DECUS
PROGRAM LIBRARY

DECUS NO.	FOCAL8-227a
TITLE	FOCL/F - AN EXTENDED VERSION OF 8K FOCAL/69
AUTHOR	D. E. Wrege
COMPANY	Georgia Institute of Technology Atlanta, Georgia
DATE	May 1, 1973
SOURCE LANGUAGE	PAL

FOCL/F - AN EXTENDED VERSION OF 8K FOCAL/69

DECUS Program Library Write-up

DECUS NO. FOCAL8-227a

ABSTRACT

FOCL/F is a version the FOCAL* language which implements several extensions for increased power and versatility. Among these are: user defined functions, user defined interrupt service, execution of machine language instructions from FOCAL*, arrayed variables, PS/8 compatibility, line number computation, extended commands, ASCII character commands, links for ease of addition of user assembly-code subroutines, new TTY-high speed reader control commands. A PS/8 overlay is available for file handling from FOCAL*, which permits device independent program calling/saving, variable files, and ASCII files. FOCL/F version 12/1/72 is closely compatible with FOCAL-10, the newly released implementation of FOCL/F on the DECSYSTEM-10 by Rob Warnock III at the Chemistry Department of Emory University. This document includes additions to the earlier version dated 6/1/72.

Introduction

FOCL/F was developed to fit the needs of a multi-disciplinary user environment which includes extensive scientific and engineering calculations, "on-line" experiment data-acquisition, control and analysis, and general educational assistance. It is intended to be an extended version of 8K FOCAL/69* with emphasis upon improved power of the language.

The FOCAL language provides some unique features in the small machine environment. Of particular significance is the ease of program writing and debugging via the resident editor with its MODIFY command, and the ability to execute immediately, from the command mode, any line or group as the program is being written. The recursive nature of the language, the ability to call any line or group as a subroutine (even if the line or group is part of another subroutine), the ability to abbreviate commands, and the efficiency of text storage, permit larger and more significant programs to be run in minimum machines without bulk storage. The interpretative-interactive system permits decision making without coding elaborate decision algorithms. Program development time is significantly reduced over compiled languages. Programs can be readily executed in FOCAL/8K* whose magnitude is comparable to large machine problems. Although the interpreter is slower than compiled or assembled code in many cases, the ease of program development and the consequent time reduction makes the slower runtime a cost-effective trade-off in the multi-user, multi-problem environment, and an extremely effective language to use for development.

In order to provide increased power and to reduce variable storage and searching time, a significant feature has been incorporated into FOCL/F. This is the facility to call lines or groups of FOCL/F coding as user-defined functions. These functions may be called recursively, passing multiple arguments as "dummy" variables. The dummy variables are not assigned permanent storage locations, resulting in core savings.

Within these user-defined functions the variables created are listed separately and the variable search is confined to this list rather than searching the entire (global) list. Global variables are also accessible however. There are ten user-defined functions available, FØ-F9. The functions are defined by a "LET" command, e.g. LET F3 = (Line or group number). The recursivity of the functions so defined is an extremely powerful feature since it allows solution of transcendental functions or other non-analytic functions in minimum coding. Complex variables, matrix algebra, and other functions may be simply coded via the user-defined functions.

FOCL/F has arrayed variables with array-address computation so that only three words rather than five are required per variable. The computation of the address permits direct access rather than searching the variable list, to improve speed. Unlike 8K-FOCAL/69*, variables and text are stored in contiguous core so that the user may take advantage of the variable-text trade-off to provide more efficient core utilization. For small programs a larger number of variables can be used than in other 8K versions. Unlike other FOCALs with variable-text trade-offs the use of the editor MODIFY command does not erase the variables. This permits run-time modifications to be made without having to rerun the entire program.

A second major feature, carried over from the earlier FOCL/S (and FOCAL+ by D. Dyment) is the ability to execute any machine-language instruction from within FOCL/F via FX(2, ARG1, ARG2). This causes the execution of machine language instruction, ARG1, with ARG2 in the AC. The FX(2, X, Y) function is particularly useful in controlling I/O devices from FOCL/F. The BRANCH (or BR) command is a con-

*FOCAL is a registered trademark of the Digital Equipment Corporation.

+This work was an extensive recoding of FOCAL/69 by D. E. Wrege and supported in part by Institutional funds from the State of Georgia.

ditional "GOTO" to test if the last FX(2,xx) caused a "skip" to occur. The FX(1, xx) is similar to the FOCAL+ "FCOR" command. Other FX commands added include FX(3, X, Y) (after J. Alderman) for the logical "AND" on the low order 12 bits of ARG1 and ARG2; FX(8, ARG) and FX(10, ARG) are used for octal to decimal and vice-versa conversion. An improved FRAN function has also been included (see DECUS FOCAL-1).

User-defined interrupt handling has been added to permit insertion of appropriate user device codes in the interrupt chain, and execution of FOCL/F subroutines on the appropriate condition if required. With this addition, SKIP tests and FLAG "Clears" are executed at machine language speeds, permitting the addition of environment-specific devices without the necessity of adding machine language patches.

Eight-bit character handling capability has been provided via FIN(arg) and FOUT(arg) functions in a manner similar to the OMSI version[1]. Both ASCII and non-standard codes (e.g. IBM-Selectric, etc.) may be converted in FOCL/F for output to ASCII devices, or for use within the program.

In order to improve the versatility of FOCL/F, links have been provided for user written commands and functions, and the user commands may have two letter abbreviations to avoid conflicts, and to provide more sensible mnemonics. Two FX functions, four normal functions (FNEW, FCOM, FADC, FN) and eight additional commands are provided for those who wish to write their own machine language subroutines for special applications. The floating point package has been modified so that it may be used from any field for ease of subroutine coding.

Other features of FOCL/F include: line number computation, a line continuation character providing unlimited line length for text, conditional subroutine call via the "ON" command, MOVE command to move text from one line-number to another, decrementing FOR loops, common array storage (using three words instead of five per indexed variable), variable-text trade-off, and a more comprehensive list of error messages.

In order to provide PS/8 compatible commands, some changes have been made. These include: CNTL/C (Return to PS/8 monitor) with CNTL/O being the new interrupt character; "E T" erases text; "E V" erases variables; "E A" erases text and variables; and the "toggling" TTY/high-speed-reader command "*" has been deleted and replaced by LIBRARY INPUT and OUTPUT commands, with the default condition being the TTY. Data may be input from the low-speed-reader via ASK command if no ECHO is required. However, with minor exceptions, FOCAL/69 programs will run under FOCL/F.

A separate version is available for the 8/E which sets the TTY reader "run" only when FOCL/F requests a character, thus permitting low-speed tape data input via "ASK" and permitting "batch-mode" processing via low-speed tape. A second feature is that the MQ register displays the contents of the core address specified by the switch register. Undefined interrupts are cleared at the end of the interrupt service routine.

In use, FOCL/F has proven to be a powerful extension of the extremely versatile FOCAL language.

[1] Schneider, D. and Smith, B., "PS/8 FOCAL, 1971," OMSI 4.1-D, Oregon Museum of Science and Industry, Portland, Oregon (1971).

Command Variations

The modified interpreter in FOCL/F is not completely compatible with the standard DEC supplied FOCAL*. The motivation for these changes was based on the need to add some PS/8 compatible commands for human engineering. Most of the following incompatibilities reflect either a tendancy toward PS/8 standardization or a clean-up of syntactical forms which are believed to be purely historical in origin. The changes that are made effect mostly the operation from the command mode, hence with minor exceptions programs that run in FOCAL/69 will run in FOCL/F.

Summary of Incompatibilities

#	is the new command mode acknowledge character (instead of "*")
CNT/O	is the new interrupt character (instead of CNT/C) return to Focal Command Mode
CNT/C	Returns to PS/8 (or other monitor) via an effective JMP 7600. [implemented in 1/11/72 version][2] .
CNT/U	Replaces back arrow [for rubbing out lines]. However back arrow still works for "ASK" input (as well as CNT/U). (Up arrow) (U) (CR) (LF) is echoed upon receipt of CNT/U.
*	The high-speed reader command has been replaced by a more comprehensive set of library commands. This incompatibility may require some old program modification.
E T	"ERASE TEXT" erases text.
E V	"ERASE VARIABLE" erases variables [also "E"].
E A	"ERASE ALL" erases both text and variables.
FADC	UNDEFINED. Experience has shown that most users wish to code their own FADC handler.
Note:	In FOCL/F commands may be abbreviated to a single letter. However if they are spelled out the first two letters must be correct, e.g. COMMT and C are legal for "COMMENT" but "C-" or CALC are illegal.

FOCL/F INPUT/OUTPUT Specifications

As previously mentioned the "*" or high-speed reader command has been deleted. The primary reason for this was the user uncertainty about the state of the complementing switch during program execution. To circumvent this problem, complete control over both the high-speed reader and high-speed punch has been implemented (as well as control over the TTY). A one page high-speed -reader buffer is included to reduce the wear on reader clutches. Conceptually there are three devices: (1) the input device which may be the TTY or the High-Speed Punch, (2) the output device which may be the TTY or the H.S. Reader or both, and (3) the echo device which may be the TTY or the H.S. Reader or both or neither. "#", ":", "(up arrow)U", and other acknowledge characters go only to the echo-device. Errors or CNT/O reset the devices to TTY. Also when the HSR runs out of tape and at the conclusion of the WRITE command the devices are reset (input dev. and outdev. respectively) to the teletype the commands are:

```
LIBRARY INPUT, TELETYPE     (L I,T)
LIBRARY INPUT, HIGHSPEED    (L I,H)
```

[2] The user may insert a patch here to his monitor system or JMP I to 200 to return to FOCL/F command mode as in FOCAL/69. Viz.

```
7600    5601
7601    0200
```

```
LIBRARY OUTPUT, TELETYPE    (L O,T)
LIBRARY OUTPUT, HIGHSPEED   (L O,H)
LIBRARY OUTPUT, BOTH        (L O,B)

LIBRARY ECHO, TELETYPE      (L E,T)
LIBRARY ECHO, HIGHSPEED     (L E,H)
LIBRARY ECHO, BOTH          (L E,B)
LIBRARY ECHO, NONE          (L E,N)
```

When input is from the HSR, a one page buffer is read in, and characters are fetched from this buffer. Input may be switched to TTY for a while and then back to HIGHSPEED with input from the buffer resuming where it is left off. In order to ignore the rest of the HSR buffer (e.g. when there has been a mistake and one wishes to start over from the beginning of a tape) an addition command has been implemented:

```
LIBRARY INPUT, REWIND       (L I,R)
```

Examples:

To input a program from the HSR and list:

```
#L I,H(ret.)
```

To input from the HSR and not echo program:

```
#L E,N;L I,H(ret.)
```

To read a data point from tape and then ask for response from the TTY followed by another data point:

```
#L I,H;A D(J);L I,T;A FLG, !;L I,H;A D(J+L);L I,T
```

NOTE that the input from the HSR will be echoed.

By turning off the ECHO to the teletype, data may be input via low-speed paper-tape via the ASK command. The user must input the entire list of data on the tape, and perform no significant calculations during input to prevent loss of data. The high-speed inputs and the 8/e version are not subject to this limitation. The user should insert blank leader or "@" at the end of the files to be input on the low speed reader to provide time to turn off the reader.

User Defined Functions

The most signigicant addition to the FOCAL language that FOCL/F has to offer is User Defined Functions. Conceptually, a user defined function is a method which allows the FOCL/F programmer to call a line or a group of FOCAL programming as a function. There are 1Ø user defined functions available: FØ, F1, Fx,...,F9. The functions are defined via the "LET" command, e.g.

```
LET F3 = [line of group number].
```

Henceforth, whenever F3 is called the line or group specified in the "LET" command is executed. To deassign a function: LET F3 = Ø.

In addition there is a general purpose user defined function, F(line#,ARGS), where the first argument is the line or group to be executed.

The rules for writing the function are:

1. Arguments will be passed to the function by value in the following order A',B',C',D',E',F',.....
2. The value of the function will be the value of the variable Z' upon completion of the line or group.
3. All primed variables will be local to the function.
4. All newly created non-primed variables will be local with the exeqption of functions called by that function.
5. All previously defined non-primed variables are global.
6. The function may be called recursively to any depth (limited only by the size of the stack [Push-down list]).

Note that F' is a legal variable name in FOCL/F.

Example:

To write a function, say F7(ARG1,ARG2), which will raise ARG1 to the ARG2 power:

```
#LET F7 = 3Ø.1
#3Ø.1Ø SET Z' = REXP(B'*FLOG(A'))
```

When F7 is used in the above example line 3Ø.1 is entered with A' = ARG1 and B' = ARG2. After 3Ø.1 is completed with no errors the function F7 takes the value of Z'. Hence, F7(2,Ø.5) will have the value 1.414. An equivalent call to this function is F(30.1,2,Ø.5) using the general purpose user defined function.

User defined functions may be used recursively, arguments may be any arithmetic expression, and the variables A', B', C',... are deleted upon successful completion of the function evaluation. If any error occurs or CNT/O is struck while in the function these variables will not be deleted. Note that any other variables that were created for the first time within the function are also deleted. Variable search for the primed variables is confined to the prime list rather than the entire list. This feature can result in considerable time saving for complex programs.

The recursivity of these functions is an extremely powerful feature of the user defined functions since it allows solution of transcendental functions or other normally non-analytic functions. For example, suppose that we have a function

```
Y = F1(X)+C*FCOS(Y)+D*FATN(Y/2)
```

where F1 is some defined function of X (evaluatable) and the last two terms are known to be correction terms much smaller than F1(X). We may define a function say FØ(X,ARG) where ARG is a guess at the value of FØ (e.g. F1(X)). Now we may code FØ to iteratively approximate itself as follows:

```
LET FØ = 1Ø
10.1 SET Z' = F1(A')+C*FCOS(B')+D(FATN(B'/2)
10.2 IF (FABS(Z'*.001)-FABS(Z' -B'))10.3;RETURN
10.3 SET Z' = FØ(A',Z')
```

This function will iterate on itself until a .1% value is obtained. Note how simple this iterative function is with recursive subroutines.[3]

User Defined Interrupt Service

In the past, interrupt handlers for non-standard devices have been patched into the interrupt service routine by some users via machine language overlays. FOCL/F has the additional feature of allowing the addition of up to three user devices to the interrupt chain, directly from the high level language. Interrupt service for these devices may be programmed, to a limited extent, in the high level language. This interrupt handling capacity is suitable for most non rate-limited devices (e.g. PT-Ø8's, HSR, HSP, etc.).

The device IOT's are inserted into the interrupt skip chain via the function

```
FX(4,Device#,SKIP IOT, CLEAR IOT)[4]
```

where Device # = 1, 2, or 3

SKIP IOT = IOT instruction causing PC skip

CLEAR IOT = IOT instruction causing the device flag to be cleared

The SKIP IOT and CLEAR IOT are in decimal

[3] "To iterate is beautiful, to recurse is devine," Anonymous

[4] If using PS8 Overlay see "Hints and Kinks."

(see FX(10,...)for octal to decimal conversion). FX(2,X,Y) is used for other IOT's required for device service.

The FX(4,S,Y,Z) function makes appropriate entries in the interrupt service routine:

```
IOTSKP1     /USER SKIP IOT DEVICE #1
JMP  .+5
IOTCLR1     /USER CLEAR IOT DEVICE #1
DCA UBUFR1  /IN CASE IOTCLR1 read something
ISZ SOFTF1  /USER SOFTWARE FLAG
JMP UDINT   /SET UP FOR USER DEFINED INTERRUPT
IOTSKP2     /CONTINUE DEVICE #2
```

The high level (FOCL/F) interrupt service routine is defined via the command

LIBRARY BREAK, line or group # (or L B,#).

Upon receipt of an interrupt from a device specified by the FX(4,...) function, the interrupt is handled directly in the skip chain at machine language speeds and a software user interrupt flag is set (not "SOFT1"). When the next carriage return is encountered in the currently running FOCL/F program, the following sequence of events take place.

1. The text pointers are saved for later restoration.
2. The line or group specified by the LIBRARY BREAK command is executed (if non-zero). This is effectively a "DO" command.
3. The text pointers are restored and the software user interrupt flag is cleared (not "SOFTF1"). FOCL/F continues running the interrupted main program.

Normally one would give the LIBRARY BREAK command before the FX(4,...) to prevent a missed interrupt.

FOCL/F has access to the interrupt service routine information through the following functions:

FX(5,ARG)[4]
where ARG is the user device number. Returns number of times, (=SOFTF), that device interrupted since the last FX(5,ARG). Note that FX(5,ARG) resets itself after 4096 interrupts.

FX(6,ARG)[5]
If ARG = Ø then FX(6,Ø) is the number of the lowest numbered device causing an interrupt since the last FX(5,...). FX(6,Ø) has a value >3, if there were none.
If ARG ≠ Ø then the UBUFR is returned for device ARG. Note that garbage will be returned if ARG <1 or ARG >3.

Note: The functions FX(4,...), FX(5,...), and FX(6,...) may not be used recursively among themselves, i.e. they may not be used in the arguments for FX(4,...), FX(5,...), or FX(6,...) functions.

While this facility will service most devices, those that are rate limited must be serviced via overlaying machine-language device handlers, which are added in a manner similar to that described in DECUS FOCAL 8-17. Note carefully that FOCL/F has been extensively recoded thus the symbol table is different from FOCAL.W and FOCAL/69.

Decrementing FOR Loops

A useful procedure has been added to allow use of negative constants or variables as arguments of the FOR statement

Negative steps are now allowed in the FOR, for example:

```
FOR I = 1, -3, -15; [command]
```

As before with positive increments, the program will execute the statements included in the FOR loop at least once, regardless of the limit values; that is FOR I = 1, 1, 0, and I = -1, -1, 0 will execute once.

Line Number Computation

Line numbers may be computed from any arithmetic expression. The rule is that the expression for a line number must not start with Ø-9. If the line number begins with a character Ø-9, the standard line number formation routine is called, which does not evaluate expressions. Hence, legal expressions are:

```
DO X
GOTO (3*FSQT(Y))
IF (EXPR) 2.1,X*2,J+2
```

Whereas the following are illegal:

```
GOTO 3*FSQT(Y)
IF (EXPR) 2.1,2*X,2+J
```

Note that the low order bits of the evaluation are truncated for line number computation, to obtain group numbers up to 32 and line numbers up to 99. If the expression should have a value greater than 32; there is an unpredictable truncation.

Line Continuataion Character

ALTMODE is a true line continuation character [unlike CNT/K in FOCL/S] with the equivalent status of a space. Hence ALTMODE may be used in an arithmetic expression to continue a long line to the next line. ALTMODE is saved internally as the single ASCII character ALTMODE, however whatever it is printed the sequence (ALTMODE) (ASCII 373) ($) (CR.) (LF.) (6 spaces) is typed. The reason for the ASCII 373 is so that on input [reading a program from paper tape] all of the garbage characters may be ignored.
Example:

```
1Ø.1Ø  DO 5; DO 6;↑SET X = FATN((Y1-Y2)/$
       (Y1 + Y2)) 2 + FLOG(Y1); TYPE 2 $
       ↑12
```

This line, when executed, will function exactly as if the ALTMODES were spaces, i.e. the last command on the line will respond with the typing of 4096.
This feature allows long subroutines to be coded as a single line and also permits a long series of commands to be executed from command mode.

This latter feature is extremely useful in experiment control since a variable sequence "BATCH" job list may be defined at run time. (i.e. Subprogram 1 may be run until some condition is met, then Subprogram 2, etc., permitting multi-user and/or multi-task operation, in an unattended manner).

"ON" Command

The "ON" command is similar to the "IF" command except a DO is performed instead of a GOTO. Hence, ON is a conditional subroutine call. Example:

```
#ON (EXPR)2.1,3,X;[more commands]
```

In this example if EXPR is less than Ø line 2.1 will be executed, if EXPR = Ø group 3 is executed, and if

EXPR is positive, line or group X will be executed. In all cases [more commands] will be executed if the line or group number is valid. As in the IF command, less than three line numbers (or groups) will make the ON command effectively a NOP if there is no line number corresponding to the condition of EXPR.

"MOVE Command

For some time FOCAL users have desired some method for changing line numbers. A flexible way of preforming this operation is implemented in the MOVE command. The syntactical form is:

MOVE Source Line #, Destination Line #

This command will move the contents of the source line to the destination line number MODIFYing as it goes. All control characters applicable to the MODIFY command are applicable to the MOVE command. (Users of KSR-35 teletypes or other machines with hardware FORM/FEED will be happy to know that FOCL/F does not echo CNTRL/Ls).

Arrays (non-PS/8 Version)

In addition to subscripted variables FOCL/F has true arrays similar to FOCL/S except that they are one-dimensioned arrayed variables only. Up to eight one-dimensional arrays may be specified. Arrayed variables make more effective utilization of core than normal FOCAL variables. Normal FOCAL variables use the index as an extension of the name, and require five locations, whereas the arrayed variables only require three locations each. Variable storage address computation is performed directly rather than searching the variable list, resulting in an increase of speed. These arrays occupy a "common" area of core which is specified by the "VARIABLE LIMITS" command. The user must reserve variable names for these arrays via the VARIABLE OPEN command and make sure that they do not overlap. No protection is provided for the user overwriting one array with another.

VARIABLE LIMITS, (NO. OF VARIABLES)
This command sets aside an area of core to be used by the arrayed variables.
VARIABLE OPEN, NAME, ORIGIN
Reserves the variable NAME for an arrayed variable starting at ORIGIN in the reserved variable storage area.
VARIABLE CLOSE, NAME
Releases NAME from arrayed variable status.
VARIABLES KILL
Releases all names from arrayed status.

For Example:

To create arrayed variables A(j), B(j), C(j) with 5Ø elements each - not overlapping

```
#V L,15Ø;C 15Ø VARIABLES
#V O,A,1
#V O,B,51
#V O,C,101
```

Note: It is up to the user to make sure that one array is not overwriting the other. For example in the above case A(51) will occupy the same storage location as B(1). FOCL/F will not flag this error.

Special Functions

FX(1,ARG1,ARG2)[4,5]
The core memory function. This function may be used in two distinct ways: FX(1,ARG1) takes as its value the contents of the memory location specified by ARG1 (which must be a decimal value in the range Ø-32767), and FX(1,ARG1,ARG2) which performs similarly, first depositing ARG2 in the memory location specified by ARG1, and finally taking the value ARG2. Thus the statement

SET X = FX(1, 12345,FX(1, 12345)+1)

would increment the contents of memory location 12345 (field 3, 0071(8)) and set "X" equal to the new value. All arguments are decimal. The FX(10,xx) and FX(8,xx) may be used to convert from octal to decimal.

FX(2,ARG1,ARG2)[4 5]
The execute function. This will execute the machine language instruction specified by ARG1 and ARG2 in the AC. ARG2 is assumed to be zero if omitted. Needless to say "JMP" instructions and the like should be executed with extreme caution. A simple example: SET Y = FX(2,3844) will set Y equal to the value of the switch register.

BRANCH[5] (note this command can only be abbreviated to "BR")
The branch command is a conditional "GOTO" command. This command functions in a manner identical to the GOTO command except that it will be a NOP if the last instruction caused a skip to occur (the last FX(2,xxx)) thus the presence or absence of a skip may be tested.

Note: Users of the FX(2,xx) command to control peripheral hardware must keep in mind that FOCL/F uses the interrupt system for I/O servicing; interrupts generated by additional devices may be handled via FX(4,)etc. (see above, Interrupt Service). PDP8/E owners are fortunate to be able to disable the interrupt on most peripherals, but others BEWARE!!!!

FX(3,ARG1,ARG2)
Performs logical AND between the low order 12 bits of ARG1 and ARG2. This should be useful for "bit banging" and examining status words of peripherals. (after original concepts by J. C. Alderman).

FX(4), FX(5), FX(6) Described above (Interrupt Service)

FX(8,ARG)
Converts ARG to its octal equivalent. This function knows about two fields only, hence, ARG must be in the range Ø-8191. Note that this function is not available in the PS/8 overlay version.

FX(1Ø,ARG)
Converts the octal number ARG into its decimal equivalent. Once again FX(1Ø,xx) only knows about two fields, i.e. ARG must be between Ø and 17777. Note that this function is not available in the PS/8 overlay version.

FRAN( )
FRAN is an improved random number generator which generates repeatable random numbers uniform on the interval Ø to 1 (non-inclusive of 1). It uses the generation formula

$X(N) = (2^{17} + 3)*X(N-1)$ modulo 36

with the lead 23 bits supplying the random number. (This algorithm is discussed in detail in DECUS FOCAL-1 by G. A. Griffith and P. T. Brady in DECUS 5-25).

[5] Patterned after FOCAL+ by D. Dyment.

FIN( )
Character input function fetches one 8-bit character from the current input device and converts it to the decimal equivalent. (See also FX(2,)).

FOUT(ARG)
Outputs, ARG, a decimal number, as an 8-bit binary character to current output device (ARG in the range $\emptyset$ to 225_{10})[1]

Tabulation Character (:)

A tabulation character has been included in TYPE OR ASK commands. The general form is a colon (:) followed by an arithmetic expression. This will cause spacing on the current line to the carriage location equal to the integer part of the given expression, e.g. TYPE :15,"*" will print a "*" in the 15th character location from the left margin. If the evaluated expression is missing, or has a value of zero, a normal tabulation will be preformed to the next multiple of eight (8) spaces. (As in the "EDITOR"). The "T#" command has been deleted since the equivalent is "T :1".

Command Terminators

All prior versions of FOCAL have required that all commands be terminated by a space, semicolon, or carriage return. FOCL/F has expanded the list of command word terminators to include those commonly legal in FORTRAN, and other "clumsy" languages. This list now includes
(% " ! $: , ; carriage return, space, and ALTMODE.
Hence the commands "IF(X)", T!, and the like are legal in FOCLF.

Miscellaneous

The text, variables, and push-down list are all resident in field $\emptyset$. Hence, with the new structure there is: (1) no limit to the length of command lines, (2) trade-off of core between text and variables (allowing more variables if the program is short), and (3) the ability to gain text or variable room at the expense of extended functions. The extended functions may be deleted via FX functions in the basic version. Later versions may restore the direct command mode deletion via the initial dialog. The current version has no initial dialog. In addition, for those who wish to write their own machine language subroutines two FX functions and four normal functions (FNEW, FCOM, FADC, FN) are available as well as eight commands. In addition the floating point package has been modified to know about fields - hence may be used any field.

FOCL/F on the PDP8/E

For those lucky owners of 8/E's there are several additional features available as a result of hardware niceties.

The teletype reader run is only set when FOCL/F needs a character. Thus low-speed input is feasible for data input or program input without someone standing there turning the reader on and off. Also "batch mode" processing is possible by putting commands on a paper tape and leaving it in the reader.

The MQ register is loaded with the contents of the field one core location specified by the switch register upon receipt of each carriage return in text or when waiting for teletype I/O. Of course, one may specify field 2 only. This feature is very handy for debugging while executing.

If the interrupt service routine cannot find an interrupt, a program-generated power clear is executed to clear the unidentified flag.

4-Word Overlay

There is available a four-word overlay for FOCL/F. Note that this does not work with the PS/8 Overlay.

Error Messages

The error messages are now in five octal-digit format and correspond to the field and core location of the error subroutine call. In addition, the list of error messages is larger, giving a more comprehensive set for easier debugging. For those users who wish to write their own machine language subroutines, it is simple to determine the corresponding error messages from the listing of their additional coding.

PS/8-OS/8 OVERLAY

There is available a PS/8 overlay which implements device independent program storage/retreval/calling, File variables, and ASCII I/O. Two page handlers are allowed. It should be noted that with the PS/8 overlay the size of the text area is reduced by about one half. However, this is not a serious limitation since programs may be called from files.

File Specification - ASCII I/O

Input and output files may be specified in FOCL/F via the commands:

```
INPUT  DEV:NAME.EX
OUTPUT DEV:NAME.EX
```

These commands will do all of the file lookup and creation for ASCII I/O. It should be noted that the abbreviations for these commands are "IN" and "OU" respectively and may not be abbreviated to a single letter. Once the input and output files have been specified, the use of these files is as in basic FOCL/F, i.e.:

```
LIBRARY INPUT, FILE       (L I,F)
LIBRARY INPUT, TELETYPE   (L I,T)

LIBRARY OUTPUT, FILE      (L D,F)
LIBRARY OUTPUT, TELETYPE  (L O,T)
LIBRARY OUTPUT, BOTH      (L O,B)

LIBRARY ECHO, TELETYPE    (L E,T)
LIBRARY ECHO, FILE        (L E,F)
LIBRARY ECHO, NONE        (L E,N)
```

The only difference between this and basic FOCL/F is the use of the word "FILE" instead of "HIGHSPEED". Note that the interrupt is turned off while accessing PS/8 devices.

All input and output is buffered with a single PS/8 block (256 words). Hence, the user need only wait for peripheral operation every 384 characters, long enough to read one block.

Whenever a L I,F command is given, the effect is to replace the TTY with the character stream in the file. Therefore programs or data may be input from any PS/8 device.

The user must specify when an output file is to be closed [made permanent] via:

LIBRARY OUTPUT, CLOSE.

Note: It is a restriction in PS/8 that only one output file may be active at a given time on a specific device. Hence, care must be taken to close any active output file before executing "OUTPUT", "VARIABLE MAKE", OR "PROGRAM" commands.

Example:

To ASK for ten numbers from the file DATA. FD on DTA1, followed by a variable from the TTY, followed by a Alphanumeric string (terminated by a carriage return) to be output to the file, RESULTS, on SYS:

```
10.1 INPUT DTA1:DATA.FD;L I,F:FOR J=1,10;A DA(J)
10.2 L I,T;A VARIABLE;OUTPUT SYS:RESULTS;L 0,F
10.3 S CHAR = FOUT(FIN( ));IF(CH-FX(10,215))10.3,
           10.4, 10.3
10.4; L 0,T
```

If an additional L I,F follows, input will resume from DATA.FD, after the 10th variable. The output file on the systems device is still open for further output.

To save a program (named PROG) as an ASCII File and recall it to run, starting at line 10.1, the following syntax is used: (Note alternate method below)

```
#OUTPUT DEV:PROG.PA
#LIBRARY OUTPUT,FILE
#WRITE ALL
#LIBRARY OUTPUT FILE;TYPE "GOTO 10.1",!
#LIBRARY OUTPUT,CLOSE
```

To read back this program then:

```
#INPUT DEV:PROG.PA
#LIBRARY INPUT,FILE;LIBRARY ECHO,NONE
```

These commands all may be issued from the command mode, or may be incorporated into FOCL/F Text.

PS/8 Program Storage

A comprehensive set of commands have been implemented to allow the saving and calling of FOCL/F programs saved in PS/8 files (as image files):

```
PROGRAM SAVE,   <DEV:>NAME.EX
PROGRAM GET,    <DEV:>NAME.EX
PROGRAM DELETE,<DEV:>NAME.EX
PROGRAM RUN,    <DEV:>NAME.EX<;COMMANDS>
PROGRAM CALL,   <DEV:>NAME.EX<;COMMANDS>
PROGRAM EXIT
```

For all of the program commands the default device is DSK: expressions in < > are optional, and the commands may be abbreviated to one or two letters, e.g. PROGRAM EXIT may be written as P E or PR E or P EX or PR EX.

The SAVE command saves the current program on device DEV with file name NAME.EX. It is recommended that the user use extentions .FP or .FL for saved programs, .FV for variable files, and .PA for ASCII I/O files. The extension .FL is the default extension for all files.

The GET command loads the program into the text buffer and exits to the FOCL/F command (#) mode.

The PROGRAM RUN command takes two forms: If the optional ";COMMANDS" is missing the entire program is run ("DO ALL"). If there are more commands following the PROGRAM RUN command they are passed to the called program as a command line (just as if typed in the command mode).

The PROGRAM CALL command functions similar to the RUN command except that the current program, called MAIN, is saved on scratch blocks for later restoration. When the called program has finished execution the calling program is restored and continues running from the point following ";COMMANDS". Upon restoration, the header line contains the current PS/8 date and the program name "#MAIN". Variables created in MAIN are global to the called program, but variables created in the called program are erased when MAIN is restored. Of course, the called program may change any global variables to pass information back to "MAIN". If the called program erases variables, then they are, of course, erased when MAIN is restored and variables created by the subroutine may now be global (depending on how many were created). It is a good rule to follow that called programs do not erase variables.

The "PROGRAM CALL" command is not recursive. If a called program calls another program it becomes "#MAIN" and eventually will terminate with an error message on completion. If a called program executes a "PROGRAM RUN" command, the main program can only be restored via a "PROGRAM EXIT" command.

The PROGRAM EXIT command causes a return to MAIN. Whenever a program is SAVED the header comment line is changed to reflect the file name and the current system date, e.g. P S,FLPROG will change the comment line to

```
C PS/8 FLPROG .FL MM/DD/YY.
```

This line is saved with the program so that whenever it is called the header line is changed to remind the user of the file name and the date created. An "ERASE TEXT" or "ERASE ALL" restores the name "FOCL/F" and upon restoration of a main program from a "PROGRAM CALL" command the name is "#MAIN".

It should be noted that it is impossible to reference a file with a null extension. The reason for this was that for both input and output file names the default extension is always assumed as .FL.

The PROGRAM DELETE command deletes files.

PS/8 Variable Storage

A set of "VARIABLE" commands have been implemented to allow the saving and calling of FOCL/F arrayed variables from "virtual core" PS/8 files. These commands are

```
VARIABLE MAKE, (length in blocks),DEV:NAME.EX

VARIABLE OPEN, VNAME, DEV:NAME.EX

VARIABLE CLOSE, VNAME

VARIABLE KILL
```

The "VARIABLE MAKE" command creates a permanent PS/8 file on DEV with the filename NAME.EX. The default device is DSK and default extension .FL. The file length is specified by the "(length in blocks)" which may be any arithmetic expression. There are 85 variables per block, Ø is a legal sub-

script.

The "VARIABLE OPEN" command reserves the variable name VN as an arrayed variable taken from file NAME.EX. There may be up to 8 arrayed variables open at any time. Since there are only two buffers, one-block each, core resident at any given time it is preferred to use these arrayed variables in something resembling numerical order to minimize loading and updating of blocks in core. Also since only two are resident at a given time a third variable must be loaded over one of the previous ones requiring rather slow access times for even sequential use of more than two of these arrayed variables. FOCL/F does make an attempt at efficient loading by only updating a block if a variable has been changed and by loading a new block into the not most recently accessed buffer.

The "VARIABLE CLOSE" command releases the variable name VNAME from reserved status and writes any final core resident block back into the file. This command should always be given prior to dismissing FOCL/F or the variable file may not be correct for future use.

The "VARIABLE KILL" command releases all variable names from reserved status without updating the currently open files.

Finally, since FOCL/F uses the system scratch blocks for an overlay when doing file lookups, it is not restartable. Therefore whenever the user types CNTL/C,FOCL/F will ask

```
REALLY?
```

and wait for the response "Y". Any other response will result in an error message, retaining FOCL/F.

Another general note: FOCL/F will allow two page device handlers. It will first attempt to load a one page handler and when that fails will try a two page handler.

NOTE: The PS/8 overlay to FOCL/F starts at location 02600 to execute once-only initialization code. After initialization, the normal restart is from location 00200 as in standard FOCAL*.

Thus the loading procedure is:

```
.R ABSLDR
*FOCLF, FOCFPS,FOCFUN=2600$
.SA SYS:FOCLF
```

The loading procedure for non-PS/8 FOCL/F is:

```
.R ABSLDR
*FOCLF
.SA SYS:FOCLF
```

Applications

In addition to generalized problem solving, FOCL/F may be used for testing and debugging non-standard hardware, such as interfaces in prototype development. Maintenance programs may be written with ease in FOCL/F. A major utility is to provide a means for "on-line" data acquisition and control, with all the advantages of a high level interactive language. The ease of programming, and the flexibility of the interactive mode, permit many different kinds of experimental applications to be run with liggle or no machine language coding. For example, FOCL/F has been used for data acquisition and control of a three-axis neutron diffractometer, reactor control rod reactivity calibration, control of optics experiments at the University of Wisconsin Storage Ring Facility, and digitization and analysis of optical and graphical images. By the addition of small machine language functions, rate-limited experiments may be run with rates up to 50 kHz. Higher rate data-break devices may be set-up directly from the high-level language.

Hints and Kinks

There are two ASCII codes for ALTMODE depending on the terminal. FOCL/F will accept both types on input, however it only echo's the 375(8) code. The only difficulty observable may be when searching for the 375 ALTMODE with the 376(8) character in a MODIFY or MOVE. To correct this for a terminal which uses 376:

```
S Z = FX(1,5682,254)
```

which is the equivalent of changing location 13062 from a 375 to a 376.

The extended functions overlay, FOCFUN, is only necessary when using the PS/8 overlay. As these functions are saved with the program in "PROGRAM" commands, it is possible to change available functions depending on the program called. This feature is intended to facilitate user additions of special purpose machine language functions. The functions which preform the FX(4,-),FX(5,-), and FX(6,-) are contained in the area with the extended functions and therefore they appear to be unavailable in the PS/8 version. They may be restored by the command

```
S Z = FX(1,6021,2470)+FX(1,6022,2473)+
      FX(1,6023,2476)
```

and removed via

```
S Z = FX(1,6021,1889)+FX(1,6022,1889)+
      FX(1,6023,1889).
```

It is recommended that these functions be not available for the uninitiated user.

Appendum: Documentation Update

The functioning of the IF and ON commands have been altered to increase their flexibility. If a line number is left out, e.g. two commas together, then the program will continue execution with the next statement following the IF or ON command. e.g.

```
IF (X), 4.3;  MORE COMMANDS
```

will only branch to line 4.3 if X=0. "More commands" will be executed if X≠0. The above discription will apply also if the evaluated line number is zero. e.g.

```
IF (X) Ø, 4.3;  MORE COMMANDS.
```

With this construction one may branch only if X is non zero, viz

```
IF (X)5.2, ,5.2;  T "X WAS ZERO"!
```

The "MODIFY" command has been modified so that if no line number is specified, the last line referenced will be modified. This is very useful for modifying lines after an error message results, or for multiple modifications of the same line, viz

```
12345@5.10
#M
5.1Ø S Z=1/Ø [line feed typed after 5.10]
#M
5.1Ø  S Z=1/XØ
```

The interrupt character CNTR/P causes the same results as CNTR/O. This break character was added to conform to PS/8 - OS/8 standards (see TECO).

A new command, "EXIT", is implemented in the PS/8 version to allow return to the keyborad monitor from within a program without the "REALLY?" question.

Refreshed display is available, in place of the extended functions for PDP-12 owners.

FOCLF ERR DIAGNOSTICS

VERS. 12/1/72

?00000 USER TYPED ↑O OR ↑P.
?00237 INPUT TO FAST FOR OUTPUT: TURN OF TAPE READER OCCASIONALLY.
?00646 ANSWER TO "REALLY?" (↑C) WAS NOT YES
?00656 TEXT OVERLAPPING VARIABLES IN "PROGRAM GET"
?00713 TEXT OVERLAPPING VARIABLES IN "PROGRAM RUN"
?01065 LOG OF NEGATIVE NUMBER REQUESTED.
?01102 HANDLER READ ERROR IN ASCII I/O INPUT.
?01207 NO ASCII OUTPUT "FILE" DEFINED.
?01260 ERROR LOADING HANDLER FOR ASCII I/O OUTPUT.
?01271 HANDLER ERROR WRITING ASCII I/O OUTPUT.
?01312 NO MORE ROOM FOR OUTPUT FILE IN ASCII I.O.
?01341 UNABLE TO CLOSE ASCII I.O. OUTPUT FILE.
?01361 ASCII I.O. OUTPUT FILE ALREADY OPEN.
?01406 IMAGINARY SQUARE ROOTS REQUESTED.
?01470 HANDLER ERROR RESTORING CALLING PROGRAM = TROUBLE.
?01613 DEVICE DOES NOT EXIST ON SYSTEM.
?01655 ":" ILLEGAL IN FILENAME SPEC.
?01666 ":" ILLEGAL IN FILENAME SPEC.
?01675 ":" ILLEGAL IN FILENAME SPEC.
?01676 DOUBLE "." IN FILENAME SPEC.
?01762 USER DEFINED INTERRUPT DEVICE NUMBER NEGATIVE.
?01767 USER DEFINED INTERRUPT DEVICE NUMBER G.T. 3.
?02000 UNDEFINED FIELD T FUNCTION
?02025 FILE DOES NOT EXIST ON DEV:
?02061 TENATIVE FILE ALREADY OPEN ON DEV. OR BAD NAME.
?02116 UNABLE TO CREATE SPECIFIED FILE.
?02134 UNABLE TO SAVE PROGRAM
?02146 UNABLE TO DELETE PROGRAM(NOT ON DEV.)
?02236 UNABLE TO CREATE ("MAKE") VARIABLE FILE.
?02405 HANDLER ERROR READING PROGRAM.
?02504 HANDLER ERROR READING PROGRAM.
?02511 HANDLER ERROR READING PROGRAM.
?02532 SYSTEM HANDLER FAILURE.
?02540 SYSTEM HANDLER FAILURE.
?03231 SUBSCRIPT TOO LARGE IN ARRAYED VAR.
?03306 ERROR WRITING VARIABLE FILE BLOCK.
?03333 ERROR READING VARIABLE FILE BLOCK.
?03500 UNABLE TO UPDATE VARIABLE FILE.
?05065 LOG OF NEGATIVE NUMBER OR 0 REQUESTED.
?05406 IMAGINARY SQUARE ROOTS REQUESTED.
?05750 ARGUMENTS MISSING.
?05762 USER DEFINED INTERRUPT DEVICE NUMBER NEGATIVE.
?05767 USER DEFINED INTERRUPT DEVICE NUMBER G.T. 3.
?06000 UNDEFINED FIELD ! FUNCTION
?10240 ILLEGAL LINE NUMBER ON INPUT.
>10311 GROUP NUMBER TOO LARGE;31.XX IS LARGEST ALLOWED.
?10333 DOUBLE PERIODS IN LINE NUMBER.
?10344 LINE NUMBER TOO LARGE: XX.99 IS LARGEST.
?10360 GROUP 0 IS ILLEGAL:0.XX
?10420 GROUP DOES NOT EXIST IN "DO" OR "FN".
?10444 LINE DOES NOT EXIST IN "DO" OR "FN".
?10477 PUSH DOWN LIST OVERFLOW; PROGRAM TOO LARGE.
?10572 DUMB FORMAT IN "FOR" OR "SET": C.R. TOO EARLY.
?10575 LINE NOT THERE IN "GOTO".
?10636 ILLEGAL COMMAND: MISSING COMMAND OR MIS-SPELLED COMMAND.
?11035 LEFT OF "=" IN ERROR: 'FOR' OR 'SET'
?11053 EXCESS RIGHT PAREN IN "FOR" OR "SET".
?11063 ILLEGAL TERMINATOR IN 'FOR'
?11214 COMMAND NOT IMPLEMENTED. EXPANDABLE COMMANDS.
?11215 COMMAND NOT IMPLEMENTED.
?11216 COMMAND NOT IMPLEMENTED.
?11217 COMMAND NOT IMPLEMENTED.
?11220 COMMAND NOT IMPLEMENTED.
?11221 COMMAND NOT IMPLEMENTED.
?11323 LINE NUMBER DOES NOT EXIST IN "MODIFY".
?11402 BAD ARGUEMENT IN 'FOR', 'SET', OR 'ASK'.
?11411 F ILLEGAL FOR FIRST CHARACTER OF VARIABLE.
?11525 VARIABLE OVERFLOW: TOO MANY VARIABLES OR TEXT.
?11625 MISSING OPERATOR IN EXPRESSION OR BAD FORMAT.
?11645 OPERATOR MISSING BEFORE PAREN.
?11745 OPERATOR MISSING BEFORE PAREN.
?11754 DOUBLE OPERATORS OR ILLEGAL FUNCTION NAME.
?12051 PARENS DO NOT MATCH.
?12543 PROGRAM TOO LARGE OR TOO MANY VARIABLES.
?13111 ILLEGAL "LIBRARY" COMMAND.
?13242 ILLEGAL "VARIABLE" COMMAND.
?13431 CONVERSION OF NON-OCTAL NUMBER REQUIRED
?13457 ARGUMENT MISSING IN FX(10) OR FX(11).
?13506 ARGUMENT MISSING IN FX(3,XX,XX).
?13513 ARGUMENTS MISSING IN FX(3,XX,XX).
?13532 MISSING ARGUMENTS.
?13533 MISSING ARGUMENTS.
?13534 MISSING ARGUMENTS.
?13541 FX FUNCTION NOT AVAILABLE
?13552 VARIABLE IS NOT ARRAYED VARIABLE.
?13554 NAME NOT RESERVED FOR ARRAYED VAR.
?13645 NO ARGUMENTS IN FX(2,X,X).
?13670 NO ARGUMENTS IN FX(1,X,X).
?14216 "FIN" FUNCTION NOT AVAILABLE THIS VERSION
?14217 "FOUT" FUNCTION NOT AVAILABLE THIS VERSION
?14243 ILLEGAL SYNTAX IN USER DEFINED FUNCTION.
?14244 ILLEGAL SYNTAX IN USER DEFINED FUNCTION.
?14255 "=" MISSING IN "LET FN=LN OR GRP".
?14275 GROUP 0 NOT ALLOWED IN USER DEFINED FUNCTION.
?14477 VARIABLE NAME ALREADY IN USE.
?14504 ALL VARIABLE NAMES IN USE (ONLY 8 ALLOWED).
?14552 ARRAY SUBSCRIPT TOO LOW.
?14556 ARRAY SUBSCRIPT TOO LARGE.
?14625 ARRAY VARIABLES OVERLAPPING TEXT. VARIABLE LIMITS ILLEGAL.
?15137 ILLEGAL "PROGRAM" COMMAND
?15143 PS/8 VARIABLES ILLEGAL IN "FOR" COMMAND
?15222 SYSTEM DEVICE HANDLER ERROR
?15241 CALLING PROGRAM DOES NOT EXIST
?15644 INPUT OVERFLOW ERROR: TOO MANY DIGITS.
?16526 TOO LARGE OR NEGATIVE EXPONENT.
?17110 DIVISION BY ZERO REQUESTED.
?17444 INVALID CHARACTER. (RETYPE LINE).
?17510 "LIBRARY INPUT,REWIND" NOT AVAILABLE THIS VERSION
?17511 "LIBRARY OUTPUT, CLOSE" NOT AVAILABLE IN THIS VERSION.
?17516 NO Y VALUE IN FDIS(X,Y).