DECUS
PROGRAM LIBRARY

DECUS NO.	12-80
TITLE	FOCAL-RT
AUTHOR	William Siegel and Keith Whittle University of Western Ontario London, Canada
COMPANY	Submitted by: Kenneth Ellson Digital Equipment Corporation Maynard, Massachusetts
DATE	June 1972
SOURCE LANGUAGE	FOCAL, DIAL

FOCAL-RT

USER'S GUIDE

June, 1972

Keith Whittle and William Siegel

Department of Psychology

The University of Western Ontario

London, Canada

Description: Modifications of FOCAL-12 that include device-independent chaining of FOCAL and assembly-language programs, computed GOTO and DO commands, new FRAN( ) function, FIN( ) and FOUT( ) to handle character strings in FOCAL files, subroutines for opening and closing FOCAL files within assembly-language programs, LPØ8 printer option, return-to-DIAL command, and expanded text buffer.

Hardware Required: 8K PDP-12, LINCtapes or disk

Software Required: DIAL-MS Monitor

ACKNOWLEDGMENTS

Many of the new features in FOCAL-RT were inspired by other software systems. For example, the FIN( ) and FOUT( ) instructions were modelled after those of PS/8 FOCAL, developed at the Oregon Museum of Science and Industry. The idea of scaling random numbers internally came from PSYCBL, a software system developed by Braida, Callahan and Herman at the Massachusetts Institute of Technology. Prof. J.A. Siegel contributed a number of suggestions for this Users' Manual.

FOCAL-RT USER'S GUIDE

June, 1972

I. INTRODUCTION

1.1 FOCAL-RT: A RESEARCH-ORIENTED FOCAL FOR THE PDP-12

FOCAL-RT is a modified version of FOCAL-12 that is specialized for real-time applications, such as laboratory experiments. Like FOCAL-12, FOCAL-RT may be used to program simple data-acquisition tasks, provided that timing is not too critical and data rates are low. However, the usefulness of FOCAL-RT does not end here. Because it is possible to interleave FOCAL-RT and assembly-language programs that share common data files, one can do the bulk of the programming of even very complex tasks in FOCAL.

In a typical laboratory experiment, one may input parameters for a day's session, set up sequences of trials, and randomize extraneous variables in FOCAL. All of the information necessary to run the session is listed sequentially in a FOCAL data file. A short assembly-language program to run the experiment may be called up with the FOCAL-RT LOAD BINARY command. The assembly-language program simply works through the FOCAL data file, controlling the machinery, collecting data, and timing. Since FOCAL is not core-resident during the data collection phase of an experiment, one is not restricted by the limitations of an interpreted language at this point. After the data have been collected, a chained FOCAL-RT program may be called up for data analysis. By using a combination of FOCAL and assembly-language, one may achieve significant savings in software development time over straight assembly-language programming.

As well as the chaining features described above, FOCAL-RT has a number of features that are useful for research applications. These include a new random number function, computed program branches and subroutine calls, and the ability to handle alphanumeric text in FOCAL data files.

1.2 SUGGESTED READING

(a) FOCAL-12 Programming Manual. Order no. DEC-12-AJAA-D from Program Library, Digital Equipment Corp., Maynard, Mass. 01754, U.S.A.

(b) Programming Languages, 1970, Ch. 11, FOCAL. Available from the DEC Program Library.

(c) Laboratory Computer Handbook, 1971. Available from the DEC Program Library.

(d) LAP6-DIAL Programmer's Reference Manual, DEC-12-SE2D-D. Available from the DEC Program Library.

(e) Siegel, W. Combining FOCAL and assembly language. Behavior Research Methods and Instrumentation, 1972, 4, 105-106.

(f) Siegel, W., & Whittle, K. Using FOCAL in research, Proceedings of the DECUS Symposium, Spring, 1972.

II. NEW COMMANDS

2.1 LOAD BINARY

L B, Name, Unit ↵

where "Name" refers to a self-starting binary program that is filed in the DIAL index, and "Unit" refers to the device on which the program is stored. Units Ø - 7 refer to LINCtape, and units 1Ø - 25 refer to RSØ8 or RK8 disks (cf., p. 5, FOCAL-12 Programming Manual).

L B, RUN, Ø ↵

will load and start a machine-language program named RUN that is located on LINCtape unit Ø.

The program specified by LOAD BINARY may start in either LINC-mode or 8-mode. At present, LOAD BINARY will load only into FIELD Ø, but it could be easily adapted to handle 8K of core. A flow chart outlining the operation of this command is found in Section 8.1.

2.2 LOCATE FILE

L F, Name, Unit 2

stores the unit, starting block, and length of a designated FOCAL data file in locations ØØØ1, ØØØ2, and ØØØ3 of FIELD 1. "Name" and "Unit" are as in 2.1.

LOCATE FILE is used when chained FOCAL and assembly-language programs share common data files. The parameters saved by LOCATE FILE are used by a subroutine named RD-WRI (Section 5.1) in order to access a FOCAL data file by name within an assembly-language program.

One may open or close a FOCAL file previously specified by a LOCATE FILE command simply by jumping to RD-WRI in the assembly-language program. This feature is especially useful if one moves data from one storage device to another, as the actual tape or disk operations are invisible to the user, and since it is not necessary to worry about the absolute addresses of the file on the different storage devices.

The operation of the LOCATE FILE command is flow-charted in Section 8.1.

NOTE: LOCATE FILE should only be used immediately before exiting from FOCAL via a LOAD BINARY COMMAND, since one of its effects is to scramble text displayed on the CRT using OUTPUT SCOPE. It doesn't affect the teletype operation, however.

```
2.1Ø  L F, DATA, Ø          /SAVE PARAMETERS OF
                            /FILE NAMED "DATA STORED
                            /ON UNIT Ø

2.2Ø  L B, RUN, 1           /LOAD ASSEMBLY-
                            /LANGUAGE PROGRAM
                            /FROM UNIT 1
```

2.3 LIBRARY EXIT

```
L E, 2
```

returns the user to the DIAL monitor.

2.4 COMPUTED GOTO

$$G\ X,L_1,L_2\ldots L_n\ \textcircled{2}$$

where X is any legal variable, and L_i is a program line number.

The computed GOTO command transfers program control to the Xth line listed in the command. Only the integer value of X is used by the computed GOTO.

```
1.1Ø  SET Y=3
1.2Ø  G Y, 4.1, 5.1, 6.1, 7.1
```

will cause a program branch to line 6.1.

As many line numbers as desired may be listed in a computed GOTO command provided that they can all be fit into a single line of FOCAL program text.

If a literal rather than a variable follows the GOTO, then the command will be interpreted as a standard FOCAL GOTO.

```
e.g.,  1.1Ø G 3.1
```

will cause a program branch to line 3.1.

With a computed GOTO command, if the integer value of the variable X is greater than the number of program lines listed in the command, then the command following the GOTO statement will be executed.

```
e.g.,  1.1Ø S X=1Ø
       1.2Ø G X, 2.1Ø, 3.1Ø
       1.3Ø T "ERROR" !
```

will result in "ERROR" being typed on the teletype.

If X=Ø, then the command will be interpreted as G Ø, and control will be transferred to the lowest line number in the program.

2.5 COMPUTED DO

$$D\ X,G_1,G_2\ldots G_n \hookleftarrow$$

where X is a legal variable, and G_i is a program group or line number.

FOCAL-RT allows computed subroutine calls using the DO command, and the syntax is similar to that of the computed GOTO. If DO is followed by a legal variable X, that variable is evaluated and integerized. The Xth line or group of lines listed in the command will be returned to the command following the DO statement.

```
e.g.,  1.1Ø SET J=2
       1.2Ø DO J, 2.Ø, 3.Ø, 4.13
       1.3Ø Q
```

will result in execution of all group 3 lines, after which the program will halt. In the above example, if J=3, line 4.13 will be executed, and then the program will halt.

If DO is followed by a literal rather than a variable, the command will be interpreted as a standard FOCAL DO command.

```
e.g.,  1.1Ø DO 2.Ø
       1.2Ø QUIT
```

will result in execution of all group 2 lines, and then the program will halt.

NOTE: If the value of X in a computed DO command is greater than the number of lines listed in the command, then the program exits from the DO statement and proceeds to the next FOCAL command.

III. NEW FUNCTIONS

3.1 FRAN( )

SET A=FRAN(X)

defines the variable A as a random integer such that 1≤A≤X.

In many research applications, it is important to have a random number generator with adequate distributional and sequential characteristics. The algorithm used by FOCAL-RT's FRAN( ) function was developed by Green, Smith, and Klem (1959)*, and it has been thoroughly documented and tested. The periodicity of the sequences produced by it is in the neighborhood of 67 million.

In order to generate a random integer from 1 to 25, one need only call FRAN( ) with 25 as the argument:

```
1.1Ø SET A=FRAN(25)
```

Jiggling FRAN( ). With FOCAL-RT, the random number function is "jiggled" when FRAN( ) is called for the first time. If this were not done, the same sequence of numbers would be produced each time FOCAL is loaded. The KW-12A clock is used in conjunction with sense-switch #1 on the computer console to produce a random seed number. If the KW-12A clock is not available, then the right switch register may be used to vary the initial random number (see 8.2).

When FRAN( ) is called for the first time after FOCAL-RT has been loaded, the clock is set ticking at its fastest rate (4ØØ KHz.). The computer then waits for the operator to hit SNS-1. As soon as this is done, the current value of the clock counter is used to jiggle FRAN( ). It is only necessary to hit SNS-1 the first time FRAN( ) is called. After the random number function has been initialized, it generates numbers directly.

During the execution of a program, it may not be obvious to the user that he is supposed to hit SNS-1 at a given point in time, unless a reminder is given. The following routine may be useful for initializing FRAN( ):

---

* Green, B.F., Jr., Smith, J.E.K., & Klem, L. Empirical tests of an additive random number generator. Journal of the Association of Computing Machinery, 1959, 6, 527-537.

```
2.1Ø T "HIT SNS-1" !      /A HELPFUL REMINDER
2.2Ø O I,2; O I           /WAIT FOR TYPING TO FINISH
2.3Ø S A=FRAN( )          /INITIAL R.N.
2.4Ø O I                  /THIS WON'T WORK: SEE BELOW
```

NOTE: Since the initial FRAN( ) call uses the KW-12A clock, it will change the clock rate and mode from the values determined by a previous OUTPUT INTERVAL command, e.g., line 2.2Ø. The O I command in line 2.4Ø will not produce a delay of 2 sec. unless the clock is reset as follows

```
2.4Ø O I,2; O I
```

Generating Random Sequences. If it is not necessary to specify how many times each item occurs in the sequence, then the following routine may be used to give randomization with replacement:

```
1.1Ø ASK "NO. OF ITEMS TO BE RANDOMIZED?" S,!
1.2Ø ASK "SEQUENCE LENGTH?" N,!
1.3Ø TYPE "HIT SNS-1",!
1.4Ø O I,2; O I
1.5Ø FOR I=1,N; TYPE FRAN(S),!
```

A no-replacement randomization routine is provided in Section VI, together with several examples of the use of the new FRAN( ) function.

3.2 FIN( )

SET A=FIN( ) ↵

sets A equal to the decimal ASCII code of a keyboard input.

Normally FIN( ) is used in conjunction with FOUT( ) (Section 3.3) to handle character strings in FOCAL integer data files. This is particularly useful if one wishes to store large amounts of alphanumeric text with FOCAL, since FOCAL data files do not compete with program text for core.

```
e.g., 1.1Ø  L M,1,DATA,Ø
      1.2Ø  L O,FØ,I,DATA,Ø
      1.3Ø  FOR I=Ø,19;  SET FØ(I)=FIN( )
      1.4Ø  L C, FØ
```

This program creates a FOCAL file named DATA, and then opens it as an integer file. Line 1.3Ø accepts 2Ø characters from the teletype keyboard and stores their ASCII codes in locations Ø to 19 of the data file.

CTRL/Z may be used as a terminator when FIN( ) is used in conjunction with the FOR command. This is useful when one does not know in advance how long the character string will be...as may be the case with a question/answer dialogue. If, in the above example, the 1Øth character typed is CTRL/Z, then

(a) the FOR statement will be terminated with FØ(9) containing the last character (the ASCII code for CTRL/Z) of the input string;
(b) the value of I will be set equal to its value at the time CTRL/Z was inputted, plus one; in this case, I=1Ø;
(c) the program will proceed to the next command; in this case, it will close the data file.

Programming Notes:

(1) The RUBOUT facility is available with FIN( ).
(2) With FOCAL-RT, two tape blocks worth of core (512_{10} locations) are used as a buffer for FOCAL files. When a file location that is not represented in core is referenced in a FOCAL program, tape motion will occur as a tape block containing the relevant datum is searched. If this happens while a character string is being inputted using FIN( ), some of the input may be lost. This can be avoided by making sure that a single FIN( ) instruction refers only to file locations that are currently in the core file buffer.

(3) In some situations, it may be necessary to input a series of successive character strings. For example, one might want to have a respondent type his name, his place of birth, the date, etc., and in each case, the programmer does not know in advance how many characters are required for each piece of information. If the respondent uses CTRL/Z to terminate an input string, it is necessary to keep track of where that string ends so that the next string can be read into adjacent file locations. In the following example, this is done recursively:

```
1.1Ø  L O, F1,I,DATA,Ø
1.15  SET A=Ø
1.2Ø  TYPE !  "NAME:"; DO 2.Ø
1.3Ø  TYPE !  "RANK:"; DO 2.Ø
1.4Ø  TYPE !  "SERIAL NUMBER:"; DO 2.Ø
2.1Ø  FOR I=A, 1ØØ; SET FØ(I)=FIN( )
2.2Ø  SET A=I
2.3Ø  TYPE !
```

(4) CTRL/Z works as a terminator with FIN( ) and FOUT( ) only if these commands directly follow a FOR command.

```
e.g.,  1.1Ø FOR A=1,1Ø; S C=FIN( )
```

CTRL/Z does not work in the following DO loop:

```
1.1Ø  F I=Ø,2O; D 2.Ø
1.2Ø  Q
2.1Ø  S A=FIN( )
2.2Ø  T !!!
```

(5) A table of decimal ASCII codes is given in Section 8.3.

3.3 FOUT( )

```
SET D=FOUT(A) 2
```

will type or display on the current output device a character whose decimal ASCII code is equal to A. Here, D is a dummy variable.

Normally, FOUT( ) is used to output characters whose ASCII codes have been previously stored in a FOCAL data file with FIN( ).

```
2.Ø5  OUTPUT TELETYPE
2.1Ø  L O,FØ,I,DATA,Ø
2.2Ø  FOR I=Ø, 19; SET D=FOUT (FØ(I))
2.30  L C, FØ
```

This example would type out the characters whose ASCII codes were stored in locations Ø to 19 of an integer file named DATA.

CTRL/Z acts as a terminating character with FOUT( ) just as with FIN( ). If during execution of line 2.2Ø, the code for CTRL/Z is encountered, the program loop will be terminated and control will be transferred to line 2.3Ø. The variable I will be incremented by one over its value at the time CTRL/Z is found.

Often a set of character strings may be stored sequentially in a data file, each separated by the code for CTRL/Z. In that case, the user may not know exactly where each string is stored. The following routine may be used to keep track of file locations when outputting successive strings.

```
3.1Ø  L O, FØ,I,DATA,Ø
3.15  SET A=Ø
3.2Ø  TYPE ! "NAME:"; DO 4.Ø
3.3Ø  TYPE ! "RANK:"; DO 4.Ø
3.4Ø  TYPE ! "SERIAL NUMBER:"; DO 4.Ø
3.5Ø  QUIT
4.1Ø  FOR I=A,1ØØ; SET D=FOUT(FØ(I))
4.2Ø  SET A=I
4.3Ø  TYPE !
```

3.4 FPRI( ): LINE-PRINTER OPTION

SET D=FPRI( ) 2

diverts output to the LPØ8 line printer. Use 0 T to return to the teletype, 0 S to return to the scope.

IV. OTHER MODIFICATIONS OF FOCAL-12

4.1 FUNCTIONS DELETED

Several mathematical functions available in FOCAL-12 have been deleted in FOCAL-RT in order to increase the area available for user programs. These are:

FEXP
FATN
FLOG
FSIN
FCOS

4.2 FOCAL-RT USER AREA

Table 4.2-1 outlines the space available for program text, variable storage, and push-down list in FOCAL-12 and FOCAL-RT.

TABLE 4.2-1

	Core Boundaries	No. of Locations	Locations Gained	% Gain
FOCAL-12	3220_8-4617_8	767_{10}	--	--
FOCAL-RT	3220_8-5111_8	953_{10}	186_{10}	24.2

The end of the user area of both FOCAL-12 and FOCAL-RT is labelled BOTTOM; i.e., for FOCAL-RT, BOTTOM=5111_8. User overlays are added to FOCAL by redefining BOTTOM to a lower memory location, thereby sacrificing text storage. The user patch is then positioned between the old and new definition of bottom.

4.3 LIBRARY SAVE AND LIBRARY MAKE

With FOCAL-12, the user is warned if he attempts to save a program under a name already listed in the DIAL index. He must indicate that he wishes to replace the file by typing "R" on the keyboard. FOCAL-RT does not have this feature, and files are saved directly regardless of whether or not the file name was used previously.

The same holds true when files are being created with the LIBRARY MAKE command.

V. LINKING SUBROUTINES FOR ASSEMBLY-LANGUAGE PROGRAMS

Two subroutines are described here that facilitate the interleaving of FOCAL-RT and assembly-language programs. When incorporated into assembly-language programs, RD-WRI allows the user to open or close a FOCAL data file whose name had been previously specified by a LOCATE FILE command (Section 2.2). A second subroutine, FOCLDR, allows the user to chain FOCAL-RT user programs to assembly-language programs. Both subroutines simplify the residual machine-language programming necessary with the FOCAL-RT system by allowing the user to access program and data files by name rather than by absolute tape or disk addressing, and by making the tape or disk I/O operations invisible to the user.

5.1 RD-WRI: OPENING AND CLOSING FOCAL DATA FILES WITHIN ASSEMBLY-LANGUAGE PROGRAMS

RD-WRI uses information previously stored in locations ØØØ1-ØØØ3 of FIELD 1 by a FOCAL-RT LOCATE FILE command (Section 2.2) to read a FOCAL data file into core from tape or disk, or to write it back onto the storage device.

With RD-WRI, it is not necessary for the user to know where a particular file is located on a storage device, since that information is obtained by the LOCATE FILE command. Moreover, RD-WRI eliminates the need for modifying the assembly-language program if a file is moved, even to a different storage device.

To open the FOCAL file whose parameters have been saved by a LOCATE FILE command, enter RD-WRI with a JMS READIN OR JMS I PREADIN.

To write the file back onto tape, do a JMS WRIOUT or a JMS I PWRIOUT.

When adding RD-WRI to an assembly-language program, it is necessary to set location "MBLK1" of that subroutine to the starting memory block for the data file. For the purposes of the RD-WRI command, the first 8K of the PDP-12 memory are divided into a total of 40_8 blocks, each containing 400_8 locations. This organization is shown in Figure 5.1-1.

The maximum length of the data file is determined by the amount of core in FIELD Ø that the user wishes to reserve for his assembly-language program. It is generally a good idea to place the assembly-language program in the lower part of FIELD Ø, as shown in Figure 5.1-1. Then, the remainder of FIELD Ø and all of FIELD 1 (except for the last two memory blocks) may be used for data files.

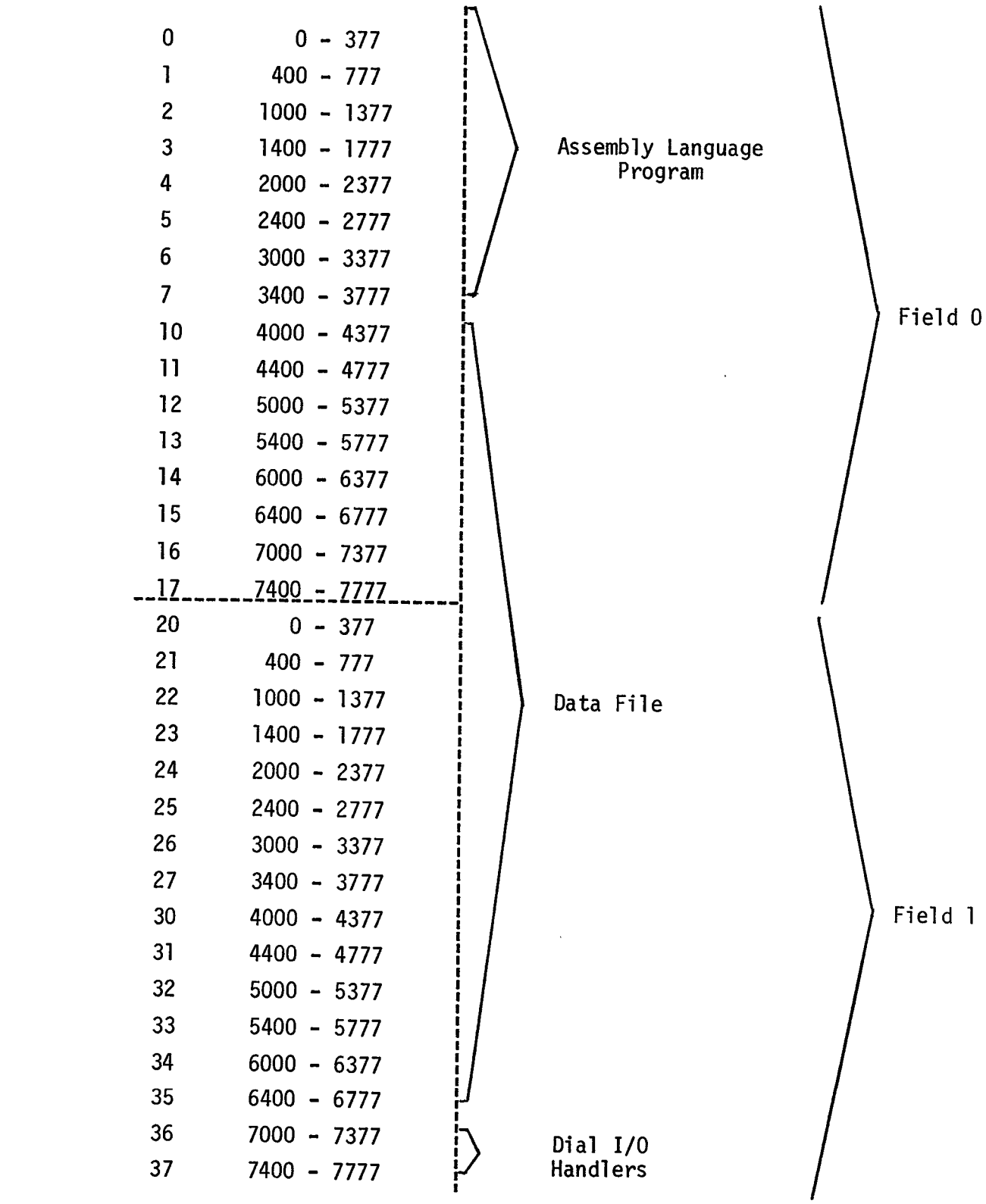

Memory Block
0 0 - 377
1 400 - 777
2 1000 - 1377
3 1400 - 1777
4 2000 - 2377
5 2400 - 2777
6 3000 - 3377
7 3400 - 3777
10 4000 - 4377
11 4400 - 4777
12 5000 - 5377
13 5400 - 5777
14 6000 - 6377
15 6400 - 6777
16 7000 - 7377
17 7400 - 7777
20 0 - 377
21 400 - 777
22 1000 - 1377
23 1400 - 1777
24 2000 - 2377
25 2400 - 2777
26 3000 - 3377
27 3400 - 3777
30 4000 - 4377
31 4400 - 4777
32 5000 - 5377
33 5400 - 5777
34 6000 - 6377
35 6400 - 6777
36 7000 - 7377
37 7400 - 7777
Assembly Language Program
Data File
Dial I/O Handlers
Field 0
Field 1

FIGURE 5.1-1

N.B.--Data files must not extend into memory blocks 36 and 37 in FIELD 1. This area is reserved for the DIAL-MS I/O handlers, which are left over from FOCAL after the assembly-language program has been read into FIELD Ø with a LOAD BINARY command, and which are used by RD-WRI.

NOTES: (1) RD-WRI is 72_8 locations long, and it is written in 8-mode.
(2) It may be placed on any page of core.
(3) The source of RD-WRI is found on the FOCAL-RT systems LINCtape (available from DECUS).
(4) RD-WRI works only on files with no header block--i.e., FOCAL data files.
(5) The operation of RD-WRI is flow-charted in Section 8.4.

5.2 FOCLDR

FOCLDR is an assembly-language subroutine that loads FOCAL-RT into core, initializes it, then loads and starts a FOCAL user program from tape or disk. With the availability of FOCLDR and the FOCAL-RT LOAD BINARY command (Section 2.1) it is possible to go back and forth between FOCAL and assembly-language programs without touching the computer console.

FOCLDR is capable of reading any of 10_{10} different FOCAL programs into the FOCAL text buffer from unit Ø. To load a FOCAL program named PROGRAM3, jump to FOCLDR with 3_8 in the AC. For PROGRAMN, (Ø≤N≤9) one calls FOCLDR with the octal equivalent of N in the AC.

FOCLDR works by placing the ASCII codes for the following characters into the command buffer after FOCAL-RT has been loaded and initialized:

L G, PROGRAMN,Ø

where N=contents of AC when FOCLDR is called.

By substituting these ASCII codes with others of the user's choice, it is possible to load programs with any admissible name from any storage device handled by DIAL-MS. Otherwise it is necessary to save FOCAL programs as PROGRAMN, and to use LINCtape Ø as the storage unit.

NOTES:

(1) The positioning of FOCLDR in core is critical. It should start at location 4ØØØ of FIELD Ø so that it will not be read over by FOCAL-RT.
(2) Location PTR+2 of FOCLDR must be set to the starting binary tape block of FOCAL-RT. If FOCAL-RT is the first file added to the tape, it will start at block 243_8. This is one block past the location of FOCAL-RT specified by the DIAL index, since the first block recorded in the index is the header block. FOCLDR ignores the header block and reads the FOCAL-RT binary file into core directly.
   To simplify matters, always save FOCAL-RT as the first program on the tape. That way PTR+2 will always have the same value (i.e., 243_8).
(3) FOCLDR is a LINC-mode routine, and it occupies 113_8 core locations.
(4) A flow-chart of FOCLDR's operation is given in Section 8.5.

VI. PROGRAMMING WITH FOCAL-RT

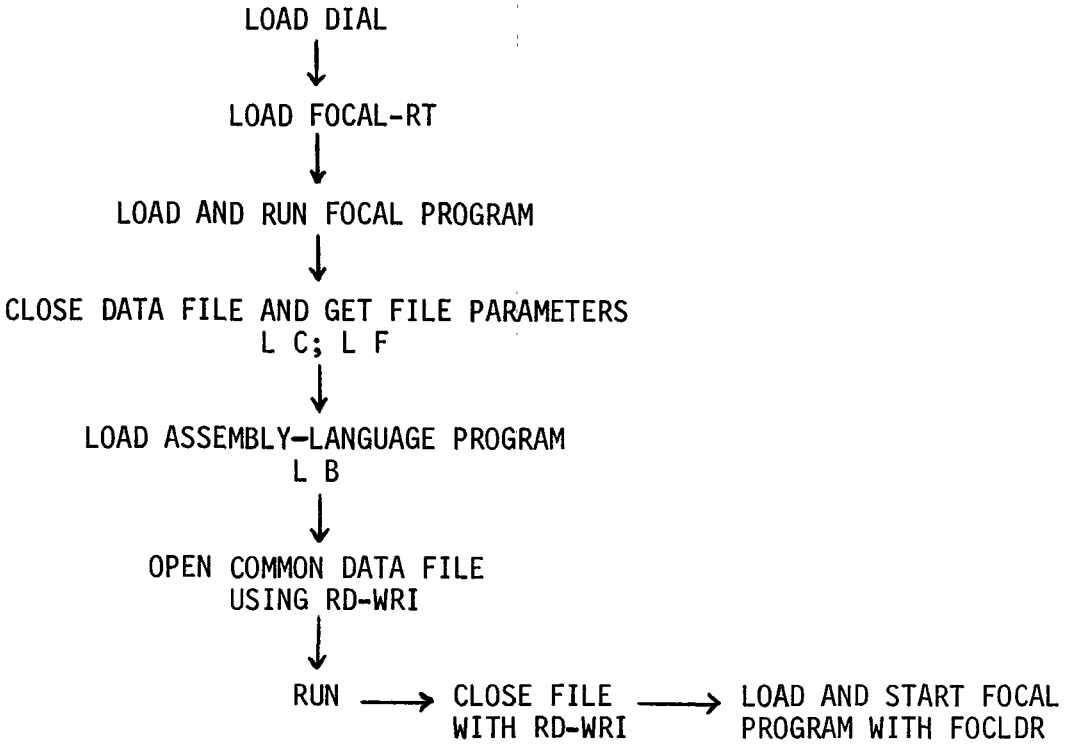
In many situations where timing is not too critical, where data rates are low, and where core limitations are not problemmatical, one may be able to program a research paradigm entirely with FOCAL-RT. This is especially feasible if the task involves alphanumeric input and output, where FOCAL-RT's ability to handle character strings comes to the fore.

More demanding tasks may be programmed with a combination of FOCAL and assembly language. Here one may use FOCAL-RT to do all of the more difficult work: inputting of parameters at the beginning of an experiment, setting up sequences of trials, data analysis, and so forth. A short assembly-language program may be used during the data-collection phase of the experiment. Because of the chaining features available with the FOCAL-RT operating system, it is not necessary to have the interpreter in core during critical phases of the research, and the limitations of an interpreted language are thus not applicable during those times. In most cases, a very considerable reduction in program development time may be achieved by interleaving FOCAL and assembly-language programs.

6.1 CHAINING FOCAL AND ASSEMBLY-LANGUAGE PROGRAMS

With FOCAL-RT's LOAD BINARY command, it is possible to go from a FOCAL program to an assembly-language program stored on tape or disk without touching the computer console (Section 2.1). Similarly, it is possible to chain a FOCAL user program to an assembly-language program by jumping to the FOCLDR subroutine (Section 5.1). Moreover, one may open and close named FOCAL files within an assembly-language program by using the RD-WRI subroutine (Section 5.2) in conjunction with the FOCAL-RT LOCATE FILE COMMAND (Section 2.2).

These chaining procedures may be summarized as follows:



6.2 RANDOMIZING SEQUENCES

If one does not wish to place restrictions on the sequence, then programming random events is very simple with FOCAL-RT. The following routine randomizes the order of three sentences that will be displayed on the PDP-12 scope. It uses the new FRAN( ) function and the computed DO.

```
2.1Ø  A "SEQUENCE LENGTH?" N !
2.2Ø  T "HIT SNS-1", !        /JIGGLE FRAN( )
2.3Ø  O I, 2; O I; S A=FRAN( )
2.4Ø  O S; F I= 1,N; D 3.Ø
2.5Ø  QUIT
3.1Ø  S A=FRAN(3)
3.2Ø  DO A, 4.1Ø, 4.2Ø, 4.3Ø  /COMPUTED DO
4.1Ø  T "THIS IS SENTENCE 1." !
4.2Ø  T "THIS IS SENTENCE 2." !
4.3Ø  T "THIS IS SENTENCE 3." !
```

Randomization Without Replacement.

If one wishes to restrict the sequence by specifying how many times each item occurs, then the following routine may be used to achieve randomization without replacement:

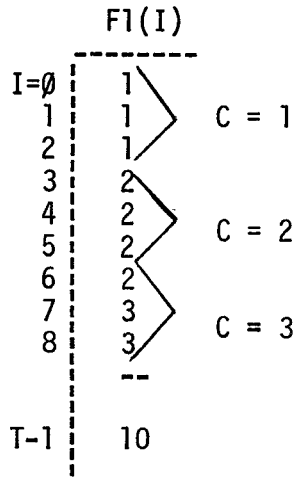


Figure 6.2-1

(1) Open a FOCAL data file, Fl.
(2) Determine the number of times, N(c), each of the items is to occur.
(3) Determine the total no. of trials, T.
(4) List each item N(c) in arbitrary order in Fl. The file will look like Fig. 6.2-1 at this point.
(5) The contents of Fl(Ø) will now be exchanged with those of one of the T file elements selected at random.
(6) The contents of Fl(1) are exchanged with those of one of the remaining T-1 file elements drawn at random.
(7) This process is repeated until all T elements have been shuffled. The file will now look like Fig. 6.2-2.

	F1(I)
I=Ø	9
1	2
2	5
3	2
	--
T-1	6

Figure 6.2-2

A FOCAL-RT routine using this algorithm is given below. It allows the user to specify the starting location in the data file, SL, where the random sequence is to begin.

```
01.10 G 10.1

02.10 T %6.0,C,"    ";A N(C),!

03.10 F A=1,N(C);D 4.0

04.10 S F1(I)=C
04.30 S I=I+1

05.10 S X=SL+T-FRAN(T-J)
05.20 S A=F1(X);S F1(X)=F1(SL+J);S F1(SL+J)=A

11.10 A "NUMBER OF ALTERNATIVES?" S,!!
11.20 T "ITEM NO.  NO. OF TIMES",!
11.30 F C=1,S;D 2.0
11.40 S T=0;F C=1,S;S T=T+N(C)

11.10 L 0,F1,I,SCRAM,0
11.20 A "STARTING FILE LOCATION" SL,!;S I=SL
11.30 F C=1,S;D 3.0
11.40 T "HIT SNS-1",!
11.50 O I,3;O I
11.60 F J=0,T-2;D 5.0
11.70 F I=SL,SL+T-1;T F1(I),!
*


*
NUMBER OF ALTERNATIVES?:3

ITEM NO.   NO. OF TIMES
       1     :2
       2     :2
       3     :2
STARTING FILE LOCATION:5
HIT SNS-1
       1
       3
       3
       2
       2
       1
*
```

6.3 COMMUNICATION BETWEEN FOCAL AND ASSEMBLY LANGUAGE

NOTE: The following material is largely exerpted from Siegel and Whittle (1972)*.

If one is to interleave FOCAL and assembly-language programs, it is necessary for the data produced by the various chained programs to be compatible. That is, codes produced by FOCAL programs must be intelligible to assembly-language programs, and vice versa. This is no problem if one places all of the common data in FOCAL files using 12-bit integer format.

If the decimal number 1Ø29 is placed in a FOCAL integer file, it is stored as a 12-bit binary equivalent. Since all of the machine codes for the PDP-12 consist of 12-bit binary numbers, then one can compute any machine-language instruction or datum using FOCAL-RT. Given the extensive set of mathematical functions that are built into FOCAL, this means that one can use this interpretive package as a very sophisticated compiler...a computational compiler, if you will.

When we use the term "computational compiler," we don't mean that FOCAL is used to compile an entire assembly-language program. Rather, it is used to compile a set of codes that can be used sequentially by subroutines in the assembly-language program for running the trials of an experiment. The compiled codes may be actual machine-language instructions, but more typically they are in the form of data that are used as operands by machine-language instructions.

With this system, machine-language programming can often be reduced to hooking together a few device handlers and timing routines. This is because all or most of the variable information can be set up in advance with FOCAL programs. The remaining assembly-language programming is often trivial.

Since FOCAL-RT can compute any 12-bit machine code, it can be used to compute the information necessary to control any standard PDP-12 peripheral...e.g., clocks, D:A converter, relays...as well as any computer-controlled user device. For example, we have used FOCAL-RT to set up the oscillator and timing codes for a complex psychoacoustic experiment that runs in assembly-language. A linear equation was solved by FOCAL-RT in order to compute the machine code necessary to produce any desired oscillator frequency.

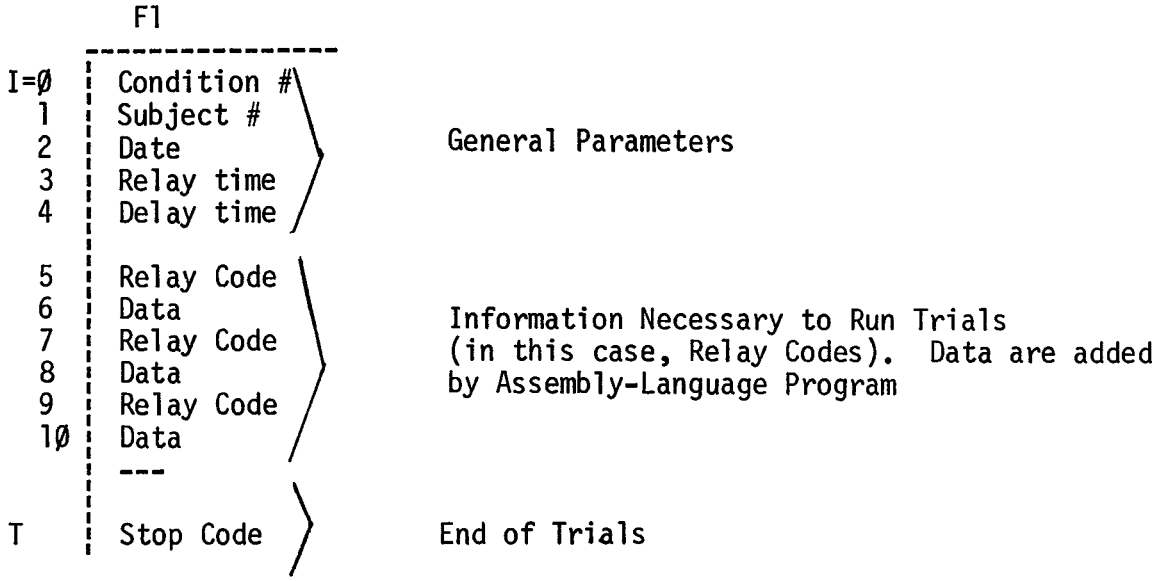
---

* Siegel, W., & Whittle, K. Using FOCAL in research. Proceedings of the DECUS Symposium, Spring, 1972.

NOTES:

(1) The trick is to specify the set of data necessary to run one's experiment in assembly-language, and to compute the decimal equivalent when working in FOCAL. An example of this strategy is given in Section 6.4.

(2) When setting up a FOCAL data file for later use by an assembly-language program, it is wise to leave space for the data that are to be added to the file when the experiment is being run. The following illustrates a general file organization that we have found useful in a number of research applications:



6.4 DEMONSTRATION PROGRAM

This section is exerpted from Siegel and Whittle (1972).*

To illustrate the use of FOCAL-RT as a computational compiler, we wrote a demonstration program to set the PDP-12 relays in random order using a combination of FOCAL and assembly-language. Program listings together with an example of the programs' output are provided in Figures 6.4-1 and 6.4-2. There are six relays on the PDP-12, and one could use the new FRAN function very easily to produce integers between 1 and 6. The problem is that these numbers don't mean much to the machine-language instruction that sets the PDP-12 relays. For relays 0, 1, 2, 3, 4, 5, the appropriate octal numbers that one must put in the accumulator to set the relays are 1, 2, 4, 10, 20, 40. The decimal equivalents are 1, 2, 4, 8, 16, 32. As you may have discerned, the required relay codes are related to the numbers 1-6 by a simple powers-of-two transformation. The required exponential equation to transform numbers produced by our random number function to codes intelligible to an assembly-language program takes only one line of FOCAL coding (Fig. 6.4-1, line 4.1Ø).

The relay demo works as follows:

(1) A FOCAL-RT program asks the experimenter how many different relays he wants to set, how long he wants them on for, what the intertrial time is, and how many times each relay is to occur in the sequence. Then a list of trials is generated, using a card-shuffling algorithm to give randomization without replacement. All of the necessary information is stored in a FOCAL data file, and then an assembly-language program is called up to run the experiment. This program works through the FOCAL file, turning the relays on and off, and timing. When the "experiment" is finished, the FOCAL demo program is loaded in again to request a new set of parameters, and so forth.

The assembly-language program for the relay demo requires only 45_{10} instructions, a number of which are involved in a canned routine for the KW12-A clock.

Note that the randomization routine given in Section 6.2 is used almost verbatim here.

---

* Siegel, W., & Whittle, K. Using FOCAL in research. Proceedings of the DECUS symposium, Spring, 1972.

```
C FOCAL-RT

01.10 T "RELAY DEMO TO SET THE PDP-12 RELAYS IN [illegible] ORDER" !!
01.20 G 10.1

02.10 T %6.[illegible],[illegible]-1,"     ";A N(C),!

03.10 F A=1,N(C);D 4.0

04.10 S F1(I)=2↑(C-1)
04.20 S I=I+1

05.10 S K=SL+T-F[illegible]A[illegible](T-J)
05.20 S [illegible]F1(K);S F1(K)=F1(SL+J);S F1(SL+J)=A

10.10 A "NUMBER OF DIFFERENT RELAYS (1-6)?",S !!
10.20 T "[illegible]        NO. OF TIMES" !
10.30 F C=1,S;D 2.
10.40 S T=0;F C=1,S;S T=T+N(C))

11.10 L [illegible],[illegible],T,[illegible],[illegible]
11.20 S [illegible]=[illegible];S I=SL
11.30 F C=1,S;D 3.
11.35 S F1(I)=7
11.40 T "HIT SENSE-SWITCH 1",!
11.50 [illegible] I,3;[illegible] I
11.60 F J=0,T-2;D 5.
11.70 A "RELAY TIME (SEC.)" RT,!;S F1(0)= RT*100
11.80 A "INTERTRIAL TIME?" IT,!;S F1(1)=IT*100

12.10 L C,F1;T "LEAVING FOCAL" !;O I,3;O I
12.20 L [illegible],RELAY,[illegible]
12.30 L R,LEXO,[illegible]


*
RELAY DEMO TO SET THE PDP-12 RELAYS IN RANDOM [illegible]

NUMBER OF DIFFERENT RELAYS (1-6)?:4

RELAY #    NO. OF TIMES
     0       :2
     1       :3
     2       :4
     3       :5
HIT SENSE-SWITCH 1
RELAY TIME (SEC.):1.5
INTERTRIAL TIME?:2.5
LEAVING FOCAL
```

FIGURE 6.4-1

Setting the PDP-12 relays in random order using a combination of FOCAL and assembly language: (a) FOCAL-RT program to input parameters and set up random sequence.

```
0000                    *20
0001                    PMODE
0002                    *400
0003                    RDWRI=100
0004    0400    4100    START,  JMS RDWRI       /GET DATA
0005    0401    6141            LINC
0006                            LMODE
0007    0402    0061            SET I 1         /INITIAL PTR
0010    0403    2010            2001
0011    0404    1021    BEGIN,  LDA I 1         /GET STIMULUS
0012    0405    1460            SAE I           /CHECK FOR STOP
0013    0406    0007            0007            /CODE.
0014    0407    0456            SKP             /CONTINUE.
0015    0410    6422            JMP RETURN      /BACK TO FOCAL.
0016    0411    0014            ATR             /CLOSE RELAY.
0017    0412    1000            LDA
0020    0413    2001            2000            /ON TIME
0021    0414    6426            JMP CLCK        /CLOCK ROUTINE
0022    0415    0014            ATR             /CLEAR RELAY
0023    0416    1000            LDA
0024    0417    2002            2001            /DELAY TIME
0025    0420    6426            JMP CLCK        /CLOCK ROUTINE
0026    0421    6404            JMP BEGIN       /LOOP.
0027    0422    0702    RETURN, 0702            /GET FOCLDR
0030    0423    0471            0471
0031    0424    0011            CLR
0032    0425    6020            JMP 20
0033    0426    0002    CLCK,   PDP             /CLOCK SUB-
0034                            PMODE           /ROUTINE.
0035    0427    7041            CIA
0036    0430    6133            CLAB
0037    0431    7200            CLA
0040    0432    6132            CLLR
0041    0433    6134            CLEN
0042    0434    1252            TAD CKCON1
0043    0435    6132            CLLR
0044    0436    6135            CLSA
0045    0437    7200            CLA
0046    0440    1253            TAD CKENU
0047    0441    6134            CLEN
0050    0442    7200            CLA
0051    0443    1254            TAD CKCON2
0052    0444    6132            CLLR
0053    0445    7200            CLA
0054    0446    6131            CLSK
0055    0447    5246            JMP .-1
0056    0450    6141            LINC
0057                            LMODE
0060    0451    6000            JMP 0
0061    0452    0100    CKCON1, 0100
0062    0453    0300    CKENU,  0300
0063    0454    5100    CKCON2, 5100
```

FIGURE 6.4-2

Setting the PDP-12 relays in random order using a combination of FOCAL and assembly language: (b) assembly language program.

VII. MISCELLANEOUS TECHNICAL NOTES

7.1 MINIMIZING PROGRAM LENGTH

The area of FOCAL-RT that is available for user programs is quite restricted. Eventually we may be able to solve this problem through a major reorganization of FOCAL-12. In the meantime, the user is referred to pp. 61-62 of the Laboratory Computer Handbook, where several tips for minimizing program length are outlined.

Whenever possible, define variables as FOCAL file variables. All non-file variables use five core locations each. On the other hand, one may define file variables without impinging upon the space available for user programs. File variables are handled in a separate core buffer area that is 512_{10} locations long. Moreover, if one has integer data, each variable requires only a single word of core. File variables that are not represented in the core file buffer will be automatically read in from tape or disk by the FOCAL file handlers, and this may take some time to accomplish (c.f., p. D-1; FOCAL-12 Programming Manual).

7.2 MINIMIZING TAPE MOTION

When chaining programs or referencing FOCAL files, considerable delays may be incurred in a LINCtape-based system. One solution is to buy a disk. A less expensive alternative is to spend some time organizing one's programs so that tape motion is minimized.

One source of tape motion arises when one opens or closes FOCAL data files, i.e., when data are transferred between core and a mass storage device. When leaving one FOCAL program and loading a second, it is not necessary to close a data file and reopen it again if the same data are used by the two programs. That is, program chaining with the LIBRARY GO instruction does not destroy the file buffer contents or the file pointers. On the other hand, the values of all non-file variables are lost when a new FOCAL program is loaded. Thus it is efficient to define all common variables as file variables; that way they will not have to be redefined each time a program is chained, and the length of the chained programs will be reduced.

Another source of tape motion arises from the small size of the core buffer area for FOCAL data files. This is the same in FOCAL-RT as in FOCAL-12...512_{10} words, or two blocks of LINCtape. Whenever a file location is referenced that is not represented in the file buffer area, the tape block containing that file location must be searched and read into core. On the other hand, one may refer to any of the file variables that are currently represented in core without incurring tape motion.

When referencing FOCAL files, there are several means of reducing tape motion:

(1) Position files on tape so that they are as close together as possible. In particular, make sure that they are not on opposite sides of the area reserved for DIAL.
(2) If possible, limit the number of data files that are referenced by the same instruction or group of instructions to one or two. It is better to have the data organized in one large file rather than in several smaller files.
(3) Within a file, position the data so that material that is referenced together or compared is as close together as possible. For example, when intercorrelating three sets of scores, it is better to have the triads that are to be compared listed together in one file than to have the data set up in three separate files. The latter organization will result in interminable tape motion.

		F1(I)
i.e.,	I=Ø	$A_Ø$
	1	$B_Ø$
	2	$C_Ø$
	3	A_1
	4	B_1
	5	C_1
	6	A_2
	:	:

is efficient in terms of tape motion.

	F1(I)	F2(I)	F3(I)
I=Ø	$A_Ø$	$B_Ø$	$C_Ø$
1	A_1	B_1	C_1
2	A_2	B_2	C_2
--	--	--	--

This organization is not.

7.3 USING THE TYPE COMMAND

FOCAL uses an interrupt-driven typing routine so that program execution will not be delayed by the slow operation of the teletype. This means that commands following the TYPE statement will be executed while the typing is being completed. Commands that require considerable execution time themselves can therefore hang up the typing of a text string. This problem may arise with any of the LIBRARY commands, as they require tape motion, and with the initial call of the FOCAL-RT FRAN( ) function (Section 3.1).

The solution is to program a delay after a TYPE command, using the OUTPUT INTERVAL command. This prevents FOCAL from executing further instructions before the typing is finished.

```
(a)     1.1Ø  T "LEAVING FOCAL" !
        1.2Ø  O I, 3;O I
        1.3Ø  L B, RUN,Ø

(b)     1.1Ø  T  "HIT SNS-1" !
        1.2Ø  O I, 2;O I
        1.3Ø  S A=FRAN( )
```

7.4 MAXIMIZING SPEED OF PROGRAM EXECUTION

Payne (1971)* has outlined two procedures for increasing the speed of program execution with FOCAL. These involve:

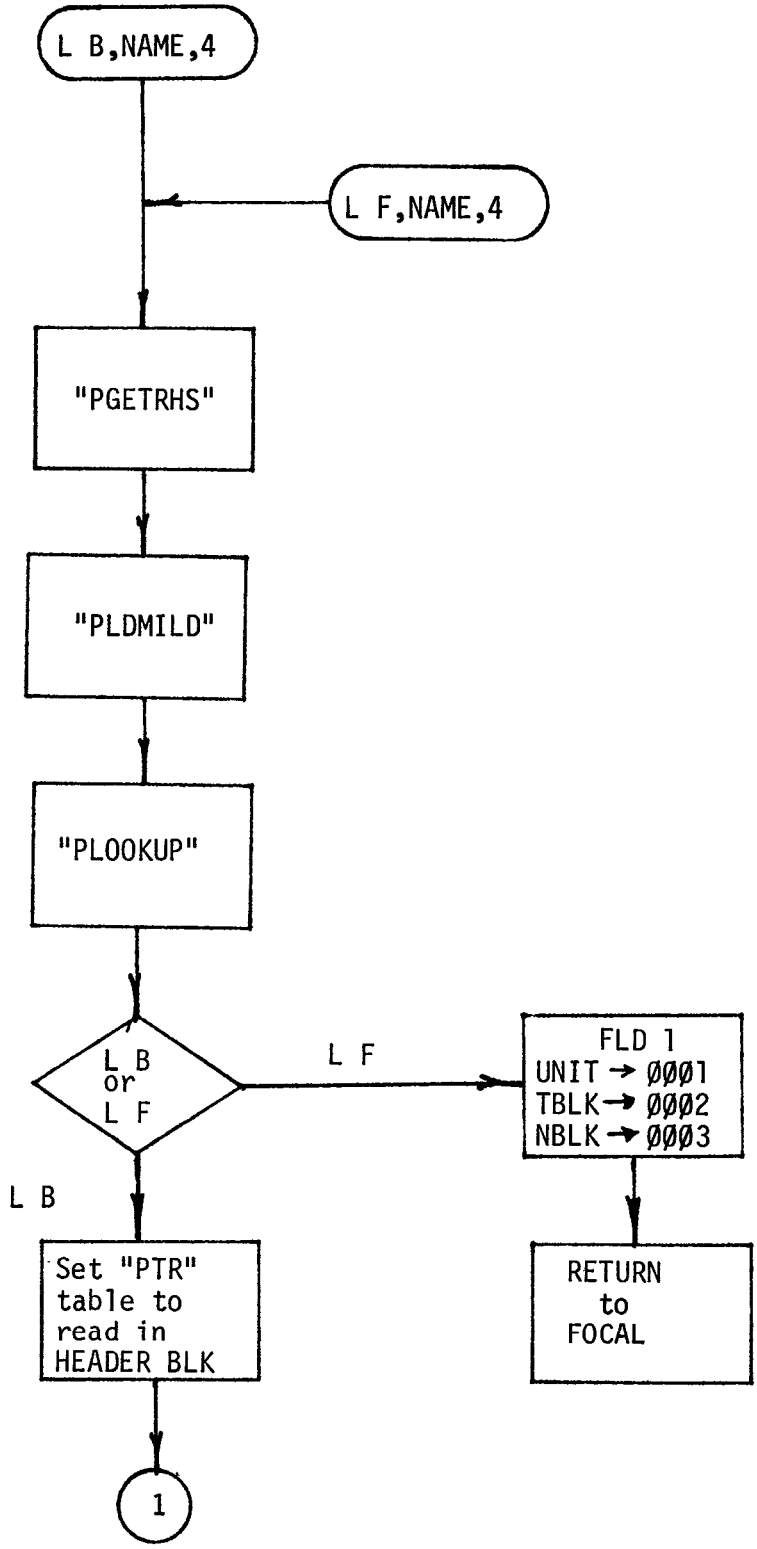
(1) Giving low line numbers to frequently referenced program lines, such as subroutines.
(2) Defining variables in order of frequency of use. This may be done by executing direct commands setting them to zero, in order of decreasing frequency of use, after the indirect program has been completed, but before it is executed or saved on tape.

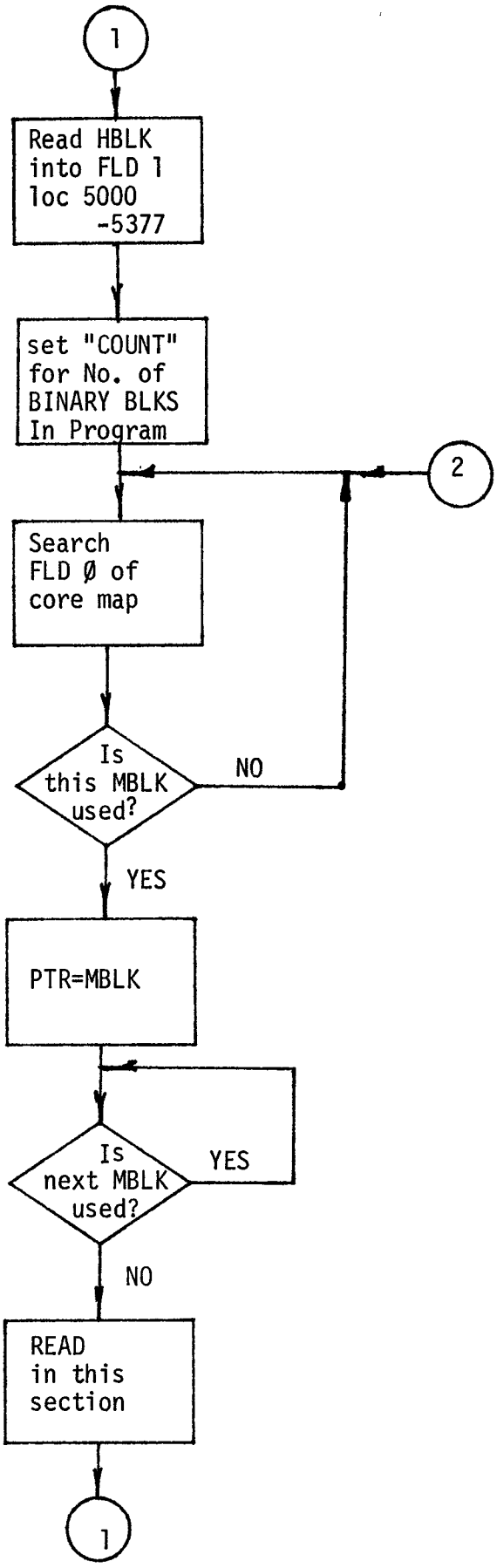
---

* Payne, W.D. Decreasing the execution time of FOCAL programs. DECUSCOPE, 1971, 9, 19.

APPENDICES

8.1 FLOW CHART DESCRIBING THE OPERATION OF THE LOAD BINARY AND LOCATE FILE COMMANDS

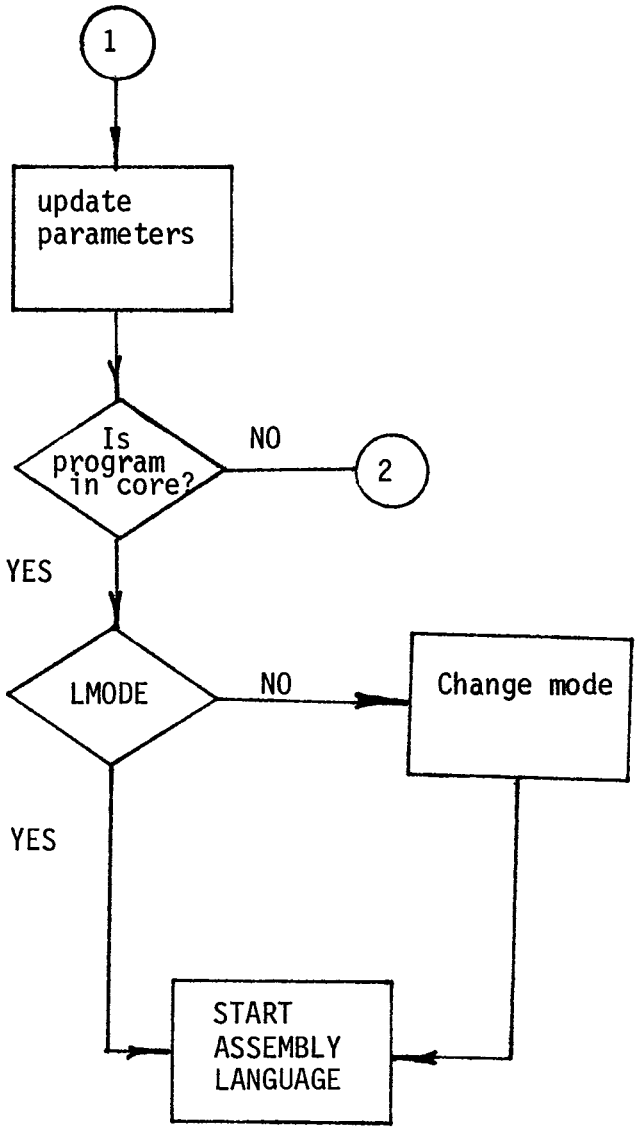




-search locations 1534Ø to 15357
-if location equals 7777, the MBLK represented by this location is used
-if ØØØØ it is not used

-find number of continuous blocks

-use DIAL I/O routine to read in section PTR, Ø /UNIT
Ø /core location
Ø /starting TBLK
Ø /number of continuous blocks


1
update parameters
Is program in core?
NO
2
YES
LMODE
NO
Change mode
YES
START ASSEMBLY LANGUAGE

8.2 FOCAL-RT PATCH FOR JIGGLING FRAN( ) THROUGH THE RIGHT SWITCH REGISTER

If the KWI2A clock peripheral is not available to the user the right switches may be substituted as a means of varying the initial random number.

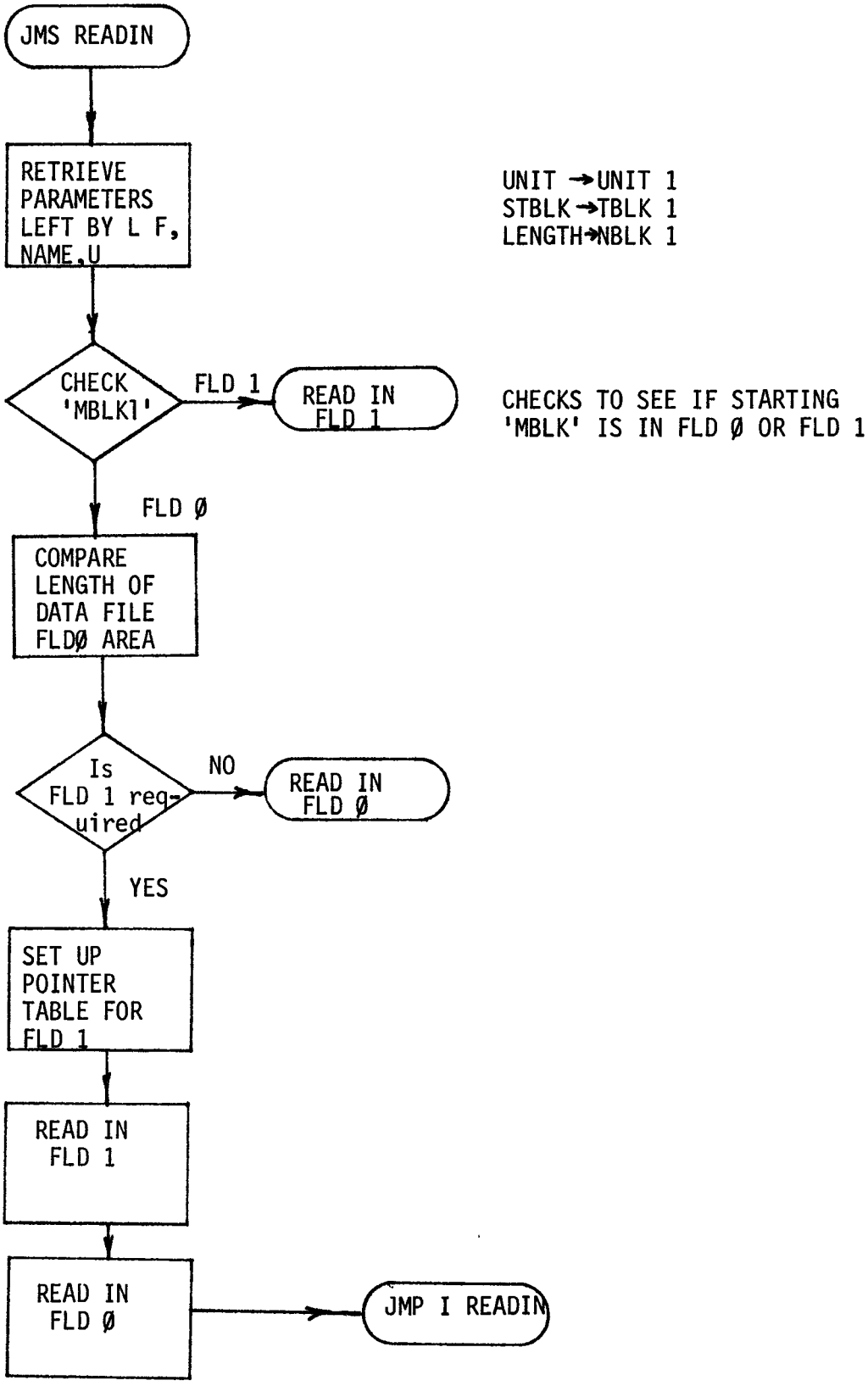
i.e.	Location	From	To
	5216	6132	RSW
	5217	6134	DCA VALUE
	522Ø	1267	SKP
	5221	6132	VALUE, Ø
	5231	6137	TAD VALUE

8.3 DECIMAL ASCII CODES

FOR FIN( ) AND FOUT( )

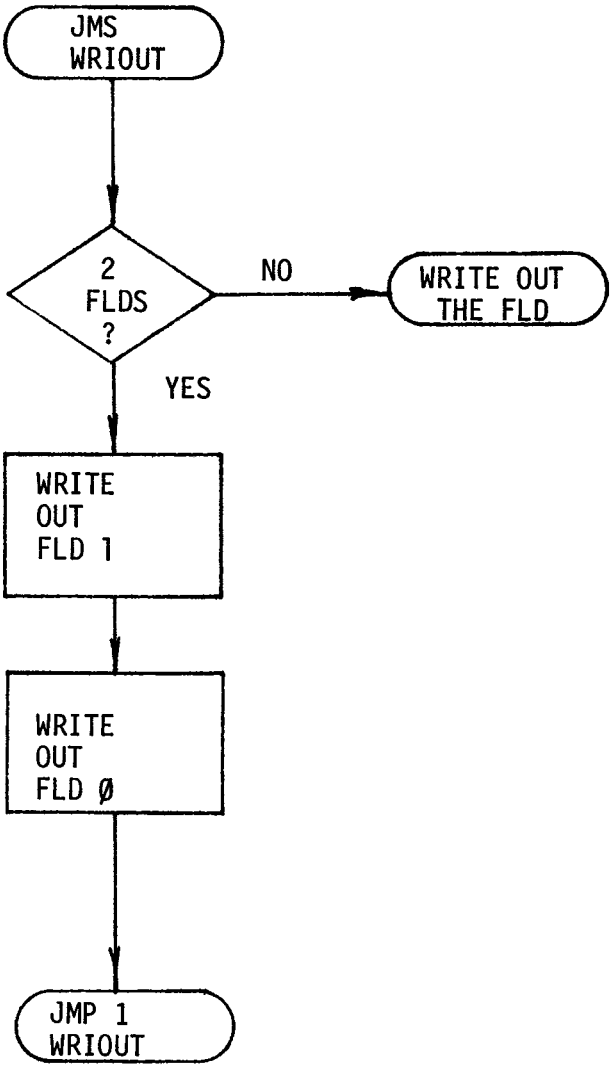
CODE	CHARACTER	CODE	CHARACTER	CODE	CHARACTER
128	CTRL/SHFT/P (LEADER)	160	SPACE	193	A
129	CTRL/A	161	!	194	B
130	CTRL/B	162	"	195	C
131	CTRL/C	163	#	196	D
132	CTRL/D	164	$	197	E
133	CTRL/E	165	%	198	F
134	CTRL/F	166	&	199	G
135	CTRL/G (BELL)	167	'	200	H
136	CTRL/H	168	(	201	I
137	CTRL/I	169	)	202	J
138	LINE FEED	170	*	203	K
139	CTRL/K	171	+	204	L
140	CTRL/L	172	,	205	M
141	CARRIAGE RETURN	173	-	206	N
142	CTRL/N	174	.	207	O
143	CTRL/O	175	/	208	P
144	CTRL/P	176	0	209	Q
145	CTRL/Q	177	1	210	R
146	CTRL/R	178	2	211	S
147	CTRL/S	179	3	212	T
148	CTRL/T	180	4	213	U
149	CTRL/U	181	5	214	V
150	CTRL/V	182	6	215	W
151	CTRL/W	183	7	216	X
152	CTRL/X	184	8	217	Y
153	CTRL/Y	185	9	218	Z
154	CTRL/Z	186	:	219	[
155	CTRL/SHFT/K	187	;	220	\
156	CTRL/SHFT/L	188	<	221	]
157	CTRL/SHFT/M	189	=	222	^
158	CTRL/SHFT/N	190	>	223	←
159	CTRL/SHFT/O	191	?	253	ALT MODE
		192	@	255	RUBOUT

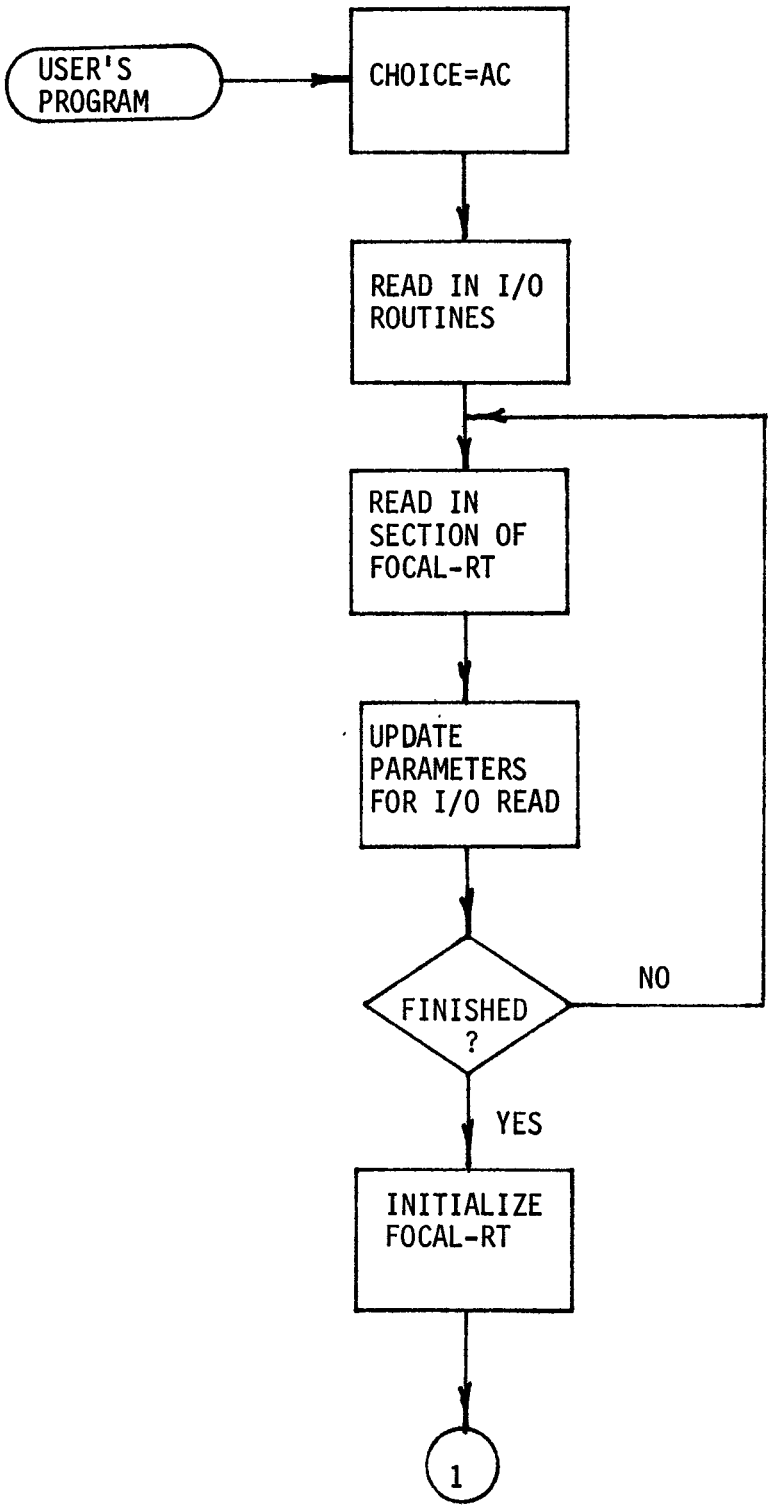
READIN PORTION

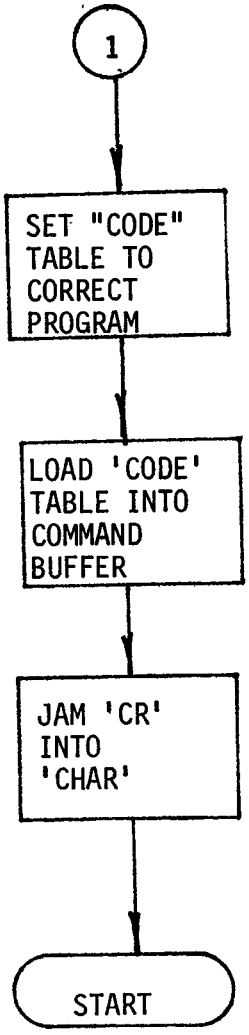


RD-WRI SUBROUTINE

WRIOUT PORTION




USER'S PROGRAM
CHOICE=AC
INDICATION OF FOCAL-RT PROGRAM
READ IN I/O ROUTINES
READ IN SECTION OF FOCAL-RT
UPDATE PARAMETERS FOR I/O READ
MBLK
TBLK
NBLK
FINISHED ?
NO
FOCAL-RT IS IN CORE
YES
INITIALIZE FOCAL-RT
1


1
SET "CODE" TABLE TO CORRECT PROGRAM
LOAD 'CODE' TABLE INTO COMMAND BUFFER
JAM 'CR' INTO 'CHAR'
START

ADDENDUM TO DECUS NO. 12-80

The Library Save, Library Load and Library Go commands in FOCAL-RT truncate the upper end of the FOCAL user area upon storage and retrieval. The last 5 words of the user area (5105-5111) are not saved or restored. (This can cause the first variable stored and saved to not appear when the program is retrieved.) If an overlay is used as suggested in the FOCAL-12 manual (DEC-12-AJAA-D) section 7.3, the last five instructions in an overlay will be forgotten. The problem is due to a wrong constant in location 11323. This constant tells FOCAL how many user locations to save or restore. In FOCAL-RT it contains 6102; it should be changed to 6075. In the DECUS NO. 12-80 tape, this is BLOCK 263; word 323. Line 1016 at the source should be changed from 3214-5112 to 3207-5112. Programs saved before this change is made should be called and checked and resaved.