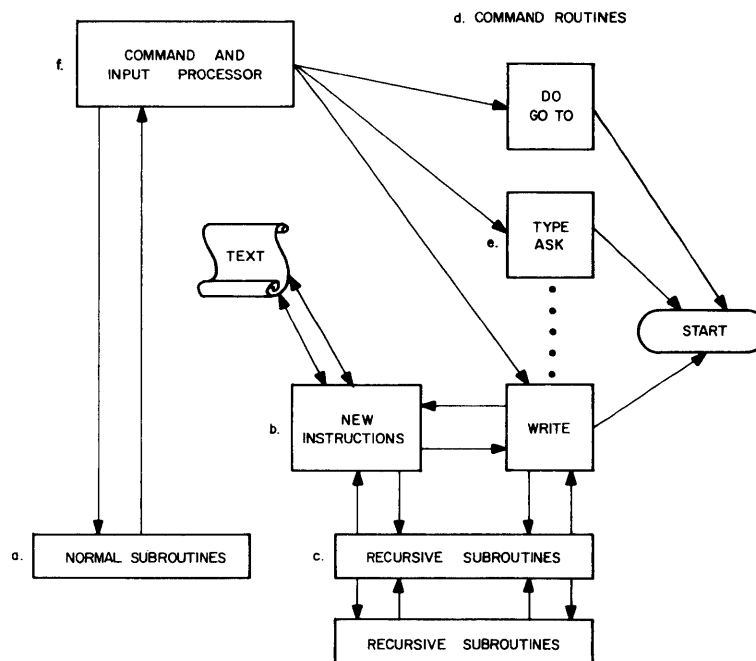


Figure 4-1

#### 4.4.4 Language

The program is written in PAL III with floating point commands, as well as program-defined commands, implemented as subroutine calls. (see Appendix G) The program must be assembled using PAL10.

## 4.5 NOTES

### 4.5.1 Core Utilization

NAMES	PLACE	SEGMENT
	0-15, 17-166	FOCAL (4K)
	167-175	8K
	176-2572	FOCAL (4K)
	2573-2577	8K
	2600-2724	(Interrupt Handler)
	2725-3117	FOCAL (4K)
IOBUF:	3120	(I/O Buffer)
COMEIN:	3140	(Command Buffer)
FRST:	3206	(Text Buffer)
BEGIN:	4420-4577	(Initialization)
	4430-4577	CLIN
FEXP:	4620-4776	(Extended Functions)
ARTN:	5000-5166	[11 free]
FCOS:	5200-5345	[32 free]
TGO:	5400-5577	[ 0 free]
DECONV:	5600-5773	[ 4 free]
FLOUTP:	6000-6157	(Output Conversion)
THISD:	6160-6176	8K
FLINTP:	6200-6317	(Input Conversion)
HREAD:	6320-6377	(High Speed Reader)
FPNT:	6400-7177	(floating interpreter)
MP4:	7200-7377	[none free]
XSQRT:	7400-7502	[FSQT( ) and format buffer]
LIBRARY:	7503-7556	(Single user L command)
XRTD:	7557-7576	8K
Storage of text is	3200-4577	14 functions
	3200-5177	11 functions
	3200-5377	9 functions

### 4.5.2 Extended Functions

Extended Functions may be reinitialized by loading in the second part of main program tape.

Functions are normally deleted by answering the questions asked when FOCAL is initiated.

However, they may also be erased by changing location 0035 to 5377, and locations 401 through 0405 to 2725. Retaining the extended functions allows approximately 1200 characters of text or 170 variables (or any combination in the ratio of 7 characters to one variable). Deleting the extended functions allows approximately 1800 characters or 250 variables.

#### 4.5.3 Error Printouts

Errors    ?01.00  
          ?00.00  
and       ?11.35

Because these errors are time dependent, they may be followed by nonexistent or false line number.

#### 4.5.4 No Interrupts

To read data tapes without running the risk of Keyboard-Input-Buffer overflow (?11.35), it is necessary to remove the interrupt. This action means that Control-C will not work.

To run FOCAL without interrupts, change:

Loc/From	To
63/2676	1353
64/2666	2413
2732/6001	5336
2762/6046	7000

The high-speed punch will now run in parallel with the low-speed punch!

To run the high speed punch at top speed change:

1356/6041	6021
-----------	------

#### 4.5.5 Operating HS Reader Without Interrupts

To run the high-speed reader without interrupts, make the above patches plus two more:

6324/1037	6011
6325/7700	7410

#### 4.5.6 Non-Typing of Program Tapes During Loading

The "echo" feature for the ASR-33 may be suppressed by changing location 2163 to 7000 (from 4551). This will cause only asterisks to be typed as the tape is read. There will not be line feeds or carriage returns. (c.f., 4.7.3.4 for multi-user system)

Any output commands will be typed out in the usual manner, as will diagnostics, answers, etc. Entries from the keyboard will not be typed.

#### 4.5.7 Explanation of NAGSW (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.00 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
Error	4001

PDP-8 code for testing NAGSW:

skip if

Or	One	All	Group
ONE	SMA	--	SMA SZA
ALL	--	SPA SNA	SNA
GROUP	SMA SZA	SPA SZA	SZA

#### 4.5.8 Data Inaccuracies

The logical conclusion from the inequality  $10^8 < 2^{27}$  is that the user can represent 8-digit decimal floating-point numbers accurately by 27-bit floating-point numbers. However, 28 significant bits are needed 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.<sup>1</sup> In FOCAL, 23 bits are used giving 6.9 digit accuracy.

#### 4.5.9 Eliminating = and : in I/O Formats

Leading equal signs and colons in I/O formats are omitted by making the following patch:

Loc/From	To
1216/4551	7600 /:
6002/4551	7600 /=

#### 4.5.10 Estimating the Length of User's Program

FOCAL requires five words for each identifier stored in the symbol table and one word for each two characters of stored program. This may be calculated by

where  $5s + \frac{c}{2} \cdot 1.01 = \text{length of user's program}$   
 $s$  = Number of identifiers defined  
 $c$  = Number of characters in indirect program

If the total program area or symbol table area becomes too large, FOCAL types an error message.

<sup>1</sup>Goldberg, B. "8-Digit Accuracy",  
Communications of the ACM  
Vol. 10, No. 2, February, 1967

FOCAL occupies core locations  $1-3300_8$  and  $4600_8-7576_8$ . This leaves approximately  $700_{10}$  locations for the user's program (indirect program, identifiers, and push-down list). The extended functions occupy locations 4600-5377. If the user decides not to retain the extended functions at load-time, there will be space left for approximately  $1100_{10}$  characters for the user's program.

The L-command may be used to indicate how much core is available for the user.

#### 4.6 FOCAL SYSTEMS

FOCAL systems are designed to take advantage of as many PDP-8 configurations as possible. With this in mind, the system source language is divided into segments which, when loaded together, fit the needs of a user and his particular configuration. Thus, when a user changes his configuration or requirements, he does not need to secure an entirely new FOCAL tape but only to load a new segment corresponding to the change in his configuration. The scheme used also has the advantage of simple maintenance, since changes are made to one source file for all possible systems and in some cases re-assembly of other segments is not needed.

Two source segments create a FOCAL system for a 4K PDP-8. Others are used to create a FOCAL system with (1) ten digit arithmetic, (2) 8K memory, and (3) circular and linear graphics.

The segments of the FOCAL system and their functions are listed in Table 4-1. The ASCII source segments FOCAL.ASC and FLOAT.ASC must be assembled with all configurations and the resulting binary segment, FOCAL.BIN, when loaded makes a one user FOCAL system for a 4K PDP-8.

The segment INIT.ASC is assembled alone, but when INIT.BIN is loaded with FOCAL.BIN into field zero it gives you the initial dialog. If the extended functions are to be retained, it is not necessary to load INIT at all. All corrections for machine type will be made anyway. After FOCAL is started and/or the dialog is completed the user may proceed to load other binary segments.

If a user has an 8K PDP-8 and wants to create a large program with extended precision arithmetic, he need only load FOCAL.BIN, start, and then load 4WORD.BIN, and 8K.BIN as indicated in Table 4-2. If he wants to share his PDP-8 with three other people, he just loads FOCAL.BIN and QUAD.BIN into field one and start.

Intra-references between segments is handled by small multiple assemblies, rather than a large assembly with conditionals for each possible system. For example, to obtain a binary copy of the segment QUAD.BIN, use PAL10 to assemble, QUAD.ASC, FOCAL.ASC, FLOAT.ASC. This assembly produces only the listing and binary files for QUAD which end with the PSEUDO-op's "XLIST" and "NOPUNCH". Tables 4-2 and 4-3 give the allowable combinations of the binary segments to produce legal configurations of the FOCAL system.

Table 4-1  
FOCAL System Source Segments

ASCII Segment Name	Function	Description
FOCAL*	The interpreter & TTY I/O driver.	
FLOAT*	Modified Floating Point Package.	
4WORD	Extended precision overlay to FLOAT (give 10 digits).	(4.6.5)
8K	Allows one user to take advantage of an 8K PDP-8.	(4.6.4)
QUAD	Allows multiple users (up to 4) to use FOCAL or 8K PDP-8.	(4.6.6)
LIBRA <sup>†</sup>	Allows multiple users (up to 7) to run and save FOCAL programs on an 8K PDP-8 with disk.	(2.14.2)
CLIN	The user may have a scope to interact with FOCAL.	(5.8)
PENT	A variation of QUAD allowing five (5) users.	
INIT	The symbolic source for the initial dialog program.	

\*These two segments must be assembled and loaded together for all configurations. They are separated for editing convenience.

<sup>†</sup>Not yet implemented.

Table 4-2  
Allowable FOCAL Systems

1 - Must be loaded into field one 0 - Must be loaded into field zero Y - Command may be used if disk system is built N - Command is illegal * - Command different		
Binary Segment	Allowed Combinations & Subsets are indicated by entries in vertical columns	Minimum Hardware Required
FOCAL	0 0 0 0 1 1 1 1	4K
INIT (optional)	0 0 0 0	
4WORD	0 0      1    1	4K
8K	0 0	8K
QUAD or PENT (non-8/S)	0 0 0 0	8K/PT08s
LIBRA (non-8/S)	0 0	8K/PT08s/DF32
CLIN (optional)	0 1    1	Graphics Terminal
LIBRARY COMMAND (for disk monitor)	Y Y Y Y N N * *	DF32
FOCAL is always loaded first in the proper field.		

Table 4-3  
Variations for FOCAL Systems

Any combination of these three sets ( $2 \times 2 \times 4 = 16$ ),

- |                     |                            |  |
|---------------------|----------------------------|--|
| a. 8K overlay<br>4K | b. Disk Monitor<br>No Disk | c. No Dialogue<br>No ext. functions<br>SINe, COSine only<br>All ext. functions |
|---------------------|----------------------------|--|

or QUAD four-user system or PENT five-user system (PENT is obtained by a modified assembly of QUAD; see listing) may be used with

CLIN graphics (4)  
4WORD overlay  
Neither  
Both

These are formed from only six sections of binary tapes.

The CLIN graphics function can be used for numerical control.

4K FOCAL can be run on the following DEC computers: 5, 8, 8/S, 8/I, 8/L, LINC-8, LAB-8, TSS-8, PDP-12.

- a. Load FOCAL & INIT
- b. do initial dialogue
- c. load any or all of 4WORD, 8K, CLIN.
- d. restart and use

#### 4.6.1 FOCAL Systems Assembly

##### a. Systems programs

```
* ↑C
.RUN T PAL10
*FOCAL.BIN,FOCAL.LST←FOCAL.ZZL,FLOAT.ZZL
*QUAD.BIN,QUAD.LST←QUAD.ZZL,FOCAL.ZZL,FLOAT.ZZL
```

##### b. Initial dialogue

```
* ↑C
.RUN T PAL10
*INIT.BIN,INIT.LST←INIT.ZZL
*
```

##### c. Overlay routines

```
.R PAL10
*4WORD.BIN,4WORD.LST←4WORD.ZZL,FOCAL.ZZL,FLOAT.ZZL
*8K.BIN,8K.LST←8K.ZZL,FOCAL.ZZL,FLOAT.ZZL
*CLIN.BIN,CLIN.LST←CLIN.ZZL,FOCAL.ZZL,FLOAT.ZZL
*
```



#### 4.6.2 FOCAL Binary Paper Tapes

```
.AS DSK D
DSK ASSIGNED

.AS PTP
PTP ASSIGNED

.R PIP

*PTP:↔/ID:QUAD.BIN

*PTP:↔/ID:4WORD.BIN,8K.BIN,CLIN.BIN

*PTP:↔/ID:FOCAL.BIN,INIT.BIN
↑C
```

#### 4.6.3 FOCAL Listings

```
*LPT:↔D:QUAD.LST,4WORD.LST,8K.LST,CLIN.LST,INIT.LST,FOCAL.LST

*TTY:↔/L DTAa:

58: FREE BLOCKS LEFT
FOCAL .ZZL
FLOAT .ZZL
QUAD .ZZL
4WORD .ZZL
8K .ZZL
CLIN .ZZL
INIT .ZZL
PAL10 .SAV
JR36
JR46
```

### 4.7 FOCAL SEGMENTS

#### 4.7.1 8K Single User Overlay - 8K

To increase the size of program, the 8K overlay uses the upper 4K for storage of the user's source text. The maximum number of variables does not change as they are still stored in the lower 8K. Load the overlay after doing the initial dialogue with the 4K version.

#### 4.7.2 Extended Precision Overlay - 4Word

This overlay provides FOCAL with 10-digit accuracy when the 10th digit goes to enable. The overlay increases the number of words needed to store a number from three words to four words. The number of variables that may be stored is decreased accordingly.

Load the overlay after doing the initial dialogue with the 4K version.

#### 4.7.2.1 Double Precision Multiply in Four-Word FOCAL

To multiply two numbers, the product of which is greater than ten digits and yet retain the least significant figures, use a double precision operation.

For example, to multiply:

M = 20243974

by

N = 69732824

let M0 = the 1st 4 digits of M and let M1 = the 2nd 4 digits of M. Similarly, N0 and N1 are the left and right halves of N.

Note the correction of an input error in the high order part of N.

```
*W
C-4WORD@1/69
14.10 ASK !,M0,M1,"*"N0,N1,!
14.20 SET A=M0*N0
14.30 SET B=N0*M1 + M0*N1
14.40 SET C=M1*N1
14.50 SET Z=FITR(C*1E-4)
14.60 SET C=C-Z*1E4
14.70 SET B=B+Z
14.80 SET Z=FITR(B*1E-4)
14.90 SET B=B-Z*1E4
14.99 TYPE !%8,A+Z,%4,B,C,!
*GO

:2024 :3974 * :6928-6973 :2824
= 14116694= 7600= 2576
*
```

#### 4.7.3 Four User Overlay - QUAD

QUAD allows an 8K PDP-8/I, -8/L with up to four teletypes to time-share FOCAL. In effect, each user has the equivalent of a 4K PDP-8 or PDP-12 with FOCAL. The QUAD overlay is located in the lower 4K, and the FOCAL interpreter is located in the upper 4K. Users are traded for one of three other users in the lower 4K. Swapping of users is based upon I/O waits and checkpoints in the FOCAL interpreter.

##### 4.7.3.1 Four User Loading and Operating Procedure

- a. Load 1st binary part into field one. (FOCAL.BIN)
- b. Load 2nd binary part into field one. (QUAD.BIN)

c. Load address

7600  
and START

.SAVE F4UB!0-2177,3000,3600,5400;200  
.SAVE F4UA!0-13220, 14600-17577;

(Any errors made here may require reloading field zero.)

d. (Calling Sequence)

.F4UA  
.F4UB

(If any problem occurs hit stop, record the PC and restart at 200 or reload.)

4.7.3.2 Swapping - At certain points in the FOCAL program it is a pure procedure. If swapping occurs at these times, then only 1K of impure data needs to be saved instead of 4K. This factor of four considerably improves system performance. Such a point is called a checkpoint.

Each time an operating program reaches a checkpoint the executive routine checks to see whether another user should be swapped in at that time.

This check is also made if the operating program goes into a state of waiting for input-output, except for output during use of trace.

4.7.3.3 Workload and Timing

a. Swapping is done on a demand (I/O wait) and a cooperative (checkpoint) basis. Therefore, no clock is needed. Not having a clock reduces system overhead by about ten percent.

b. Fully asynchronous I/O is backed up by large (over 16 characters) and uniform (easy to process) character buffers. Serial to parallel conversion of the bit stream is done in external hardware by PT08 line controllers. This reduces system load by 18 to 30 percent.

c. If each of eight user programs takes less than 100-17 msec to generate one 8-digit output string, then the system is barely output bound and no delay will be observed in response times. The 17 msec is average access time to the disk, and one TTY character takes 100 msec to be typed.

4.7.3.4 Special Controls - A control-R character (TAPE) suppresses echo of input tapes except for the line-feed. A control-T (NOT-TAPE) or Control-C restores the echo of input characters.

It is a good practice to punch a Control-R at the beginning of all off-line tapes. An alternative is simply to type Control-R manually before setting the low speed reader to RUN.

4.7.3.5 Dialogue - There is no initial dialogue with QUAD.

#### 4.7.4 Graphics for Circles and Lines - CLIN

/CLIN - GRAPHICS OVERLAY FOR FOCAL,ZZK PAL10 V133 14-MAR-69 16:01

/CLIN - GRAPHICS OVERLAY FOR FOCAL,ZZK

/FINITE DIFFERENCE EQUATION OF A CIRCLE - FOR FOCAL

/16,2 S P=X-X0;S Q=Y-Y0;S R=FSQT(Q\*2+P\*2)

/16,3 S Z=FNEW(6,3\*R\*C,P,Q,X0,Y0,S/R)

/16,4 S X0=X;S Y0=Y

/LINEAR DIFFERENCE EQUATION OF A LINE

/17,1 D 16,2;S Z=FNEW(R,P/R,Q/R,X0,Y0,0);D 16,4

6057 DXS=6057

6053 DXL=6053

6067 DYS=6067

0035 \*BOTTOM

0035 4437 \*FCIN=1

0407 \*FNTABF+14

0407 4440 FCIN

4440 \*4400+40

4440 4453 FCIN, JMS I INTEGER

4441 7040 CMA

4442 3342 DCA R /SAVE THE POINT COUNT

4443 1340 TAD XXP

4444 3010 DCA AXIN /START DATA POINTERS

4445 1117 TAD M5 /FOR 5 MORE ITEMS

4446 3316 DCA CT

4447 4537 GETA, PUSHJ /COMPUTE EACH ARG,

4450 1612 EVAL-1

4451 1044 TAD EXP /FOUR FIXED POINT RESULTS

4452 1341 TAD LP

4453 3044 DCA EXP

4454 4453 JMS I INTEGER

4455 7200 CLA

4456 1005 TAD P13 /SAVE UNNORMALIZED FORM

4457 3410 DCA I AXIN

4460 1045 TAD WORD

4461 3410 DCA I AXIN

4462 1046 TAD LORD

4463 3410 DCA I AXIN

4464 2316 ISZ CT /TEST FOR END OF DATA

4465 5247 JMP GETA

4466 1046 TAD LORD /TEST FOR CIRCLE OR LINE

4467 7640 SZA CLA

4470 5343 JMP XFCIR

```

4471 7100 XFLIN, CLL /VECTOR PLOT ALGORITHM
4472 1331 TAD X01
4473 1323 TAD P11
4474 3331 DCA X01
4475 7004 RAL
4476 1330 TAD X00
4477 1322 TAD P0
4510 6053 DXL / (6317) - FOR LAB-8
4511 3330 DCA X00
4512 7110 CLL
4513 1334 TAD Y01
4514 1326 TAD Q1
4515 3334 DCA Y01
4516 7004 RAL
4517 1333 TAD Y00
4518 1325 TAD Q0
4519 6067 DYS / (6307) - FOR LAB-8
4512 3333 DCA Y00
4513 2342 ISZ R
4514 5271 JMP XFLIN
4515 5535 JMP I EFUN3I

```

////

```

/TO DISPLAY A POINT X,Y: SET Z=FDIS(X,Y)
/TO DRAW LINE X0,Y0 TO X,Y: DO 17
/TO SET X0,Y0=X,Y: DO 16,4
/TO ERASE SCREEN : TYPE "(ERASE CODE)"
/TO RESET PRINT ORIGIN: TYPE "(RESET CODE)"
/TO DRAW A CIRCLE ABOUT X0,Y0 STARTING AT X,Y
/AND GOING COUNTERCLOCKWISE FOR FRACTION
/OF A CIRCLE ALPHA : SET S=+1; SET C=ALPHA; DO 16
/TO GO CLOCKWISE: SET S=-1; DO 16

```

```

/GROUPS 16 AND 17 CREATE OR USE THE VARIABLES
/X,Y,X0,Y0,Z,R,C,P,Q,K,AND S.
/S MAY BE REPLACED BY A 1 IF DESIRED.

```

4516	0000	CT,	0
4517	0000		0
4520	0000		0
4521	0000	PP,	0
4522	0000	P0,	0
4523	0000	P1,	0
4524	0000	QQ,	0
4525	0000	Q0,	0
4526	0000	Q1,	0
4527	0013	XX,	13
4530	0000	X00,	0
4531	0000	X01,	0
4532	0013	YY,	13
4533	0000	Y00,	0
4534	0000	Y01,	0
4535	0000	KK,	0
4536	0000		0
4537	0000		0
4540	4520	XXP,	PP-1
4541	0014	LP,	14
4542	0000	R,	0

/TO USE AN X-Y PLOTTER, CLIN IS NOT NEEDED; SIMPLY  
/ADD THE FOLLOWING LINES TO GROUPS 16 AND 17 :

```

/16,25 S K=S/R
/16,30 F I=0,6,3*R*C;S P=P-Q*K;S Q=Q+P*K;S Z=FDIS(X0+P,Y0+Q)
/17,10 D 16,2;F I=0,R;S X0=X0+P/R;S Y0=Y0+Q/R;S Z=FDIS(X0,Y0)
/17,20 D 16,4

```

/THE ITERATION PARAMETER "I" MAY BE TAKEN IN GREATER INCREMENTS IF THE  
/SCALE FACTOR IS ALSO CHANGED; I.E.

```

/17,10 DO 16,2;SET K=4/R
/17,15 FOR I=0,4,R;S X0=X0+K*P;S Y0=Y0+Q*K;S Z=FDIS(X0,Y0)

```

4543	4407	XFCIR,	FINT	/CIRCLE ALGORITHM
4544	0324		FGET QQ	
4545	4335		FMUL KK	
4546	6316		FPUT CT	
4547	0321		FGET PP	
4550	2316		FSUB CT	
4551	6321		FPUT PP	
4552	1327		FADD XX	
4553	0000		FXIT	
4554	4453		JMS I INTEGER	
4555	6057		DXS	/(6317) - FOR LAB-8
4556	4407		FINT	/CLEARS AC
4557	0321		FGET PP	
4560	4335		FMUL KK	
4561	1324		FADD QQ	
4562	6324		FPUT QQ	
4563	1332		FADD YY	
4564	0000		FXIT	
4565	4453		JMS I INTEGER	
4566	6067		DYS	/(6307) - FOR LAB-8
4567	7200		CLA	
4570	2342		ISZ R	
4571	5343		JMP XFCIR	
4572	5535		JMP I EFUN3I	

4600 NOPUNCH  
 0001 PAGE  
 FIELD 1  
 XLIST

## 4.8 FOCAL DEMONSTRATIONS

### 4.8.1 One-Line Function Plotting

This example demonstrates the use of FOCAL to present, in graphic form, some given function over a range of values. In this example, the function used is

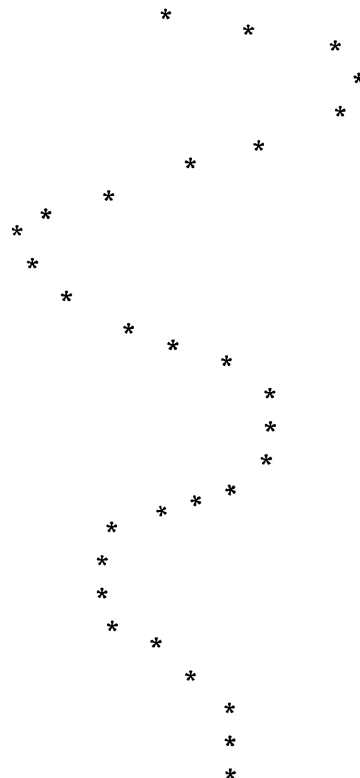
$$y = 30 + 15(\sin(x))e^{-.1x}$$

with  $x$  ranging from 0 to 15 in increments of .5. This damped sine wave has many physical applications, especially in electronics and mechanics (for example, in designing shock absorbers for automobiles).

In the actual coding of the example, the variables  $I$  and  $J$  were used in place of  $x$  and  $y$ , respectively; any two variables could have been used. The single line 08.01 contains a set of nested loops for  $I$  and  $J$ . The  $J$  loop types spaces horizontally for the  $y$  coordinate of the function; the  $I$  loop prints the  $*$  symbol and the carriage return and line feeds for the  $x$  coordinate. The function itself is used as the upper limit of the  $J$  loop showing the power of FOCAL commands.

The technique illustrated by this example can be used to plot any desired function. Although the  $*$  symbol was used here, any legal FOCAL character is acceptable.

```
08.01 F I=0,.5,15; T " ",!; F J=0,30+15*FSIN(I)*FEXP <-.1*I >T " "  
*  
*  
*DO 8.01  
*
```





#### 4.8.2 How To Demonstrate FOCAL's Power Quickly

- a. Load the program and start at 200.
- b. Explain that the initial dialogue gives you options.
- c. Try some other response like MAYBE  $\rangle$  .
- d. Now answer YES  $\rangle$  .
- e. The preceeding has demonstrated the interactive capabilities of the language and the compromises that it permits.
- f. In a 4K machine (4096 words) FOCAL gives the user 15 functions and uses only 3K, leaving enough room to solve up to 6th order simultaneous equations.
- g. The asterisk (\*) means that FOCAL can now respond to your commands.
- h. The basic command is TYPE:  

```
*TYPE      5 ↑ 2 + FSQT (5)  $\rangle$ 
```
- i. Now compute 5 factorial:  

```
*SET      ALPHA=1  
*FOR I=1, 5; SET ALPHA=ALPH*I
```
- j. The answer is ready when the next asterisk is typed out:  
Then type  

```
*TYPE      ALPHA
```

  
for the answer .
- k. Now if you are using a PDP-8 or -8/1, demonstrate a large number:  

```
*SET      A=1  
*FOR      I=1, 300; SET A=A*I
```

  
some time later  

```
*TYPE A  
= 0.395      615
```
- l. Now generate a plot via a stored program:  

```
*1.1 FOR Y=0, .5, 15; TYPE ! ; DO2  
*1.2 QUIT  
*2.1 FOR X=0, 12+10*FSIN(Y) ; TYPE " "  
*2.2 TYPE " * "  
*GO
```
- m. Now use the MODIFY Command to change 10\* to FEXP (Y/6)\* and try again.

#### 4.9 FOCAL Versus BASIC

FOCAL is superior to BASIC, not only in terms of computing power and ease of use, but also in maximum use of the memory space, which is so often limited in small computer systems.

FOCAL contains all the power of BASIC, and in addition provides the following capabilities:

- a. Control of the output format (i.e., precise figure location on a page and graphical representation);
- b. An "immediate" mode, allowing the system to operate as a desk calculator and to execute simple problems without writing a program;
- c. The capability of executing individual "stored program" statements in the immediate mode for debugging and verification;
- d. Built-in symbolic editor capable of searching program statements for specified characters and inserting and deleting characters within a statement, thereby eliminating the retyping of the entire program statement;
- e. Multiple statements may be grouped on each line for more logical ordering of the program;
- f. True multiple level re-entrant subroutining capabilities;
- g. A trace feature which types out selected segments of a program (as the program is executed) to pin point exactly where a program error occurred;
- h. Commands may be abbreviated to one letter; this eliminates wasted typing time when writing a program and increases the available storage space for use by additional program statements;
- i. Programs may be saved on disk and chained together;
- j. Point plot displays, vector displays, X, Y plotters, and analog to digital converters may be operated by FOCAL; this capability can be used in an on-line, real-time fashion;
- k. FOCAL SYSTEMS allow use of several hardware configurations: 8K, 10 digit, display, and multi-user.

## CHAPTER 5

### ADDITIONAL FOCAL APPLICATIONS

#### 5.1 FOCAL FOR THE LAB-8

##### 5.1.1 Standard

Two commands have been added to FOCAL to implement the A to D converter and the oscilloscope display on the AX08.

a. A to D Command:

FADC(N) where N is the channel number in decimal.

The command:

SET Z = FADC(28)

gives the variable Z a value of octal channel 34 depending on the position of the upper righthand potentiometer. The other 3 knobs are channels 29, 30 and 31. A subroutine in FOCAL to read the A to D in volts is as follows:

15.1 ASK CHAN;C-0,1,2,3

15.2 SET X=FADC(28+CH)

15.3 IF (X-256)15.Y,15.4;SET X=X-4096

15.4 SET X=X/255

The input variable is CH for values of 0 to 3, and the output variable is X with values  $\pm$ /volt.

b. Display Command:

The display command has been modified to use only one statement to define X and Y.

SET Z = FDIS(X,Y).

will display a point on the oscilloscope screen defined by points X and Y. X can range between 0-511 and Y from -255 to +255. The variable Z is a dummy. (It is given the value of the integer part of Y.). (c.f., Section 5.8 for circle and sector algorithms.)

#### CAUTION

Since the ADC of the AX08 hardware is an integral part of the display logic, using both display and A and D, may result in splatter of the Y direction of the oscilloscope screen.

##### 5.1.2 Additional (Possible) FOCAL Functions for AX-08

FADC (n): Converts (decimal) channel n. Returns result of conversion.

FDIS (x,y): Loads display X and Y; intensifies point.

FTIM (n): Delays n RC clock pulses ( $n < 4096$ )  
 Returns # of 100  $\mu$ s increments since last used.  
 Xtal clock interrupt is enabled.  
 Interrupt servicing for Xtal clock as follows:

```

SKXK
JMP OTHERS
CLXK
ISF TIME +1
JMP .+3
ISF TIME
NOP
ION
JMP I 0

```

Clock flag servicing will tie up 20% of processor time.

When FTIM is called, do the following sequence:

```

TAD (1002)  /enable Xtal clock, start RC clock
OTEN
get n
SNA
JMP XTIME
CMA IAC
DCA RCNTR
CLRK
SKRK
JMP .-1
ISZ RCNTR
RMP .-4
XTIME, PUT TIME, TIME +1 in FLAC
DCA TIME
DCA TIME +1
return to FOCAL

```

FNEW (a, b, c)

a = 0: Turn on relays indicated by b ( $b \leq 7$ )  
 Turn off relays indicated by c ( $c \leq 7$ )  
 as follows:

```

get b
RAL; RTL
AND (70
OTEN
get c
RTL; RAL
AND (70
CMA
ZTEN
CLA
return to FOCAL

```

a = 1: "and" external register with mask  
 b: mask (octal)  
 c: ignored

```

                                Get characters of b
                                interpret as octal #
                                DCA XMASK
                                XRIN
                                AND MASK
                                XRCL
                                CMA JAC
                                TAD MASK
                                SNA CLA
                                IAC
                                store in FLAC
                                return to FOCAL

a = 2:    "or" external register with mask
b:        mask (octal)
c:        ignored

                                get characters of b
                                interpret as octal #
                                DCA XMASK
                                XRIN
                                AND MASK
                                XRCL
                                SZA CLA
                                IAC
                                store in FLAC
                                return to FOCAL

```

## 5.2 FNEW FOR DATA ARRAYS\*

A new function for 8-K FOCAL is available which uses field one to store data arrays in floating double precision, single precision, and signed integer format. This facility is added to FOCAL via the function call FNEW. The function may be called recursively to any level, and all of the features of FOCAL are retained. In addition an ERASE or ERASE ALL command will not wipe out the array. Hence, variables may be stored for use in successive programs.

### 5.2.1 Storage Requirements

Fits into unused locations in floating point package

### 5.2.2 Usage

5.2.2.1 Loading - Load after FOCAL has been loaded into the machine (and the initial dialogue is executed). Load the first part of the overlay using the Big Loader. If a single precision floating array is desired press CONTINUE. A patch should now be read in to allow a 1980 element array in

\*Originated by University of Georgia, program not supported by DEC.

single precision floating point. If an integer array (maximum number = 3047) is desired press CONTINUE. A patch will now be read in to allow a 3965 element signed integer array.

Restart FOCAL at 200.

#### 5.2.2.2 Calling Sequence - To store a variable Z as array element J:

\* S X=FNEW (J,Z)

or

\* 4.3 S X=FNEW (J,Z)

In addition, X will be set equal to Z.

To call the array element K and set Z equal to this element:

\* S Z=FNEW(K)

i.e., if there is only one argument the instruction is interpreted as a "GET". If there are two arguments it is interpreted as a "PUT".

#### 5.2.3 Recursive Calling

The function FNEW may be called recursively at any level. viz.

\* S Z=FNEW [J, FNEW(J+10)]

sets  $Z = \text{FNEW}(J+10)$  and stores  $\text{FNEW}(J+10)$  in array element J.

\* 3.2 S Z=FDIS (J\*1000) , FDIS( $\text{FNEW}(J) * \text{NORM}$ )

the arguments may be any arithmetic expression. The following are valid:

\* S Z=FNEW (J\*10-3,  $\text{FEXP}(X^2) * Y$ )

\* S Z=FNEW (J,  $\text{FNEW}(K) * \text{FEXP}(\text{FNEW}(L))$ )

#### 5.2.4 Restrictions

Double precision floating:  $0 \leq J \leq 1320$  (23 bits of significance)

Single precision floating:  $0 \leq J \leq 1979$  (11 bits of significance)

Integer Array:  $0 \leq J \leq 3965$  (11 bits of significance)

$|Z| \leq 2047$

#### 5.2.5 Description

The function FNEW protects the binary loader in upper core. The function checks to see if J is too large, but does not check to see if Z is larger than 2047 in the integer array case (c.f., array overlay).

The user, of course, may subdivide this array into any number of smaller arrays, keeping track of his own indices.

### 5.3 DYNAMIC INTERRUPT PROCESSING VIA FOCAL, 1969

This simple patch allows real-time interrupts to initiate execution of a specific FOCAL subroutine (e.g. Group 31) which gains control (i.e., D031) when an interrupt occurs from an external device. The FOCAL subroutine could sample various channels of the A/D converter, set a few constants, then turn off the interrupt, and return to the main FOCAL program. The main FOCAL program will carry out the analysis or output of data during the time between these external device interrupts. The external device could even be an animal and the time between interrupts will be asynchronous and long (between 1 and 1000 seconds), or the external device will be a clock, in which case the time between interrupts will probably not be less than 100 ms or greater than 1 sec.

```
/patch to interrupt processor
(tag assignments from symbol table)
*EXIT                      /replaces H.S. Reader*
    IOT1                    /skip if device
    JMP.+3
    NOP                      /"HINBUF" is cleared
*PC1                        /checkpoint in main program
    JMP I 175                / valid for 8K, also
*167
    DIPCHK                  /Dynamic Interrupt Check
*HINBUF
    1                        /initialized to non-zero
*HREAD
DIPCHK,TAD HINBUF
    SZA CLA
    POPJ
    TAD PC                  /save FOCAL register
    PUSHA
    TAD SPCLN              /(your group #)
    DCA LINENO
    DCA NAGSW
    ISZ HINBUF
    PUSHJ
        DO+1
    POPA
    DCA PC
    POPJ
SPCLN, 7600                /(group 31)
```

The routine in group 31 returns control by "RETURN". This feature does not operate until main program is started. It will operate during execution of a direct command.

## 5.4 SIMULTANEOUS EQUATIONS' SOLUTIONS

This program will work with a set of simultaneous linear equations (in 4K. FOCAL 6 equations is the limit) and output the solutions. To do this the program requests a value "L", the number of equations and variables to be processed. The program then requests the coefficients and constants for each equation, in a matrix like format. The solution values are typed out in a column with the names "X(0)" through "X(L-1)". The program is available through DECUS.

## 5.5 FAST FOURIER TRANSFORMS PROGRAMS

The FAST FOURIER TRANSFORMS Program is designed to accept samples of a complex wave pattern as input and, through a FOURIER analysis, describe its component sine and cosine waves in terms of amplitudes and frequencies.

The user inputs a number "N", which must be a power of two, (in 4K. FOCAL, "4" is the limit) and which describes the number of samples to be used in the analysis. Next the samples, which are wave height measurements taken at regular intervals, are requested. Output is in the form of two columns (side by side), the left of which describes the cosine wave components while right hand column describes the sine wave components.

It should be noted that because the number of samples is always a power of two, the number of complex multiplications is cut drastically. For this reason computation time is also greatly reduced.

### NOTE

In order to use this program, the extra extended function FX(A,B) must be loaded into memory via the BIN loader.

### FAST FOURIER TRANSFORMS

W

C-FOCAL., 1968

```
01.08 A "POWER OF 2 ",NU
01.10 S N=2 ↑NU;S TP=2*3.14159/N
01.18 S S=N/2; L=1;S Q=S-1;S H=1-NU
01.20 F I10,N-1;A !;A !,XR(I);S XI(I)=0
01.22 S SR=XR(Q+S)+XR(Q);S XR(Q+S)=XR(Q)-XR(Q+S);S XR(Q)=SR
01.24 I (Q) 1.26,1.26;S Q=Q-1;G 1.22
01.26 I (L-NU) 1.28,1.54,1.28
01.28 S L=L+1;S S=S/2;S H=H+1;S P=N-1;S Z=1/(2↑(-H) )
01.32 S C=1
01.34 S U=FITR(P*Z);S K=FX(NU,U)*TP
01.36 S CO=FCOS(K);S SN=FSIN(K)
01.38 S GR=CO*XR(P)+SN*XI(P);S GI=CO*XI(P)-SN*XR(P)
```



```

01.40 S Q=P-S;S SR=GR+XR(Q);S SI=GI+XI(Q);S XR(Q)=XR(Q)-GR
01.42 S XI(Q)=XI(Q)-GI;S XR(P)=SR; XI(P)=SI
01.46 S P=P-1; I (-FABS [C-S]) 1.48; I (P-S+1) 1.52,1.26,1.52
01.48 S C=C+1;G 1.34
01.52 S P=P-S;G 1.32
01.54 F I=0,N-1;S K=FX(NU,I);T !,%3.2,2*XR(K)/N," "2*XI(K)/N
*
*C-TRANSFORM OF INTERFERENCE PATTERN FORMED BY MIXING A SINE
*C-WAVE OF AMPLITUDE 1.0 AND A COSINE WAVE OF AMPLITUDE 1.5
*
*GO
POWER OF 2 :3
:1.5
:1.768
:1
:-.353
:-1.5
:-1.768
:-1
:.353
++0.00 ==+0.00
==+1.50 ==-1.00
==+0.00 ==+0.00
==+0.00 ==-0.00
==+0.00 ==+0.00
==+0.00 ==+0.00
==+0.00 ==+0.00
==+0.00 ==+0.00
==+1.50 ==+1.00*
*

```

```

/FNEW(u,v) for FFT
*BOTTOM
4377
*FNTABF+1Y
XFX
*4400
XFX, JMS I INTEGER
Dca U
PUSHJ
EVAL-1
JMS I INTEGER
CIA
DCA T2
DCA LORD/low order
TAD U
CLL RAR
DCA U
TAD LORD
RAL
DCA LORD
ISZ T2
JMP .-7
JMP I EFUN3I

```

## 5.6 TRAVEL VOUCHER TO EXPENSE VOUCHER CONVERSION PROGRAM

Though FOCAL is not a business oriented language the use of FOCAL in business applications is not impossible. Such a use is seen in the TRAVEL VOUCHER TO EXPENSE VOUCHER CONVERSION program with which the user may ease the task of reporting his expenses after a business trip.

Working from the input of the number of the days using the expense account and the categorized input of the expenses encountered (all amounts must be entered in terms of cents rather than dollars) during that period, the computer tallies and itemizes

- a. the daily expenses and
- b. the totals of the expenses over the entire period.

The data, thus summarized, are very easily transcribed onto an employee expense voucher.

### TRAVEL VOUCHER TO EXPENSE VOUCHER CONVERSION PROGRAM

C-FOCAL., 1969

```
01.01  T  !! "EXPENSE ACCOUNTER (TYPE ALL AMOUNTS IN PENNIES)"
01.05  ERASE
01.10  ASK %6.02,!"HOW MANY DAYS?" DAYS,!
01.20  IF (DAYS) 1.1,1.1; FOR I=1,DAYS; DO 5
01.40T  !! " THE TRIP TOTALS ARE";F I=1,30;T " "
01.41T  "GRAND"!
01.60  SET LO=LT; SET ME=ET
01.70  SET OJ=OT; SET MI=MT; DO 7
01.80  TYPE " $"!!!!!!
01.90G  1.05

05.10  ASK !!!"BRKFST " B1
05.20  ASK !"LUNCH " B2
05.30  ASK !"DINNER " B3
05.40  ASK !"SNACKS " B4
05.50  ASK !"MILES TRAVELED ? "B5; SET B5=B5*9; TYPE " $ B5/100; DO 6
05.60  ASK !"HOTEL " B6
05.70  ASK !"OTHER " B7
05.73  ASK !"TELE " B8
05.75  A !"TAXI "C1
05.76A  !"PARKN "C2
05.77A  !"TOLL "C3
05.85  ASK !"MISC. " B9
05.90  TYPE !"THE DAILY TOTALS ARE"!
05.91  SET LO=B6; SET ME=B1+B2+B3+B4
05.92  SET OJ=B5+C1; SET MI=B9+B8+B7+C2+C3
05.93  TYPE "DAY NO."; DO 7.1
05.94  TYPE !%3,I," "; DO 7.2; DO 7.3
05.95  SET LT=LT+LO; SET ET=ET+ME
05.96  SET OT=OT+OJ; SET MT=MT+MI
```

06.10 ASK " MISC. TRAV. ? "B6; SET B5=B5+B6

07.10T " LODGING MEALS OTHER TRAV. MISC. TOTAL

07.15 T !

07.20T %8.02,LO/100," "ME/100," "OJ/100," "MI/100,"

07.30 T (LO+ME+OJ+MI)/100

\*

\*

\*G

EXPENSE ACCOUNTER (TYPE ALL AMOUNTS IN PENNIES)

HOW MANY DAYS ? :2

BRKFST :150

LUNCH :170

DINNER :645

SNACKS :35

MILES TRAVELED ? :36

\$ =+ 3.24 MISC. TRAV. ? :0

HOTEL :1400

OTHER :0

TELE :40

TAXI :0

PARKN :250

TOLL :0

MISC. :0

THE DAILY TOTALS ARE

DAY NO.	LODGING	MEALS	OTHER TRAV.	MISC	TOTAL
=+ 1	=+ 14.00	=+ 10.00	=+ 3.24	=+ 2.90	=+ 30.14

BRKFST :98

LUNCH :192

DINNER :650

SNACKS :30

MILES TRAVELED ? :23

\$ =+ 2.07 MISC. TRAV. ? :0

HOTEL :1400

OTHER :398

TELE :285

TAXI :0

PARKN :250

TOLL :0

MISC. :0

THE DAILY TOTALS ARE

DAY NO.	LODGING	MEALS	OTHER TRAV.	MISC	TOTAL
=+ 2	=+ 14.00	=+ 9.70	=+ 2.07	=+ 9.33	=+ 35.10

THE TRIP TOTALS ARE

LODGING	MEALS	OTHER TRAV.	MISC	GRAND TOTAL
=+ 28.00	=+ 19.70	=+ 5.31	=+ 12.23	=+ 65.24 \$

## 5.7 TWINS DEMO

The TWINS DEMO Program is an interesting experiment in the applications of plotting with a visual scope display unit. It must be noted that several functions must be loaded into memory before this program will operate. This program is an integral part of curve fitting. The Twins Demo requires V68/I Control with Tektronix 611 Scope. (i.e., 340 control)

### TWINS DEMO

```
W
C-FOCAL., 1969
01.05S  A=FDIS ( ) +FDXS ( ) +FNEW(2) + FNEW (256)
01.10S  A=.2;S  SW=19
01.70F  T=0,.05,6.284;S  T2=T+3.14159/4;DO 1.8;DO 15
01.75   G 2.1
01.80S  R=4*FSIN(T) +4;S  X=8+R*FCOS(T2);S  Y=32+R*FSIN(T2)
02.10   F Y=28.5,A,32;S  K=( (Y-30.5)/1.5) ↑ 2;S  X=9-(K*K-K);DO 15
03.10   F X=7.4,A,10.5;S  Y=26.5-( (X-9) ↑ 2)/2;DO 15
04.10   S X=10.5;F  Y=17,2*A,24.8;DO 15
05.10   F X=7.2*A,8;S  Y=22-7*(X-7); DO 15
06.10 F  X=10.5,A,15;S  Y=26-FSOT(5*(X-10) );DO 15
07.10 F  X=11.5,A,14.5;D 8.5
07.20 F  X=14.5,.2*A,15;D 8.5
08.10 F  X=3,A,4.6;DO 8.4
08.20 F  X=11,A,12;DO 8.4
08.30   G 9.1
08.40   S K=X-7;S  Y=12+(K*K)/4;DO 15
08.50S  Y=21-FSQT(6.25-(X-12.5) ↑ 2);D 15
08.60S  Y=(X-7) ↑ 2-1;D 15
08.70S  X=5+FSIN(3.14159*(Y-12)/7);D 15
09.10 F  Y=0,2*A,16;S  X=12-( (Y-8) ↑ 2)/64;DO 15
10.10 F  X=2,A,4.5;S  K=X-3;S  Y=K*(K*(.47*K-.5)+1.03)+26;DO 15
11.10 F  X=2,(.2*A),2.85;D 8.6
11.20 F  X=4.7,.2*A,6;D 8.6
12.10F  Y=4.5,2*A,12;D 8.7
12.20F  Y=15,2*A,25;D 8.7
13.10F  X=5.3,.3*A,6;S  Y=-7*(X-6);DO 15
14.10F  Y=12,2*A,24;S  K=( (Y-15.5)/11) ↑ 2;S  X=5.5+12.5*(K*K-K);DO 15
14.20F  Y=4,2*A,12;S  K=Y-8.5;S  X=8.1-FSQT(27-K*K);DO 15
14.30R
```

### NOTE

Group 15 must be supplied to scale X, Y and call appropriate display for the device. (c.f., Section 5.8)

# APPENDIX A

## FOCAL COMMAND SUMMARY

<u>Command</u>	<u>Abbr</u>	<u>Example of Form</u>	<u>Explanation</u>
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

- type new text, or
- form-feed to go to the next occurrence, or
- bell to change the search character, or
- rubout to delete backwards, or
- left arrow to kill backwards, or
- carriage return to end the line, or
- 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.

<u>Command</u>	<u>Abbr</u>	<u>Example of Form</u>	<u>Explanation</u>
COMMENT	C	C - compute area	If a line begins with the letter C, the remainder of the line will be ignored.
CONTINUE	C	C - ignore temporarily	
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 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	( )	- Napierian Log
FDIS	(X,Y)	- Scope Functions
FADC	( )	- Analog to Digital Input Function
FNEW	( )	- User Function
FX	( )	- Extra User Function

# APPENDIX B ERROR DIAGNOSTICS\*

Table B-1  
Error Diagnostics of FOCAL, 1969

Location	Code	Meaning
	?00.00	Manual Start given from console.
	?01.00	Interrupt from keyboard via control-C.
0250	?01.40	Illegal step or line number used.
0316	?01.78	Group number is too large.
0340	?01.96	Double periods found in a line number.
0351	?01.:5	Line number is too large.
0362	?01.;4	Group zero is an illegal line number.
0440	?02.32	Nonexistant Group referenced by 'DO'.
0464	?02.52	Nonexistant line referenced by 'DO'.
0517	?02.79	Storage was filled by push-down list.
0605	?03.05	Nonexistant line used after 'GOTO' or 'IF'.
0634	?03.28	Illegal command used.
1047	?04.34	Left of "=" in error in 'FOR' or 'SET'.
1064	?04.52	Excess right terminators encountered.
1074	?04.60	Illegal terminator in 'FOR' command.
1147	?04.:3	Missing argument in Display command.
1260	?05.48	Bad argument to 'MODIFY'.
1406	?06.06	Illegal use of function or number.
1466	?06.54	Storage is filled by variables.
1626	?07.22	Operator missing in expression or double 'E'.
1646	?07.38	No operator used before parenthesis.
1755	?07.:9	No argument given after function call.
1764	?07.;6	Illegal function name or double operators used.
2057	?08.47	Parenthesis do not match.
2213	?09.11	Bad argument in 'ERASE'.
2551	?10.:5	Storage was filled by text.
2643	?11.35	Input buffer has overflowed.
5042	?20.34	Logarithm of zero requested.
5644	?23.36	Literal number is too large.
6543	?26.99	↑ Power is too large or negative.
7111	?28.73	Division by zero requested.
7405	?30.05	Imaginary square roots required.
	?31.<7	Illegal character, unavailable command, or unavailable function used.

\*The above diagnostics apply only to the version of FOCAL, 1969, issued on tape DEC-08-AJAE-B

## B.1 OBTAINING ERROR CODES VIA ODT36

To obtain error codes via ODT36, proceed as follows:

- a. Start ODT at 3600.
- b. User types underlined letters:

(change, from,	to)	
4320/1357	<u>1275</u> ( <u>line feed</u> )	
4321/4745	<u>3067</u> ( <u>line feed</u> )	(LINENO)
4322/1675	<u>4552</u> ( <u>line feed</u> )	(PRNTLN)
4323/4246	<u>7000</u> ( <u>carriage return</u> )	
63/2676	1355 (C.R.)	(OUTDEV, OUTL)

- c. then

<u>M 7777</u>	<u>7777</u> ( <u>line feed</u> )
4273/0001	<u>4400</u> (C.R.)
4565W	(ERROR 2)

Calling addresses and error codes will be printed here. The first two and last error codes (00.00,01.00,31.<7) are always the same.



## APPENDIX C

### EXPLANATION OF NEW INSTRUCTIONS

#### C.1 NEW INSTRUCTIONS (see Table C-1)

##### C.1.1 Push Down List Instructions

The user's push down list begins at the start of the floating point package and grows up toward the last variable. The initial value of the push down list pointer is contained in location "BOTTOM". The pointer is kept in an auto-index labeled "PDLXR". The instructions used to manage the list are given below:

PUSHA	places the contents of the AC onto the list as the current entry
POPA	adds the current entry of the push down list to the AC,
PUSHF	saves a group of data, normally a floating point entry. This instruction is followed by a pointer to a 3 word (or 4 word) group of data. These 3 or 4 words are placed on the push down list as the current entry.
POPF	restores a 3 or 4 word group of data from the current entry on the push down list according to the pointer which follows the instruction. The location "MFLT" contains either -3 or -4 and determines the number of words affected by "PUSHF" and "POPF".
PUSHJ	calls subroutine which is pointed to by the word following the instruction. The return address is placed on the push down list as the current entry.
POPJ	the current entry is used as a return address from a subroutine.

##### C.1.2 Character Handling Instructions

These instructions are used to pick-up, save, and print characters for processing by FOCAL. Characters are fetched from the user's storage area or from the ASR-33 input buffer. Character conversion between 8 and 6 bits and the trace feature are handled by these routines.

PRINTC	is used to print a character. If the AC is zero upon entry then the character in "CHAR" is printed. If the AC is non-zero, then the contents of the AC is printed.
READC	Reads a character from the user's input buffer (ASR-33 input) and echos all characters except line feeds and rubouts. The character is placed into "CHAR".
PACKC	places the 8-bit character in "CHAR" into the user's storage area. If the character is a rubout the previous character is deleted from the user's area and a back-slash is echoed via "PRINTC". The character is

	converted into 6-bit code. The auto index "AXIN" and the flip-flop "XCTIN" are pointers to the user's storage area.
GETC	this instruction fetches the next character from the right or left side of the word pointed to by "AXOUT" and "XCT" and places it into "CHAR". If a question mark character is detected the dump switch "DMPSW" is flipped. If the dump switch is on then the character in "CHAR" is printed via "PRINTC".
SPNOR	Blanks and leading zeroes are ignored by repeated calls to "GETC".

### C.1.3 Character Testing Routines

These guide the interpreter through the source text. They are testing routines used throughout FOCAL in interpreting the program and in other instances.

SORTC	the character in "CHAR" is classified according to an ASCII list which is pointed to by the location following the instruction. If the character is found in the list an exit is made to the location following the list pointer. If no character is found exit is made to the second location following the list pointer. If the character was found in the list then "SORTCN" contains the position relative to zero in the list searched. The list is terminated by a negative word.
SORTJ	the character in "CHAR" or in the AC is classified according to a list as per "SORTC". If the character is found in the ASCII list, then a jump to an address is made from a second list. The second list is pointed to by the 2nd location following call. If the character is not found then exit is made as per "SORTC". "SORTCN" is not changed, however.
TESTC	this instruction fetches the next non-space and classifies it as a terminator, number, function, or letter. The instruction then skips zero, one, two or three cells accordingly.
TESTN	"CHAR" is classified according to whether it is a period (no skip), number (skip two), or other (skip one). If "CHAR" is a number then its binary value is in "SORTCN".
TSTLPR	This instruction skips the next instruction if the AC contains a left parenthesis.

### C.1.4 Line Number Handling Instructions

This group is used in manipulating line data and line numbers.

TSTGRP	If the group of the line number in the AC is equal to the group on the line in "LINENO" the next instruction is skipped.
PRNTLN	the coded line in "LINENO" is printed as a decimal fraction with group number and the step number separated by a decimal point.
GETLN	"SPNOR" is called and a line number is built in "LINENO" via calls to "GETC". "NAGSW" is set to indicate whether the line number was a group, line, or "ALL" designator.
FINDLN	the line number coded in "LINENO" is searched for in the user's text area. If the line is found, the auto-index "AXOUT" and "XCT" are set to point to the line's text and an instruction is skipped. If the line is not found, the pointer "AXOUT" is set to point to the next higher line and no instructions are skipped. "THISLN" points to the line found on the next larger line and "LASTLN" points to the previous/less line.
ENDLN	"ENDLN" links the line in the user's storage area to the rest of his text. It uses the result of the "FINDLN" instruction to accomplish this. The new end of the user's buffer is set-up in "AXIN". This command is used for insertion of new text, reconnecting after a deletion, and reconnection after Modify.

Table C-1  
New Instructions

PUSHJ = JMS I . XPUSHJ	/RECURSIVE SUBROUTINE CALL
POPA = TAD I POLXR	/RESTORE AC
POPJ = JMP I . XPUPJ	/SUBROUTINE RETURN
PUSHA = JMS I . XPUSHA	/SAVE AC
PUSHF = JMS I . PD2	/SAVE GROUP OF DATA
POPF = JMS I . PD3	/RESTORE GROUP
GETC = JMS I . UTRA	/UNPACK A CHARACTER
PACKC = JMS I . PACBUF	/PACK A CHARACTER
SORTJ = JMS I . SORTB	/SORT AND BRANCH ON AC OR CHAR
/NUMERICAL LIST -1	
/ADDRESS LIST - NUMERICAL LIST	

Table C-1 (Cont)  
New Instructions

SORTC = JMS I . XSORTC	/SORT CHAR
PRINTC = JMS I . OUT	/PRINT AC OR CHAR
READC = JMS I . CHIN	/READ ASR-33 INTO CHAR AND PRINT IT
PRNTLN = JMS I . XPRNT	/PRINT C (LINENO)
GETLN = JMS I . XGETLN	/UNPACK AND FORM A LINENUMBER
FINDLN = JMS I . XFIND	/SEARCH FOR A GIVEN LINE
ENDLN = JMS I L XENDLN	/INSERT LINE POINTERS
RTL6 = JMS I . XRTL6	/ROTATE LEFT SIX
SPNOR = JMS I . XSPNOR	/IGNORE SPACE AND LEADING ZEROS
TESTN = JMS I . XTESTN	/PERIOD: OTHER: NUMBER
TSTLPR = JMS I . LPRTST	/SKIP IS 5 < SORTCN < 11 (I . E. AN L-PAR)
TSTGRP = JMS I . GRPTST	/SKIP IF G(AC) = G (LINENO)
TESTC = JMS I . XTESTC	/TERM; NUMBER; FUNCTION; LETTER
ERROR2 = JMS I .	/EXCESS SOMETHING ERROR
ERROR3 = JMS I .	/MISCELLANEOUS ERROR
ERROR4 = JMS I . ERR2	/FORMAT ERROR

# APPENDIX D FOCAL CORE LAYOUT

Table D-1  
Focal Core Layout-Usage

Mnemonics	What
ZERO	
START	FOCAL PROPER
BUFBEG	BUFFER AREA
BEGIN	INITIAL DIALOGUE
FEXP	
(BET 2+ 3)	
ARTN	
(FLAG 3 +1)	
FCOS	EXTENDED FUNCTIONS
(FLOA + 11)	
(TEMPO + 1)	
DECONV	OUTPUT CONVERSION
(INFIX +5)	
FLOUTP	INPUT- OUTPUT
(OUTOG+4)	ROUTINES
FLINTP	
(P43+1)	
FPNT	
ACMINS	FLOATING-POINT INTERPRETER
(RARI+1)	
DNORM	
(BUFFER + 10)	
BINARY	
(RIM)	LOADERS

Table D-2  
Detailed FOCAL Core Layout

Miscellaneous  
Numbers  
Floating-Point Working Area  
Constants  
New Instruction Pointers  
Variables

### START

Command/Input  
Line Read Routine  
'DO' Routine  
Push-POP Routines  
'GOTO' and 'WRITE' and Misc.  
'IF', 'SET', 'FOR' and Misc.  
'ASK', 'TYPE', 'MODIFY'  
  
"GETARG" - Recursive Routine  
"SPNOR", "TESTN", "POPJ"  
'RETRUN'  
"EVAL" - Recursive Routine  
OPNEXT - read operator  
ARGNXT - read operand  
ETERM - evaluate terminator  
FLOP - floating operations called  
ENUM - number processor  
EFUN - function processor  
ELPAR - left parens processor  
EFUN3 - function returns  
"DELETE" - Recursive Routine  
DOK - group delete  
DONE - garbage collection  
"FINDLN" - Normal Routine  
Find exact match or next larger  
'ERASE' command processor  
"GETC" - unpack text and trace  
"ENDLN", "PRNTLN"  
I/O Subroutines  
Interrupt Processor  
ERROR Processor  
"PACKC" - pack text  
Rubout routine

Table D-2 (Cont)

*3120		I/O Buffer Command Buffer Text Buffer Begins
T E X T / V A R I A B L E S / . ./	*4400 -	Once-Only Code SELF-START
P U S H D O W N L I S T		CLEAR ALL FLAGS TYPE MESSAGE
	*3600	ODT-JR (for X-FUN)
	*4600	ODT-JR (for dialogue)

Floating Point Routines  
(c.f., Section 4.5.2)

*4600	Extended Functions
*5400	I/O Controller
*6400	Interpreter
*7600	Binary Loader or 8-SYS LIB Bootstrap or Disk Monitor Bootstrap
*7756	Rim Loader
End of Field Zero	
Field One	
Command Buffer	Extended Text Storage

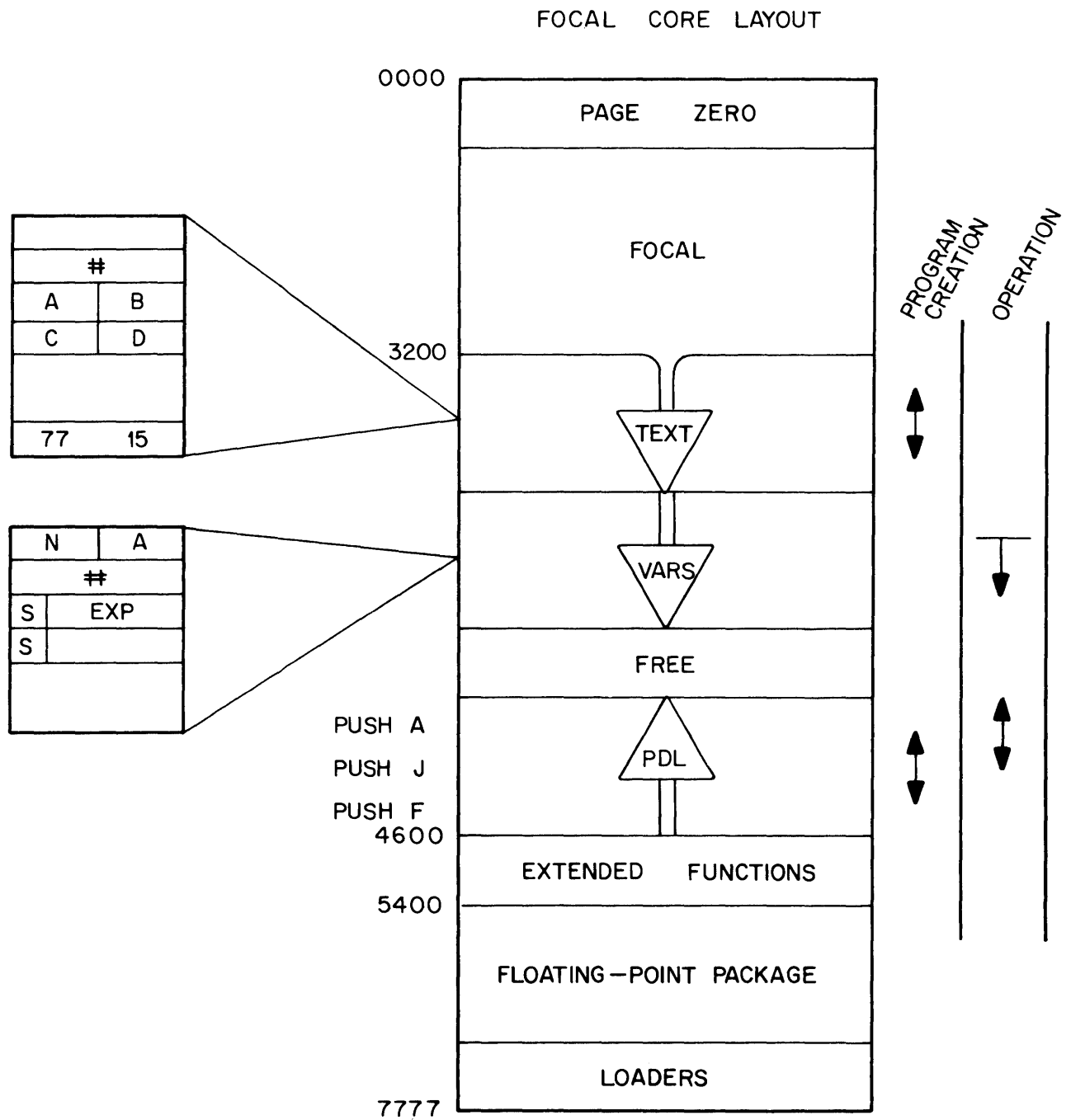


Figure D-1 FOCAL Core Layout  
Dynamic Storage



# APPENDIX E SYMBOL TABLE AND OTHER TABLES/LISTS

## E.1 SYMBOL TABLE

/FOCAL.ZZM	PAL11	V515	10-APR-69	19138	PAGE 121			
A	0045		BFXX	4556	COMEIN	3140	DUBDIV	7261
ABSOL	6751		BMOVE	1255	COMEOU	3206	DUBLAD	5733
ABSOL2	6153		BOTTOM	0035	COMGO	1163	DUMLN2	2012
ABSOL3	7375		BUFFBEG	3217	COMLST	0774	DV3	7267
ABSOLV	5571		BUFFER	7470	COMMFN	0614	E	0042
AC1H	0241		BUFR	0060	CON1	5037	EBELL	0512
AC1L	0042		BUFRS	1300	CONTIN	1147	ECALL	1601
ACMINS	6673		BUFRSP	3045	CONTN	0076	ECCR	2630
ACTING	0701		BUFST	5531	CSTAR	0225	ECHO	0454
ACTION	4420		C	0047	CTABS	0353	ECHOLS	1624
ACTIVE	0037		C100	0006	C	0041	EFOP	0056
ACTVP	1143		C140	2554	DATUM	7102	EFUN	1743
ADD	0061		C144	6140	DATUMA	7252	EFUN2	1754
ADDR	0040		C200	0123	DCONP	6303	EFUN3	2017
ADONE	6673		C200M	0065	DCONT	0471	EFUN3I	0136
AF	4677		C260	0113	DCOUNT	6143	ELPAR	1763
ALF1	4760		C3	5346	DDTJP	0004	END	0134
ALF2	4763		C5	5342	DEBGSW	0026	ENDFI	6243
ALFZ	4755		C7	5336	DECK	0040	ENDLN	4556
ALGN	6570		C9	5332	DECKP	0107	ENDT	0135
ALIGN	6623		CCR	0077	DECON	5627	ENUM	1732
ALIST	1372		CDF	7000	DECONV	5600	EOUT	0474
ALISTP	0072		CDF1	6211	DECP	5533	EP7	0052
ALPHA	1436		CEX1	6504	DECR	5521	EPAR	1710
AMOUNT	6722		CEXP	6503	DELETE	4565	EPAR2	1765
ARCALG	4732		CF	4705	DF	4710	ER5	4555
ARCRN	5024		CFRS	0133	DGRP	0425	ERASE	2204
ARGNXT	1723		CFRSX	0137	DGRP1	0441	ERG	2225
ARTN	5000		CHAR	0066	DIG	5543	ERL	2222
ASHFT	6665		CHARM	0026	DIGIT	5713	ERR2	2726
ASK	1202		CHIN	2155	DIGITS	0006	ERROR2	4566
ATEI	4465		CHKCNT	1053	DIV1	5754	ERROR3	4566
ATES	4513		CHKCON	1052	DIV2	6757	ERROR4	4566
ATLIST	1570		CHRT	6133	DIVINE	7150	ERROR5	2725
ATSW	0056		CIA	7041	DLISTP	0100	ERT	2214
AXIN	0010		CIF	6202	DMDONE	7063	ERV	2217
AXOUT	0017		CIF1	6212	DMPSW	0100	ERVX	2237
R	0046		CLA	7200	DMULT	7004	ESCA	2532
RACK	5503		CLCU	7427	DMULT4	7036	ETERM	1647
RASER	0616		CLF	0076	DNORM	7335	ETERM1	1627
RASES	1540		CLL	7100	DNUMPR	5714	ETERM2	1655
RASEX	0617		CMA	7040	DO	0420	ETERMN	1644
RDUMP	0071		CML	7020	DOK	2111	EVAL	1613
REGIN	4371		CNTR	0057	DONE	2127	EX1	0040
BELLX	0534		CNTRLC	0324	DOONE	0463	EXASK	2662
REND	4442		CNTRLX	0331	DOUBLE	0127	EXCHCK	1037
BET1	4771		CNTRM	0024	DPCVPT	6302	EXCHE	1072
BET2	4774		CNTRT	0032	OPN	6305	EXCHEC	2615
BETA	0010		CODET	0044	OPT	6145	EXGO	1007
BETZ	4766		COL	1255	OSAVE	5640	EXGON	1215
RF	4702		COMBOT	0226	DTABLE	0070	EXIT	2646
RFX	4557		COMBUF	0132	DTST	5647	EXIT1	5034

/FOCAL,ZFM

PAL1.

V515

10-APR-69

19:38

PAGE 121-1

EXIT2	5302	FLOUT	5556	HOLD	0036	KINT	2625
EXIT3	7363	FLOUTP	6000	HOLD1	1276	KRB	6036
EXITJ	2661	FLPT	6465	HOLD0	1277	KSF	6031
FXMOD	2657	FLSU	6505	HORD	0045	KSF1	6401
EXP	0044	FLTONE	2405	HREAD	6321	KSF2	6421
EXPRIN	2600	FLTXR	0014	HREAD2	6324	KSF3	6441
EXPRN	1060	FLTXR2	0015	HSGO	6364	KSF4	6461
FXPRNT	1000	FLTZER	2407	HSP	0273	L1	5126
EXRD	1014	FM12	6142	HSPSW	6375	L2	5131
FXREAD	2605	FNEG	5163	HSPX	6361	L3	5134
FXRED	1054	FNOR	7000	HSR	0273	L4	5137
EXSWP	1142	FNPT	4554	HSWITC	6343	L8A	4550
EXTR	2313	FNTAPF	0374	HTST	6376	L8AX	4553
F	0043	FNTARL	2165	I33	2414	L8AY	4552
FCONT	1171	FOR	1041	IAC	7001	L8B	4551
FCOS	5200	FOUTPU	0130	IBAR	0212	LASTLN	0025
FCOUNT	5535	FPAC1	7474	IBUFI	0106	LASTOP	0055
FEND3	2267	FPNT	6400	IBUFO	0105	LASTV	0031
FEXP	4620	FPRNT	5465	IECALL	1037	LCON	0371
FEXT	0000	FRST	3206	IF	1013	LG2E	4713
FG02	6011	FRSTX	3215	IF1	1035	LIBRAR	7503
FG03	6027	FSIN	5205	IF3	1025	LINENO	0067
FG04	6034	FXIT	0000	IGNOR	0217	LIST3	0077
FG05	6070	G8L	4466	IGNORE	0447	LIST6	0072
FIG01	6221	GECALL	1462	ILIST	0771	LIST7	0074
FIG04	6261	GEND	2334	IN	5513	LISTGO	1370
FINCR	1065	GERR	0340	INBUF	0034	LISTL	0023
FINDLN	4555	GET1	2330	INDEV	0064	LISTP	1165
FINDN	2246	GET3	2345	INDRCT	6463	LOG2	5157
FINFIN	1137	GETARG	1403	INFIK	2401	LOG5	5142
FINKP	1133	GETC	4545	INITL	3001	LOG6	5145
FINPUT	7131	GETLN	4554	INITL4	3011	LOG7	5150
FINT	4407	GETSGN	1045	INLIST	0570	LOG8	5153
FISW	0052	GETVAR	1407	INORM	6307	LOOKUP	4571
FIX	6724	GEXIT	0352	INPUT	0756	LOOP01	6431
FIXM	6753	GFND1	1505	INPUTX	0271	LORD	0046
FLAC	0044	GINC	0070	INSUR	0036	LP7	7556
FLAD	6506	GLIST	1377	INTEGE	0053	LPRTST	2035
FLAG1	5162	GO	5021	INTRPM	0201	M100	0101
FLAG2	4725	GOCR	0451	INTRPT	2603	M10PT	6147
FLARG	2030	GONE	0232	IOBUF	3120	M11	0121
FLARGP	0125	GOTO	0603	IOF	6002	M12	2413
FLDV	7107	GRPTST	0744	ION	6001	M137	2357
FLEX	6515	GS1	1437	IOIX	0110	M140	2556
FLGT	6467	GS2	1461	IPART	1040	M144	6137
FLIMIT	1075	GS3	1441	IRETN	0227	M2	0111
FLINTP	6200	GS4	1454	ITABLE	6573	M20	0105
FLIST1	0577	GSERCH	1426	ITER1	7470	M200	0064
FLIST2	0574	GTEM	0021	JUMP	6462	M20M	0056
FLMY	6563	GZERR	0362	K5	5525	M240	0114
FLOG	5040	HINBUF	0037	KEY	0321	M240M	3046
FLOP	1674	HLT	7402	KEYX	0447	M260	1526

/FOCAL,ZFM

PAL10

V515

10-APR-69

19138

PAGE 121-2

M271	1527	NEGP	4724	P	0000	PDP	4562
M4	6141	NEWU	0042	P10	0053	PDP5	4570
M40	2356	NEXT0	1146	P100	0342	PDP5X	4463
M40M	0057	NXTU	1145	P1000	0046	PDP8I	4567
M4M	0061	NL1	7301	P13	0005	PEQ	6135
M5	0120	NL2	7326	P14	0706	PER	0102
M6M	1162	NL2000	7332	P140	0532	PI	5312
M77	0103	NL3777	7350	P17	0107	PI2	5036
MBREAK	2602	NL4000	7330	P177	0106	PIOT	5316
MC200	0446	NL5777	7352	P17M	0054	PLCE	5536
MCOM	1136	NL7775	7346	P2	4566	PLS	6026
MCR	0116	NL7776	7344	P20	0055	PM2000	1144
MCRM	0063	NOECHO	0465	P2000	0373	PNTR	0031
MD	5526	NOP	7000	P27	6750	POPA	1413
MDECK	0043	NORF	6513	P277	0110	POPF	4544
MEQ	1135	NORM	6567	P2M	0707	POPJ	5541
MF	0602	NORMF	7147	P3	2034	PPTEN	6144
MFLT	0117	NOUSRS	0073	P337	0075	PRINTC	4551
MIF	7260	NOX	6675	P37	0062	PRINTO	7550
MINE	5662	NOX1	6711	P377	2553	PRNT	2442
MINSKI	0051	NOX2	6704	P4	0060	PRNT2	3114
MINUS2	7153	01	4370	P40	2552	PRNT8	7527
MINUSA	0112	02	4561	P4000	0124	PRNTI	6132
MINUSE	6301	04	4412	P43	6310	PRNTLN	4553
MINUSZ	5663	05	4563	P6777	0050	PROC	0611
MLISTP	0077	06	4564	P7	4565	PROGES	0610
MOD	5215	OBUFF0	0104	P7000	0047	PSIN	0165
MODIFY	1256	OBUFF1	0103	P7576	0764	PT1	0030
MOVE15	1232	OBUFF0	0102	P7600	0104	PTCH	0126
MOVE20	1243	OFFDEC	4422	P77	0122	PTEN	6275
MP1	7254	OM12	5530	P7700	0101	PTST	1457
MP11	0575	ONDECK	4421	P7740	0372	PUSHA	4542
MP177	0445	ONE	4716	P7750	0763	PUSHF	4543
MP2	7256	OOUT	4544	P7757	0051	PUSHJ	4540
MP3	7255	OP	3115	P77M	0045	R6	5441
MP4	7200	OPMINS	6565	PA1	2524	RAL	7004
MP5	7253	OPNEXT	1622	PACBUF	2502	RANO	1530
MP6	7210	OPTABL	1731	PACKC	4546	RAR	7010
MPER	0115	OPTR0	2663	PACKST	0027	RAR1	6571
MPLUS	5664	OPTRI	2665	PACX	2530	RAR2	6572
MQ	0035	OPTR0	2664	PALG	5261	RDIV	0152
MQA	7501	OPUT	5532	PARITY	0302	READC	4552
MRO	0444	OTHER	0215	PARTES	2047	RECOVR	2740
MSPACE	5665	OUT	2465	PC	0022	RECOVX	2761
MULDIV	7101	OUTA	5536	PC1	0614	REMAIN	5712
MULT	6566	OUTCR	2476	PCHECK	5245	REPT	6146
MULT10	5667	OUTDEV	0063	PCHK	0510	RESOL	6752
MULT2	5715	OUTDG	6154	PCK1	2535	RESOL3	7376
MULTY	4752	OUTL	1354	PCM	0101	RESOL5	6304
MX	0533	OUTX	2475	PD2	0534	RESOLV	7173
MZERO	0067	OVER1	0043	PD3	0554	RESTAR	0003
NAGSW	0065	OVER2	0047	PDLXR	0013	RESTOR	0304

/FOCAL.ZZM

PAL12

V515

10-APR-69

19:38

PAGE 121-3

RESTP	6377	SNL	7420	TESTA	4561	XABS	2014
RESUME	2623	SORTP	1314	TEXTT	0744	XACTIO	0643
RET	5452	SORTC	4550	TEXTA	1610	XADC	1343
RETRN	1563	SORTCN	0054	TEXTC	0075	XB	2655
RETURN	5536	SORTJ	4547	TEXTF	0017	XBUF	0516
REVIT	7146	SPA	7510	TEXTPM	0074	XCOM	0620
RFC	6014	SPECIA	6777	TGO	5400	XCT	0020
RMF	6244	SPL	7000	THIR	7257	XCTIN	0062
RND2	5527	SPLAT	3051	THISLN	0023	XDECK	0600
ROOTGO	7461	SPHOR	4560	THISOP	0024	XDELET	0062
POT	2557	SQCON1	7467	TINTR	1241	XDYS	1142
ROUND	6151	SQEND	7465	TLIST	1400	XENDLN	2360
RRR	6012	SRETA	0261	TLIST2	1404	XF	4560
RSF	6011	SRNLST	1363	TLIST3	2377	XFIND	2242
RTL	7006	START	0177	TLS	6046	XGETLN	0302
RTL6	4557	STARTL	5064	TPC	6044	XI33	2666
RTR	7012	STARTV	0060	TQUOT	1232	XIN	6306
RUR1	3004	SURS	1517	TRAD	6573	XINPUJ	5666
RUR2	3042	SZA	7444	TRC1	1163	XINT	1160
RUR3	3030	SZL	7430	TRC2	1164	XKEY	0412
RUR4	3037	T	0000	TSF	6041	XOUTL	2676
RUR5	3041	T1	0032	TSF1	6411	XP0PJ	1565
RUBIT	2555	T12	4426	TSF2	6431	XPR	1062
SAC	0033	T2	0071	TSF3	6451	XPR2	1064
SADR	6150	T2H	2624	TSF4	6471	XPRNT	2425
SAVAC	2670	T3	0033	TSTGPP	4563	XPRNTI	1013
SAVE	3751	TARLF	6464	TSTLPR	4562	XPUSWA	0477
SAVLK	2601	TAG1	6723	TTY	0322	XPUSHJ	0521
SBAR	1302	TASK	1204	TTYE	0347	XR10	0010
SCHAR	1273	TASK4	1253	TW0	4721	XR11	0011
SCONT	1270	TCF	6042	TWOP1	5306	XR12	0012
SCOUNT	5534	TCRLF	1251	TYPE	1203	XR13	0013
SET	1041	TCRLF2	1246	TYPE2	1226	XRAN	1553
SETW	0527	TDUMP	3052	UNDECK	0633	XRAR2	7365
SETWI	0023	TELSW	0016	UPAR	0066	XRSTAR	0312
SEX	1340	TELSW1	0275	USERNO	0041	XRT	0011
SEXC	0740	TELSW2	0276	USERTS	1210	XRT2	0012
SFOUND	1306	TELSW3	0277	UTF	2276	XRTL6	0413
SGOT	1312	TELSW4	0300	UTO	2305	XSGN	2010
SIGN	7124	TELSW5	0301	UTRA	2274	XSORTC	0721
SIGNF	0050	TEM	5156	UTX	2316	XSPNOR	1517
SILENT	0343	TEMP	4726	VAL	0032	XSQ2	4676
SIN	2662	TEMPH	0025	WALL	0664	XSQR	5326
SING	0471	TEMPT	0027	WORDS	0003	XSORT	7400
SINGLE	2636	TEMPX	0030	WRITE	0635	XT3	0717
SKP	7410	TEN	6271	WTEST2	0653	XTDUMP	0535
SLK	0034	TENPT	6152	WTESTG	0667	XTESTC	0700
SMA	7500	TERMS	1770	WX	0673	XTESTN	1533
SMIN	6136	TEST2	6736	X	5322	XTTX	0727
SMP	6101	TEST4	7366	X1	5035	XTTY	0710
SMSP	6134	TESTA	0322	X2	4675	XXTTY	0742
SNA	7450	TESTC	4564	XA	2656	XYZ	2451

/FOCAL.ZZM      PAL10    V515    10-APR-69      19138    PAGE 121-4  
                 ZERO    6520

ERRORS DETECTED: 0

RUN-TIME: 32 SECONDS

6K CORE USED

## E.2 OTHER TABLES AND LISTS

/LIST OF FUNCTION ADDRESSES, (NAMES ARE IN "FNTABL")

		FNTABF=,			
0373	2014	XABS	/ABS	-ABSOLUTE VALUE	
0374	2010	XSGN	/SGN	-SIGN PART	
0375	1161	XINT	/ITR	-INTEGER PART	
0376	1143	XDYS	/DIS	-DISPLAY AND INTENSIFY	
0377	1553	XRAN	/RAN	-RANDOM NUMBER	
0400	1344	XADC	/ADC	-READ ANALOG TO DIGITAL CONVERTER	
0401	5000	ARTN	/ATN	-	
0402	4620	FEXP	/EXP	-EXPONENTIAL FUNCTIONS	
0403	5040	FLOG	/LOG	-	
0404	5205	FSIN	/SIN	-TRIG FUNCTIONS	
0405	5200	FCOS	/COS	-	
0406	7400	XSQRT	/SQRT	-SQUARE ROOT	
0407	2725	ERROR5	/NEW	-USER DEFINED FUNCTIONS	
0410	2725	ERROR5	/COM	-	
0411	2725	ERROR5	/X	-	
0412	0000	XRTL6,	0	/ROTATE AC LEFT SIX - "RTL6"	
0413	7106	CLL	RTL		
0414	7006	RTL			
0415	7006	RTL			
0416	5612	JMP	I XRTL6		

		COMLST=,		/ENGLISH_FRENCH
0775	0323	323	/COMMAND	DECODING LIST
0776	0306	306	/SET	- ORGANIZE
0777	0311	311	/FOR	- QUAND
1000	0304	304	/IF	- SI
1001	0307	307	/DO	- FAIZ
1002	0303	303	/GOTO	- VA
1003	0301	301	/COMMENT	- COMMENTE
1004	0324	324	/ASK	- DEMANDE
1005	0314	314	/TYPE	- TAPE
1006	0305	305	/LIBRARY	- ENTREPOSE
1007	0327	327	/ERASE	- BIFFE
1010	0315	315	/WRITE	- INSCRIS
1011	0321	321	/MODIFY	- MODIFIE
1012	0322	322	/QUIT	- ARRETE
1013	0212	212	/RETURN	- RETOURNE
				/(ASTERISK)=EXPANDABLE COMMAND

	1164	COMGO=, /COMMAND ROUTINE ADDRESSES	
1164	1042	SET	
1165	1042	FOR	
1166	1014	IF	
1167	0417	DO	
1170	0604	GOTO	/(REFERENCED)
1171	0615	COMMENT	
1172	1203	ASK	
1173	1204	TYPE	
1174	7503	LIBRARY	
1175	2204	ERASE	
1176	0636	WRITE	
1177	1257	MODIFY	
1200	0177	START	/RETURN TO COMMAND MODE VIA 'QUIT'
1201	1563	RETRN	
1202	6361	HSPX	/ACTIVATE THE HIGH SPEED READER

	2165	FNTABL=,	
2165	2533	2533	/ABS
2166	2650	2650	/SGN
2167	2636	2636	/ITR
2170	2565	2565	/DIS
2171	2630	2630	/RAN
2172	2517	2517	/ADC
2173	2572	2572	/ATN
2174	2624	2624	/EXP
2175	2625	2625	/LOG
2176	2654	2654	/SIN
2177	2575	2575	/COS
2200	2702	2702	/SQT
2201	2631	2631	/NEW
2202	2567	2567	/COM
2203	0330	0330	/X

/LIST OF CODED FUNCTION NAMES

/CONTROL TABLE

0354	0451	IGNORE	/L.T.
	0355	CTABS=,	
0355	0456	ECHO	/A-HOME
0356	0333	CNTRLX	/B
0357	0326	CNTRLC	/C-END OF MESSAGE
0360	0333	CNTRLX	/D
0361	0333	CNTRLX	/E
0362	0333	CNTRLX	/F
0363	0456	ECHO	/G - BELL
0364	0333	CNTRLX	/H
0365	0333	CNTRLX	/I
0366	0467	NOECHO	/J - LF,
0367	0333	CNTRLX	/K
0370	0467	NOECHO	/L -FF,
0371	0453	GOCR	/M -C,R,
0372	0333	CNTRLX	/N -
0373	0333	CNTRLX	/O
0374	0333	CNTRLX	/P
0375	0333	CNTRLX	/Q
0376	0345	SILENT	/R-TAPE
0377	0333	CNTRLX	/S- (7000) - FOR DEBUGGING
0400	0351	TTYPE	/T-NOT TAPE
0401	0333	CNTRLX	/U
0402	0333	CNTRLX	/V
0403	0333	CNTRLX	/W -E,O,MEDIA
0404	0456	ECHO	/X-ERASE
0405	0333	CNTRLX	/Y
0406	0333	CNTRLX	/Z
0407	0451	IGNORE	/C
0410	0451	IGNORE	/A
0411	0451	IGNORE	/J
0412	0456	ECHO	/UPAR -
0413	0453	GOCR	/LEPTAR=GORO



4WORD (10 DIGIT) OVERLAY FOR FOCAL.ZZK PAL10 V133 14-MAR-69 15:54

/4WORD (10 DIGIT) OVERLAY FOR FOCAL.ZZK

	0004	WORDS=4	
	0012	DIGITS=12	
	0052	*FISW	
0052	0000		0
	0070	*GINC	
0070	0006		WORDS+2
	0116	*MFLT	
0116	7774		-WORDS
	3210	*FRST+2	
3210	0355	TEXT @C-4WORD@	
3211	6427		
3212	1722		
3213	0400		
	5526	*MU	
5526	7766		-DIGITS /EXTENDED LENGTH OF OUTPUT FORMAT
5527	0013		DIGITS+1/RND2
	5310	*TWOPI+2	
5310	3755		3755 /CORRECT CONSTANTS
	5314	*PI+2	
5314	3755		3755
	5320	*PIOT+2	
5320	3755		3755
	6143	*DCOUNT	
6143	7765		-DIGITS-1
	6277	*PTEN+2	
6277	3146		3146 /CONSTANT ONE
	6402	*FPNT+2	
6402	7410	SKP	/DO NOT CLEAR OVERFLOW WORDS
	6540	*ZERO+20	
6540	7000		NOP
	6736	*TEST2	
6736	0043		43
	7036	*DMULT4	
7036	3275		DCA DATUM=5
	7105	*MULDIV+4	
7105	7000		NOP
	7072	*DMDONE+7	
7072	7000		NOP

	7260	*MIF	
7260	7735		-43
	7271	*DV3+2	
7271	1043		TAD OVER1
7272	1047		TAD OVER2
7273	3253		DCA MP5
7274	7004		RAL
7275	1042		TAD AC1L
7276	1046		TAD LORD
7277	3256		DCA MP2
7300	7004		RAL
7301	1045		TAD HORD
7302	1041		TAD AC1H
7303	7420		SNL
7304	5312		JMP ,+6
7305	3045		DCA HORD
7306	1253		TAD MP5
7307	3047		DCA OVER2
7310	1256		TAD MP2
7311	3046		DCA LORD
7312	7200		CLA
7313	1254		TAD MP1
7314	7004		RAL
7315	3254		DCA MP1
7316	1200		TAD MP4
7317	7004		RAL
7320	3200		DCA MP4
7321	1335		TAD DNORM

/COMBINE ONE POSITION AND

/SAVE RESULT

/ADD OVERFLOW

/SKIP IF OVERFLOW

/UPDATE FLAG

/CLEAR ACCUMULATOR

/SAVE OVERFLOW BITS CIRCULARLY

0001

NOPUNCH  
FIELD 1  
XLIST

/8K OVERLAY FOR FOCAL,ZZK

/TEXT IS IN FIELD 1; VARIABLES AND POL ARE IN FIELD 0

/,SAVE ST8K!(D)-7577;200  
 /,SAVE FCL8!0-3377;  
 /,SAVE NUL8:10100;10113  
 /,SAVE NAM8:10100-(B);10113

6201 CDF=6201

2010 T=10

2000 P=0

2000 FIELD 0

	0100	LINE0=100	
	0022	*PC	
0022	0000		0
	0031	*LASTV	
0031	3206		COMEOUT
	0060	*BUFR	
0060	0126		LINE1
	0131	*COMBUF	
0131	0010		10
	0132	*CFRS	
0132	0100		LINE0
	0134	*ENDT	
0134	0126		LINE1
	0166	*166	
0166	2565	DPC,	ROT+5 /PC
0167	6160	DTHIS,	THISD /THISLN
0170	6173	DPT1,	PT1D /PT1
0171	7557	DXRT,	XRTD /(TAD I XRT)
0172	7564	DAXIN,	AXIND /(OCA I AXIN)
0173	2572	DAXOUT,	AXOUTD /(TAD I AXOUT)
0174	0120	DLIB,	DLIB8 /LINK FOR 8K L-COMMAND

////////////////////////////////////

```

      0001      FIELD 1

0000  0000      *0000      0      /ZERO PC
0001  0000      0
0002  0000      0      /TDUMP DATA
0003  0000      0
0004  0000      0
0005  5051      5051
0006  0060      BUFR
0007  0126      LINE1

      0100      *LINE0
0100  0000      0
0101  0000      0
0102  0355      TEXT @C-8K FOCAL @
0103  7013
0104  4006
0105  1703
0106  0114
0107  4000
0110  6171      6171
0111  6671      6671
0112  7715      7715

0113  6201      ST8K,   CDF P      /START 8K USER FILE AT THIS ADDRESS
0114  1007      TAD 7
0115  3406      DCA I 6
0116  6202      CIF P
0117  5525      JMP I RL1B
0120  6002      DL1B8,  IOF
0121  1406      TAD I 6
0122  3007      DCA 7
0123  6203      CIF CDF P
0124  5525      JMP I ,+1
0125  7600      RL1B,   7600      /RETURN TO DISK MONITOR,
      0126      LINE1=,

      0000      FIELD 0
      0000      *0000
0000  0000      0

      NOPUNCH
      XLIST

```

## APPENDIX F FOCAL SYNTAX

Table F-1  
Syntax in Backus Normal Form

```

<immediate command> ::= <program statement> C.R.
<indirect command> ::= <line #> <program statement> C.R.
<line #> ::= <group no.> * <line no.>
<group no.> ::= 1-31
<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> <subscript> <line #>; |
    IF <space> <subscript> <line #>, <line #>; |
    IF <space> <subscript> <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 NAME>
<Library Command> ::= CALL | SAVE | DELETE | LIST

```

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

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

<digit> : : = 1-9 | 0

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

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

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

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

<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 characters>

<control character> : : = [ bell ] <search character> |  
[ form ] | [ line-feed ] | C.R. |

[ ↑C ] | ← | [ rub-out ]

<Variable> : : = <letter> | <letter> <character> |  
<Variable> <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 | ITR |  
ABS | SGN | RAN | NEW |

#### NOTE

Spaces are ignored except when required.

Table F-2  
FOCAL Commands In French

Commandments Francais Pour Le Calculateur Electronique "IGOR"

English	French	Letter
1. SET	ORGANIZE	O
2. FOR	QUAND	Q
3. IF	SI	S
4. DO	FAIS	F
5. GOTO	VA	V
6. COMMENT	COMMENTE	C
7. ASK	DEMANDE	D
8. TYPE	TAPE	T
9. LIBRARY	ENTREPOSE	E
10. ERASE	BIFFE	B
11. WRITE	INSCRIS	I
12. MODIFY	MODIFIE	M
13. QUIT	ARRETE	A
14. RETURN	RETOURNE	R

CE N'EST PAS PARFAIT  
MAIS "IGOR" EST INTELLIGENT  
IL COMPRENDRA

NOTE

"IGOR" refers to PDP-8/1

# APPENDIX G ILLUSTRATIONS

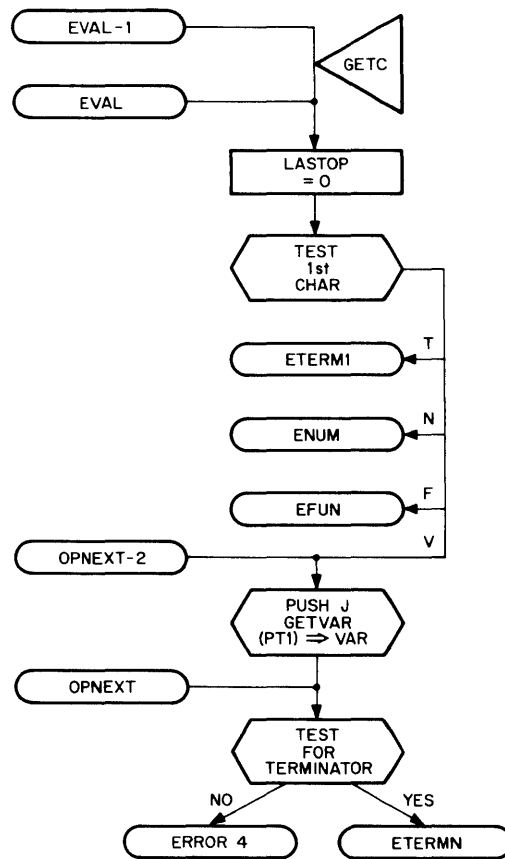


Figure G-1 (Sheet 1) Arithmetic Evaluation



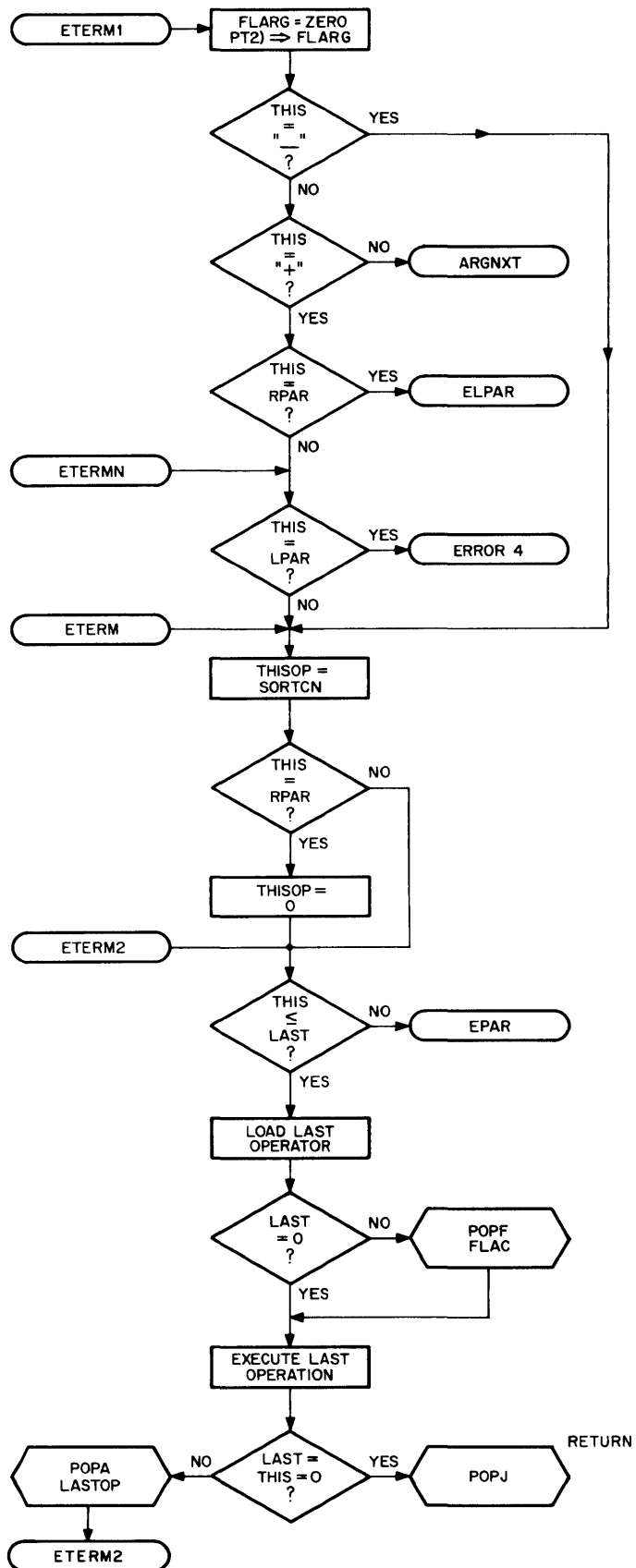
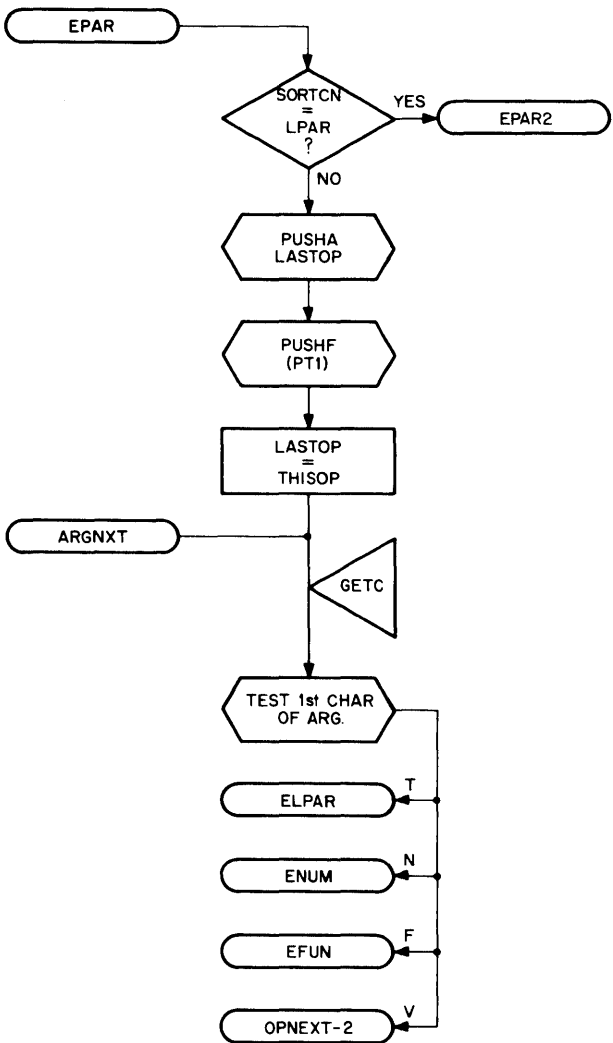
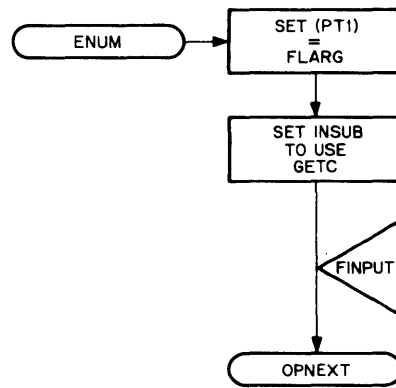


Figure G-1 (Sheet 2) Arithmetic Evaluation

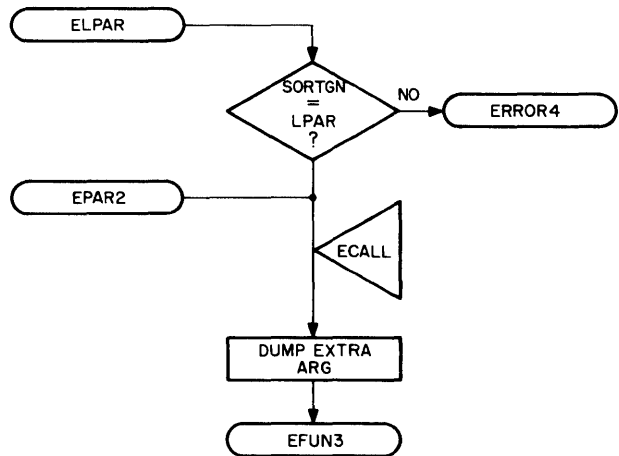


Analysis of Operands



Analysis of Sub-Expressions and Constants

Figure G-1 (Sheet 3) Arithmetic Evaluation



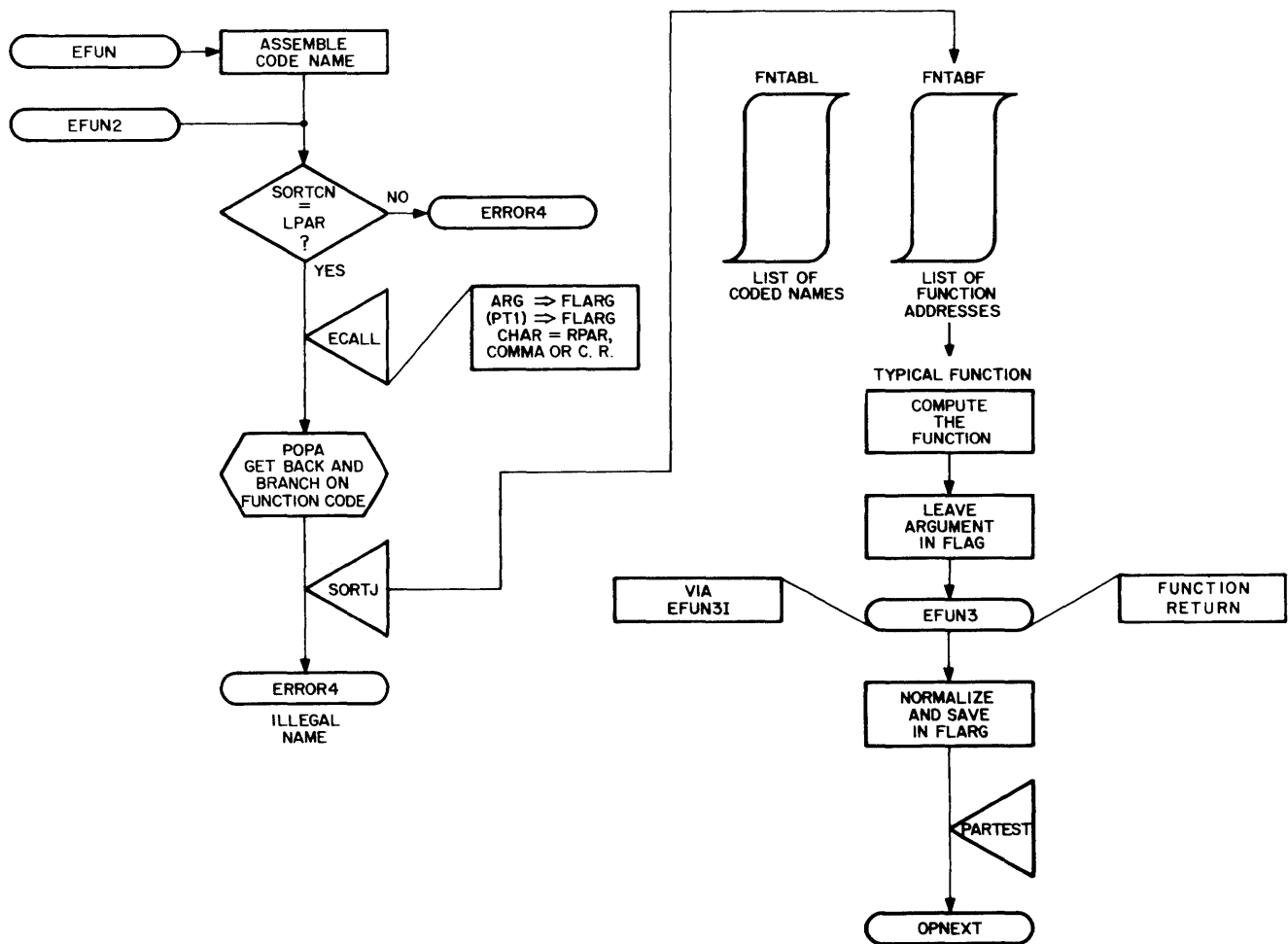


Figure G-1 (Sheet 4) Arithmetic Evaluation (Analysis of Functions)

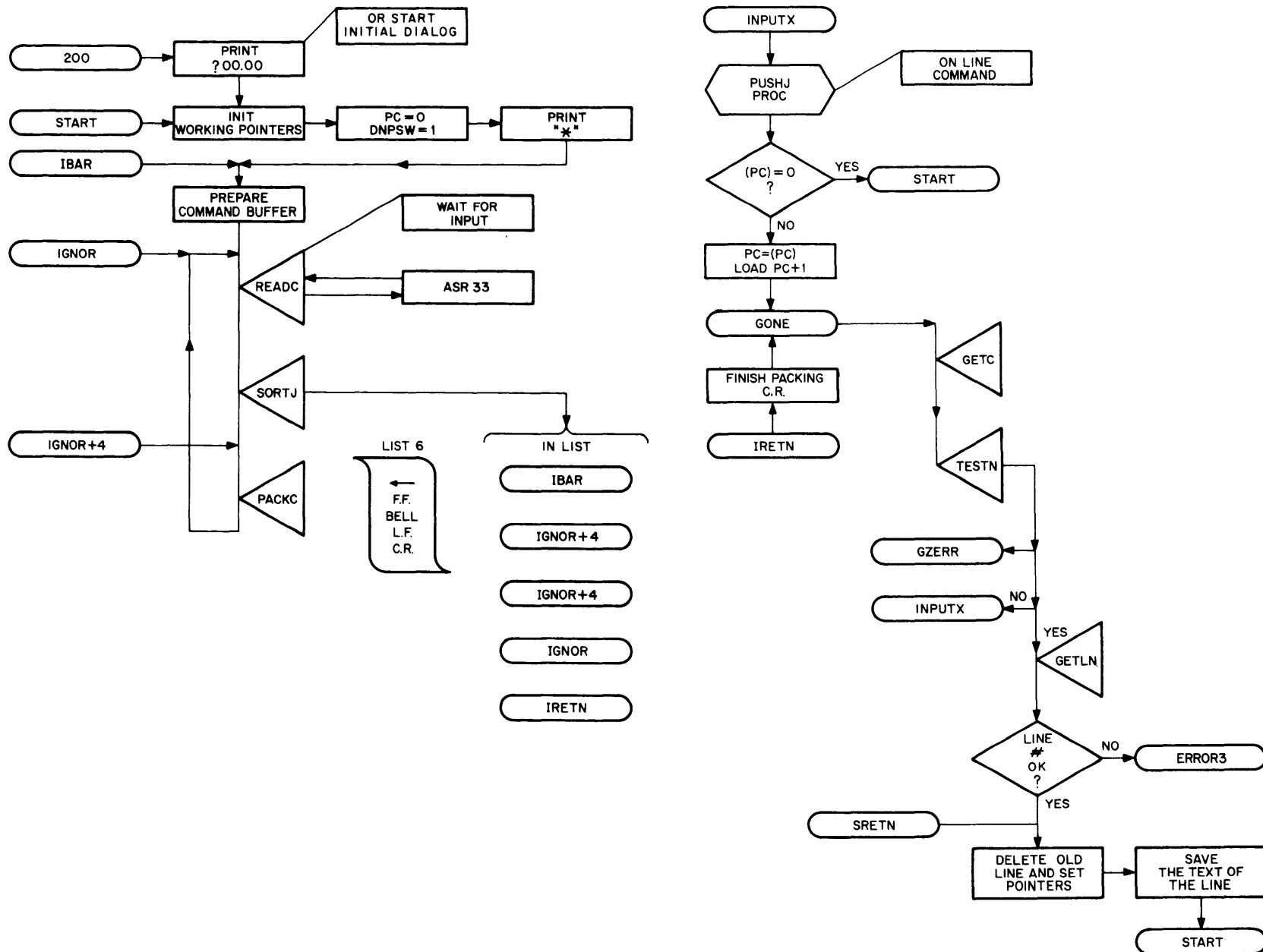


Figure G-2 Command/Input

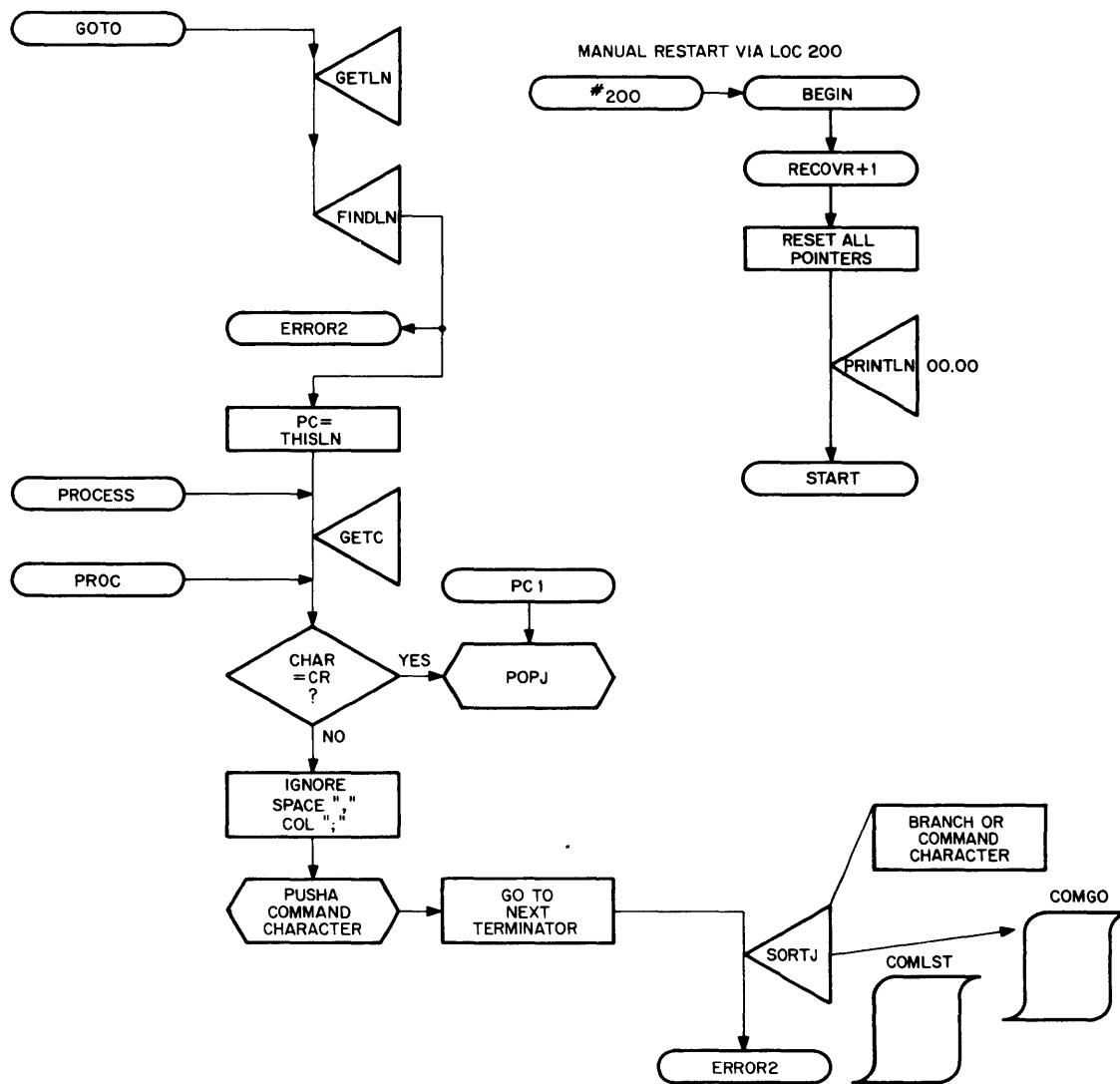


Figure G-3 Main Control and Transfer

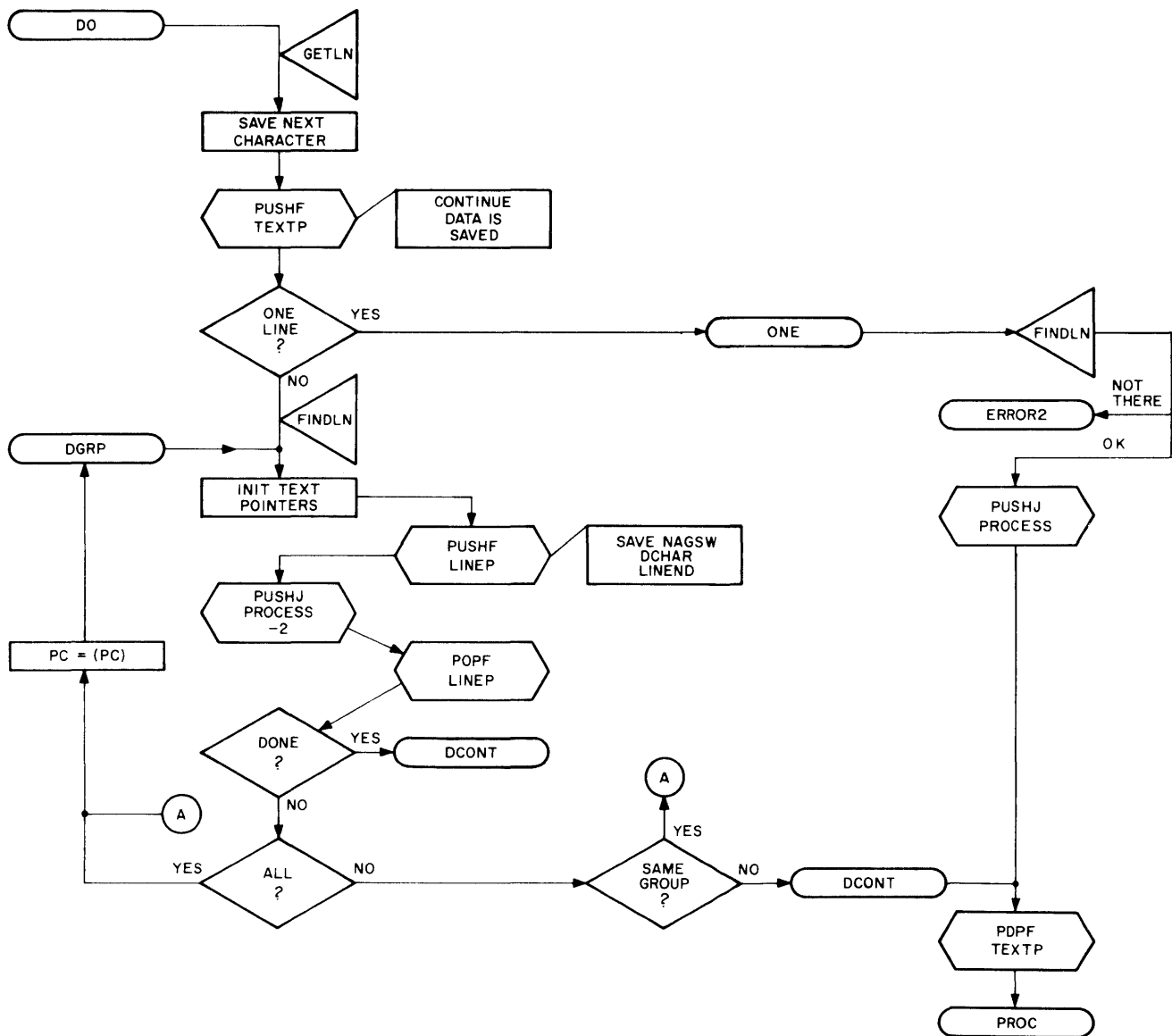


Figure G-4 DO Command

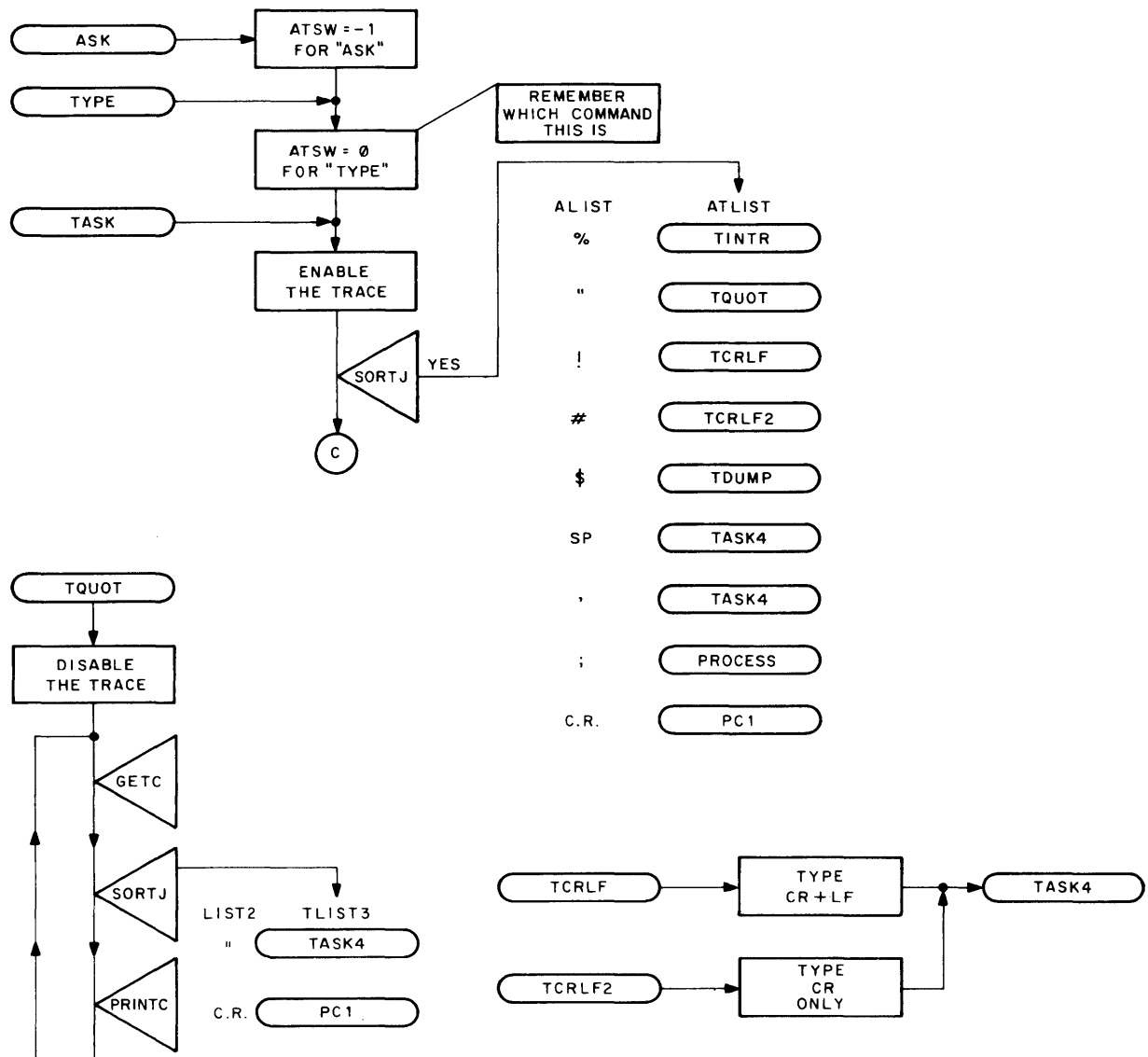


Figure G-5 (Sheet 1) Input/Output Commands

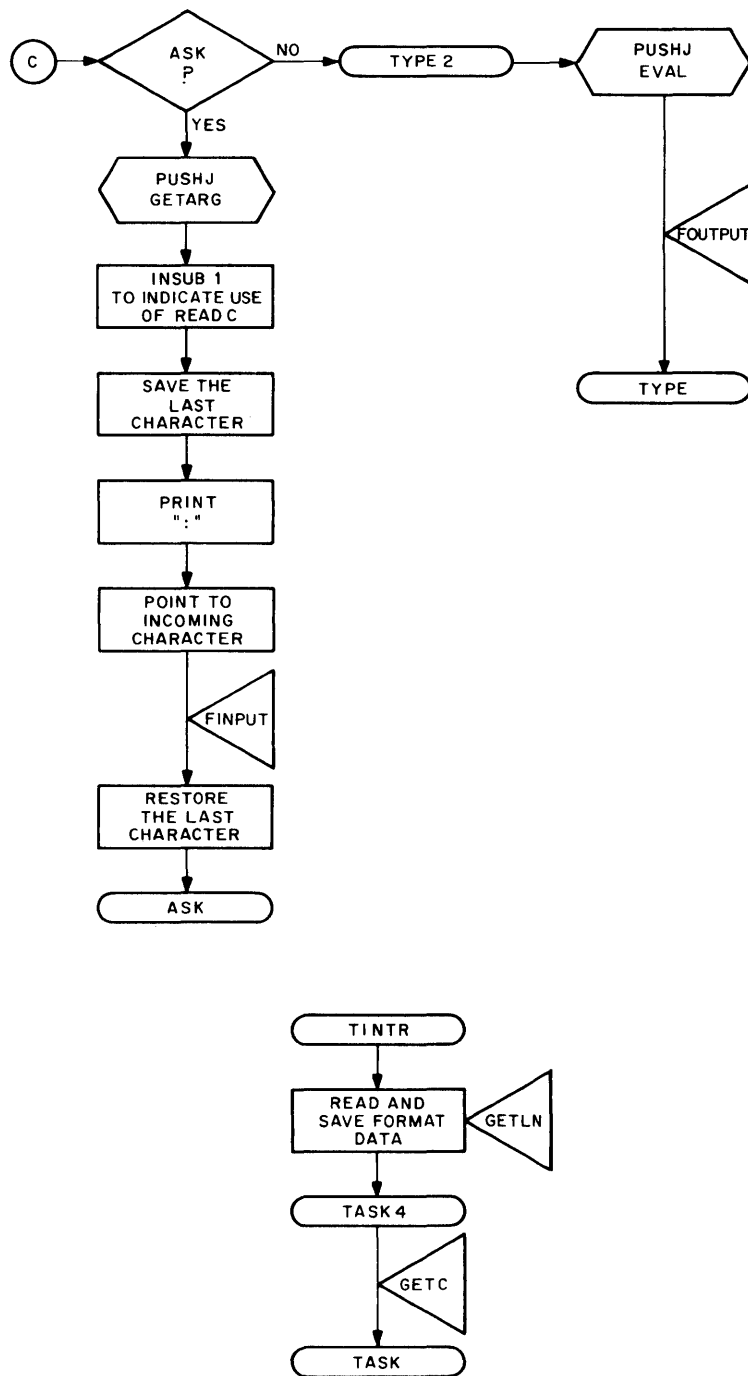


Figure G-5 (Sheet 2) Input/Output Commands



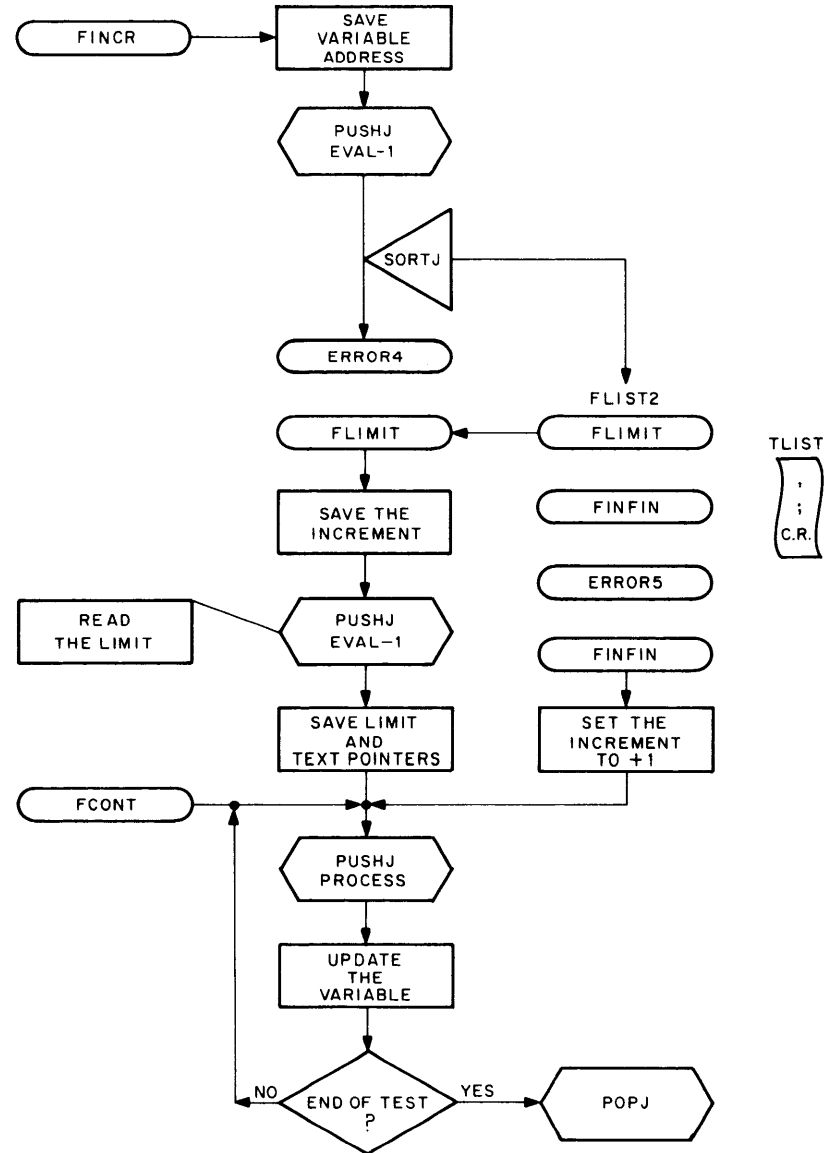
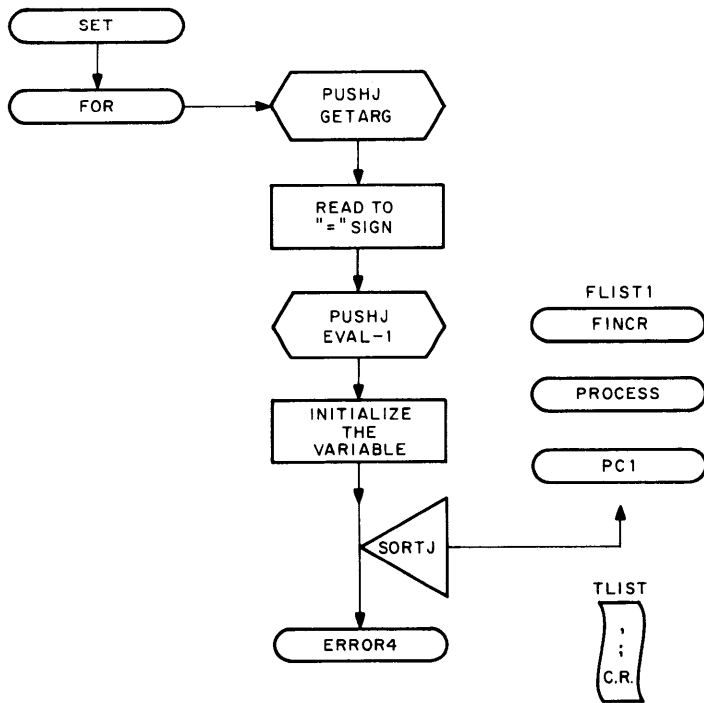


Figure G-6 Iteration Control

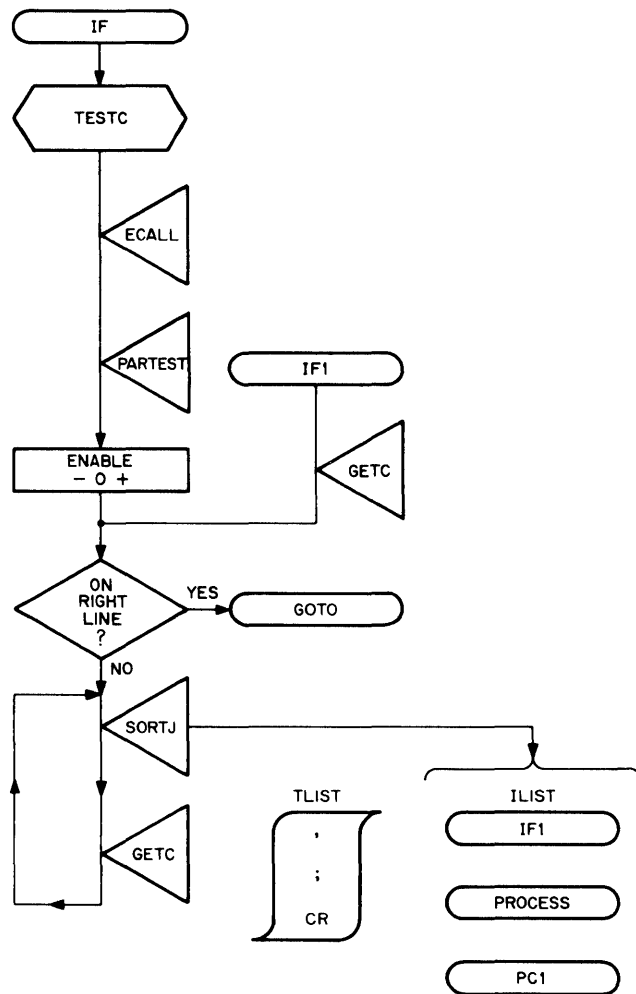


Figure G-7 Conditional Branch Command

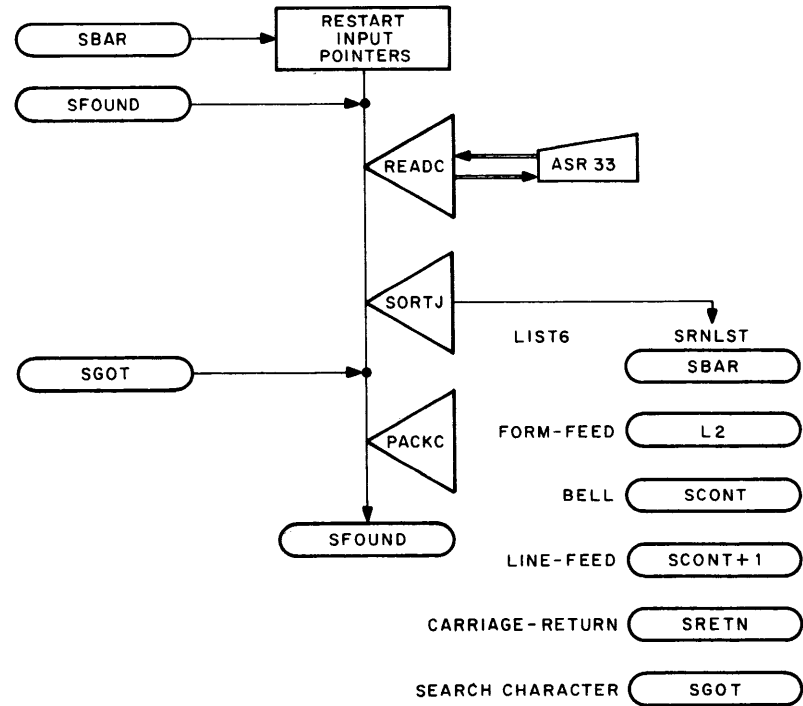
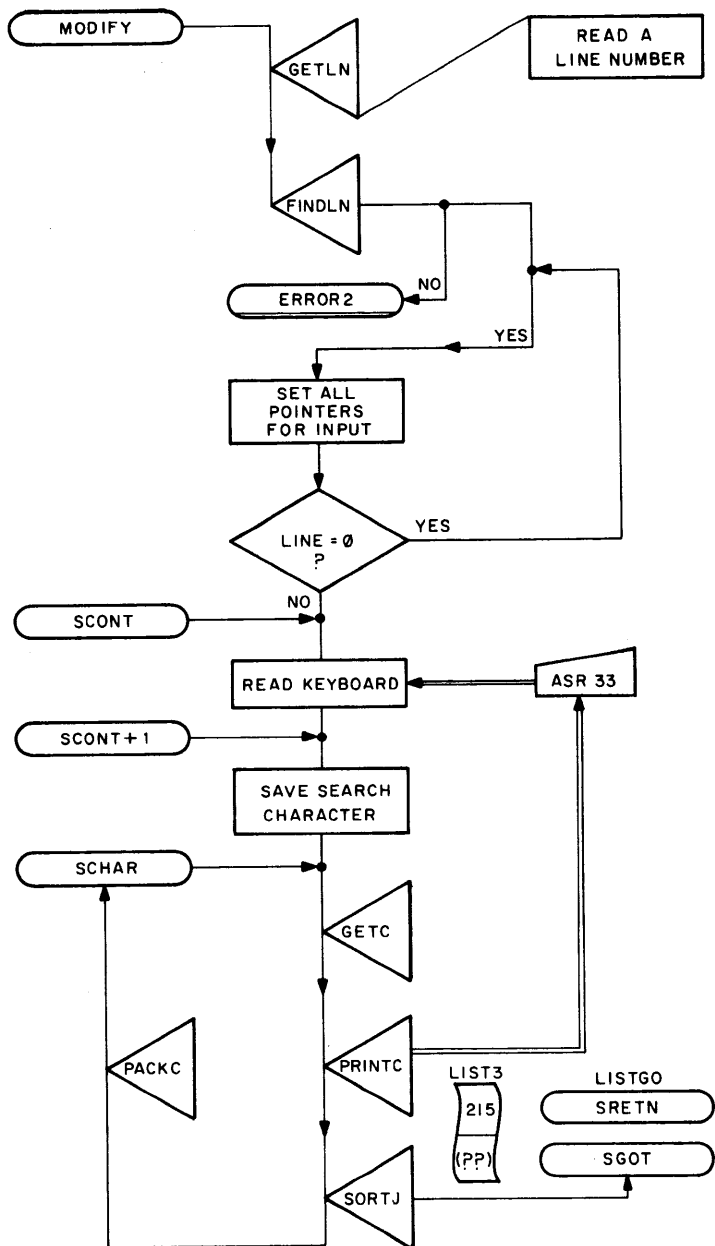


Figure G-8 Character Editing

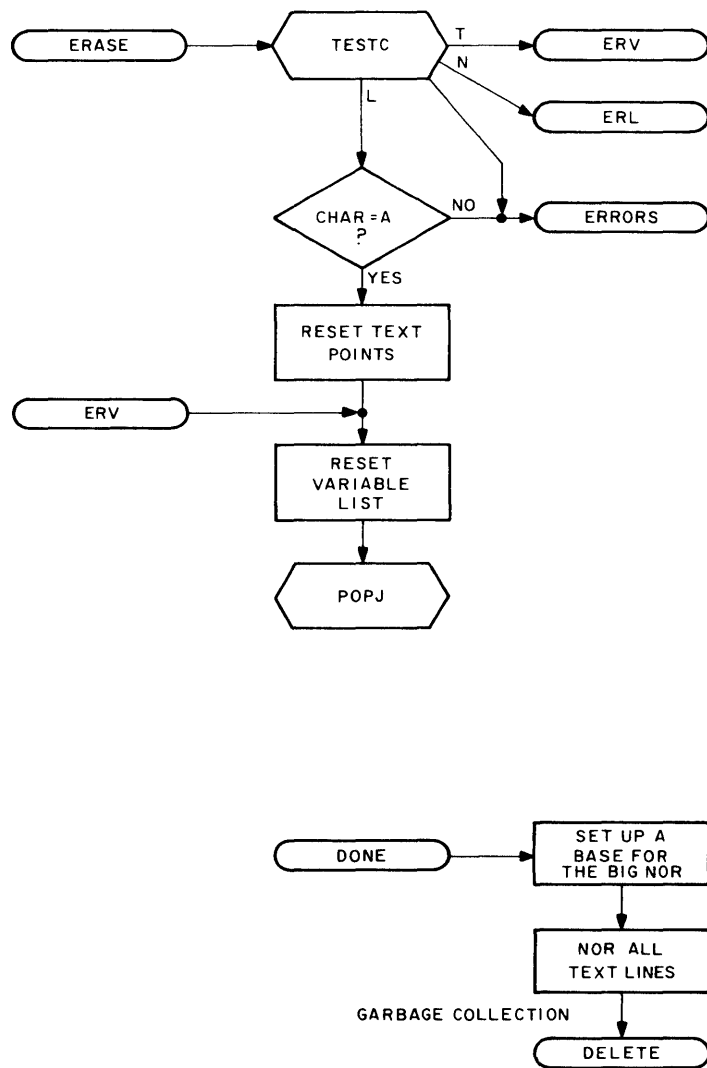


Figure G-9 (Sheet 1) ERASE and Delete

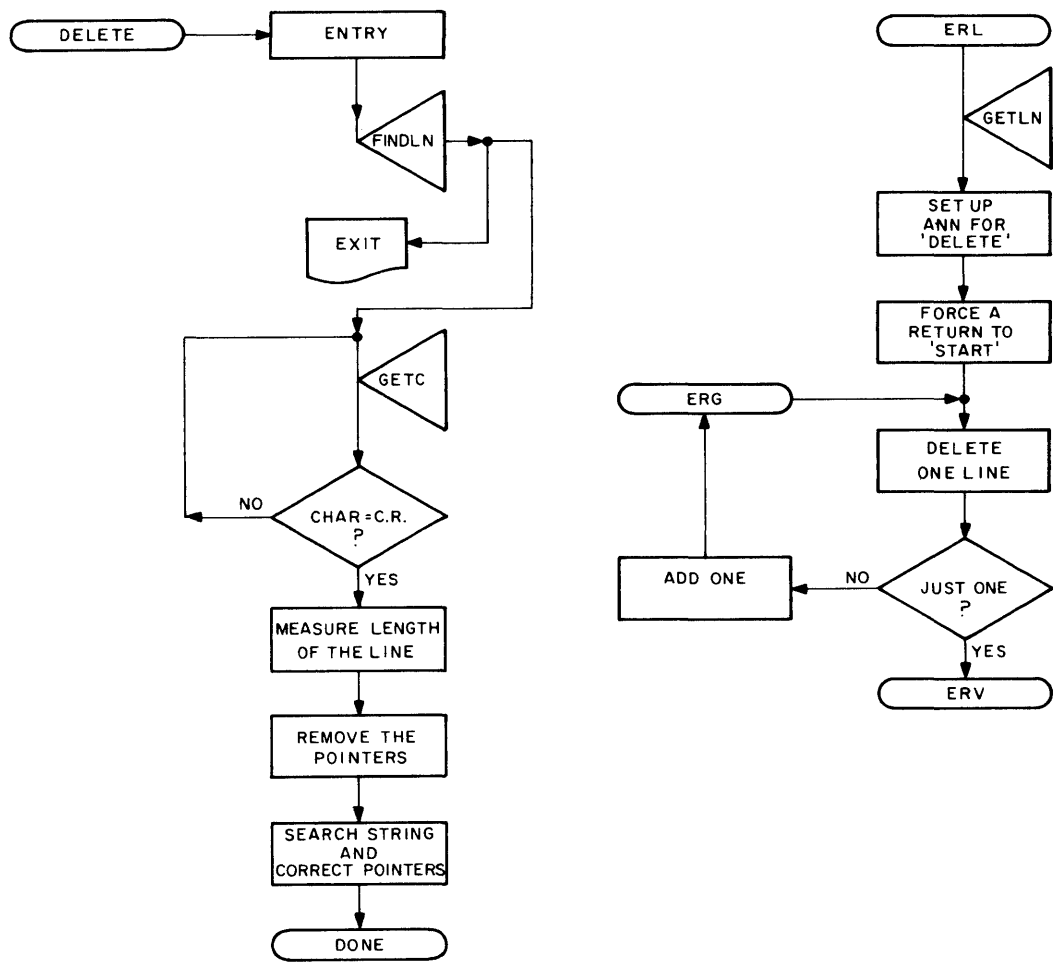


Figure G-9 (Sheet 2) ERASE and Delete

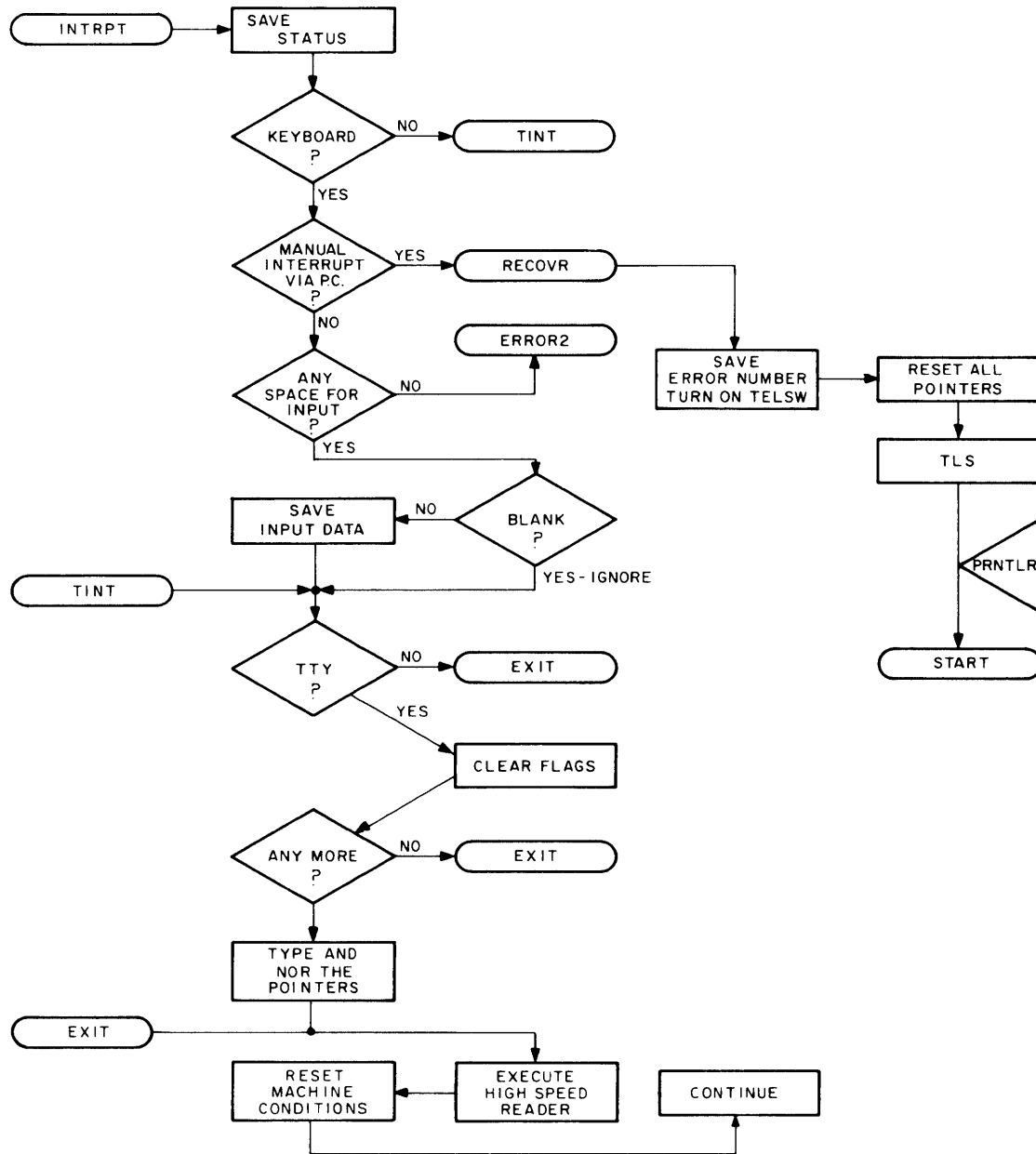


Figure G-10 (Sheet 1) Interrupt Handler

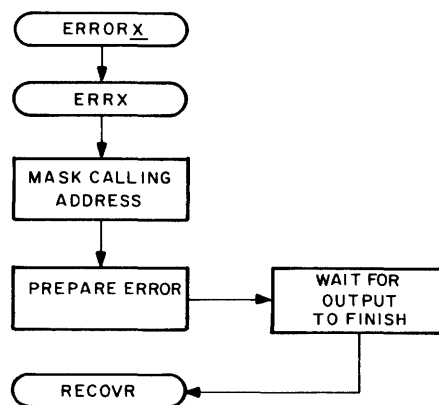
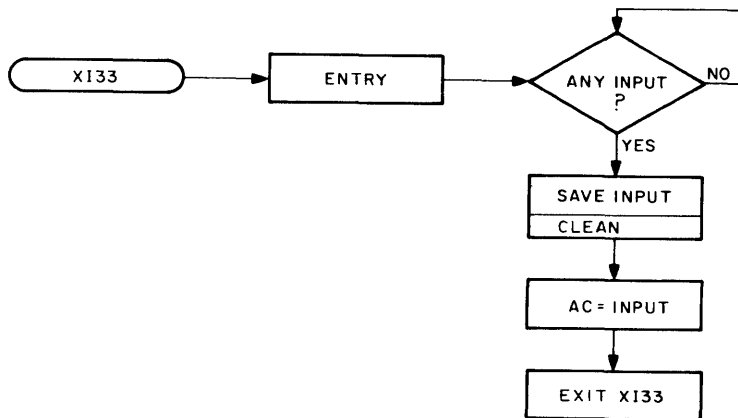
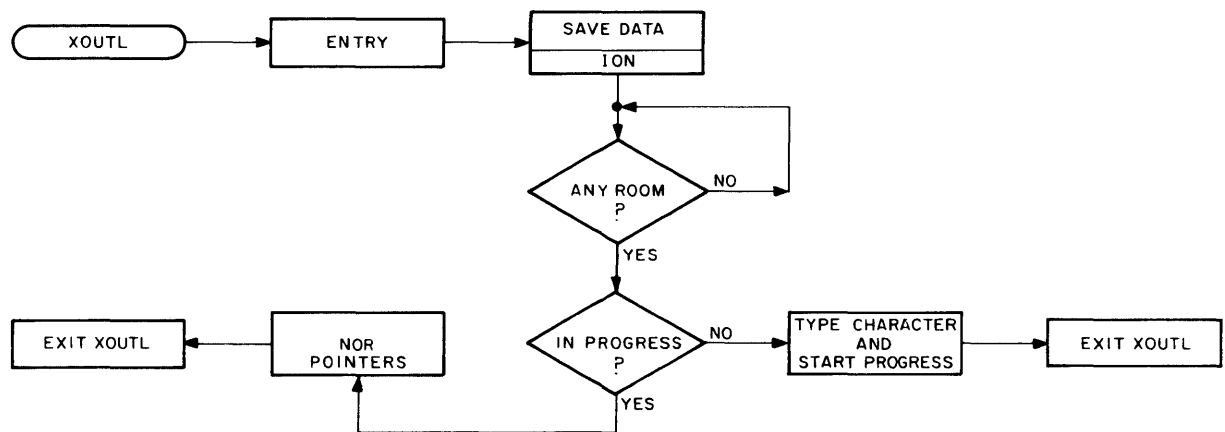


Figure G-10 (Sheet 2) Interrupt Handler

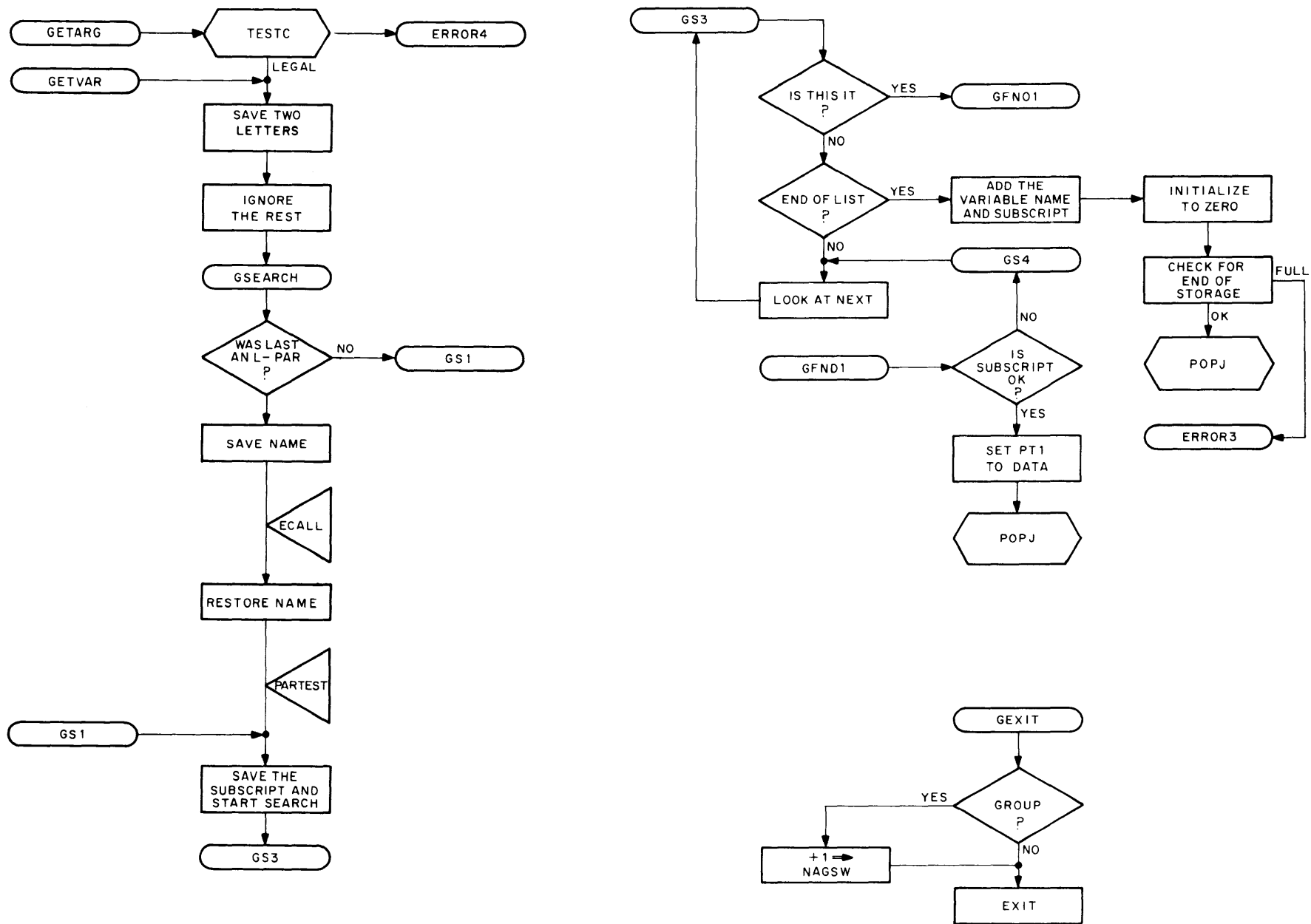


Figure G-11 Variable Look-up and Enter



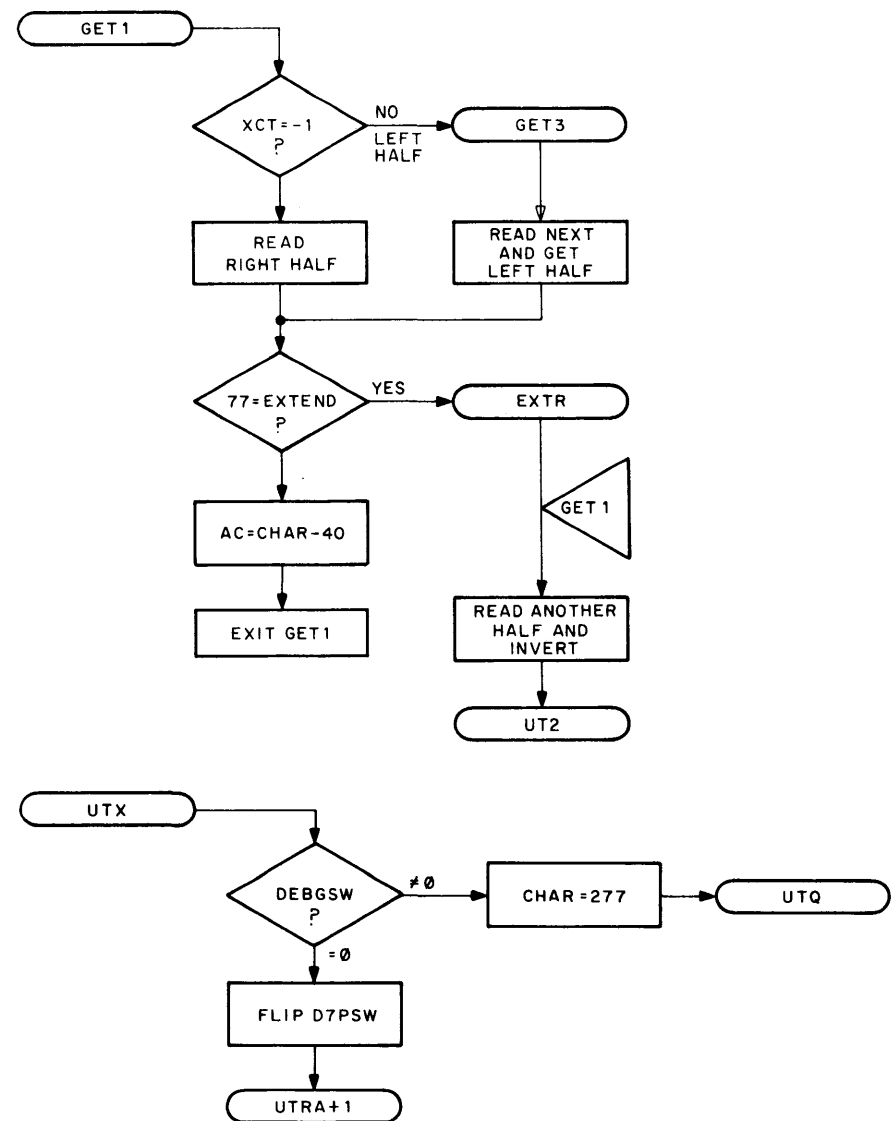
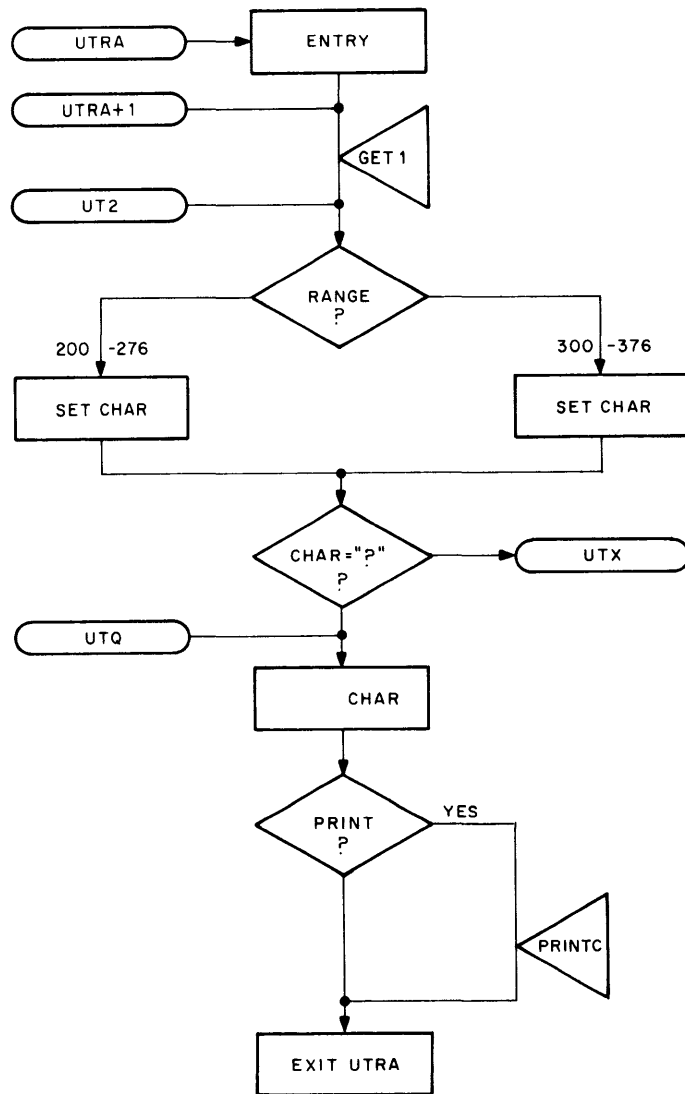


Figure G-12 Character Unpacking

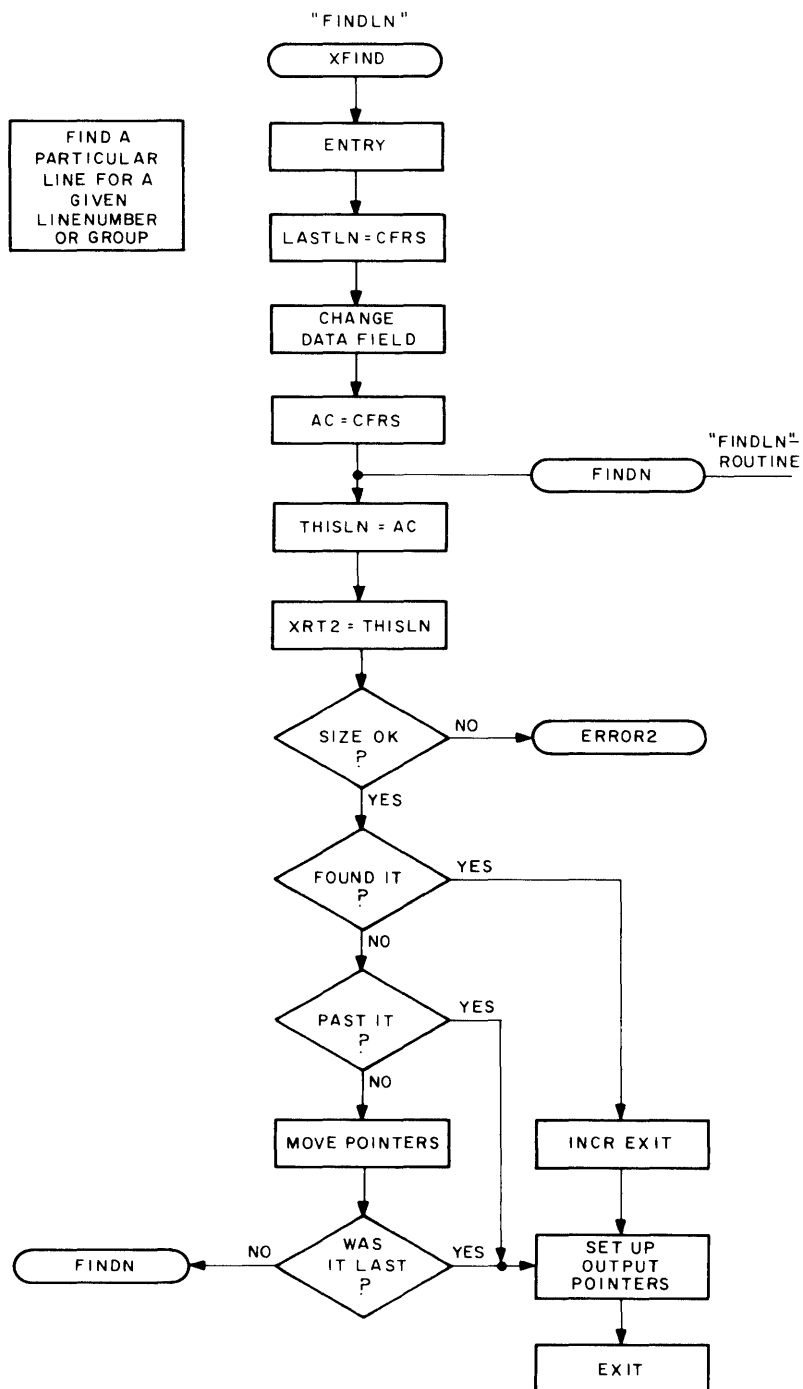


Figure G-13 "FINDLN" Routine

## READER'S COMMENTS

Digital Equipment Corporation maintains a continuous effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback -- your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability.

---

---

---

---

Did you find errors in this manual? Please explain, giving page numbers. \_\_\_\_\_

---

---

---

---

How can this manual be improved? \_\_\_\_\_

---

---

---

---

DEC also strives to keep its customers informed on current DEC software and publications. Thus, the following periodically distributed publications are available upon request. Please check the publication(s) desired.

☐ Digital Software News for the PDP-8 Family, contains current information on software problems, programming notes, new and revised software and manuals.

☐ PDP-8/I Software Manual Update, contains addenda/errata sheets for updating software manuals.

☐ PDP-8/I User's Bookshelf, contains a bibliography of current and forthcoming software manuals.

Please describe your position. \_\_\_\_\_

Name \_\_\_\_\_ Organization \_\_\_\_\_

Street \_\_\_\_\_ Department \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_ Zip or Country \_\_\_\_\_

..... Fold Here .....

..... Do Not Tear - Fold Here and Staple .....

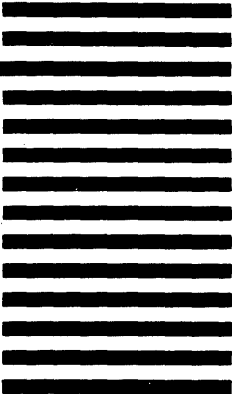
FIRST CLASS  
PERMIT NO. 33  
MAYNARD, MASS.

BUSINESS REPLY MAIL  
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

**digital**

Digital Equipment Corporation  
Software Information Services  
146 Main Street, Bldg. 3-5  
Maynard, Massachusetts 01754



**Digital Equipment Corporation**  
**Maynard, Massachusetts**

