FLOATING POINT
SYSTEMS, INC.

FPS-100 Linker (LNK100) Reference Manual

860-7441-000

Publication No. 860-7441-000
October, 1979

NOTICE

This edition applies to Release A of FPS-100 software and all subsequent releases until superseded by a new edition.

The material in this manual is for informational purposes only and is subject to change without notice.

Floating Point Systems, Inc. assumes no responsibility for any errors which may appear in this publication.

Printed in USA

by FPS Technical Publications Staff

FPS-100 Linker (LNK100) Reference Manual

860-7441-000

Publication No. 860-7441-000
October, 1979

NOTICE

This edition applies to Release A of FPS-100 software and all subsequent releases until superseded by a new edition.

The material in this manual is for informational purposes only and is subject to change without notice.

Floating Point Systems, Inc. assumes no responsibility for any errors which may appear in this publication.

Printed in USA

CONTENTS

ILLUSTRATIONS

TABLES

CHAPTER 1

OVERVIEW

1.1 INTRODUCTION

LNK100 links together separate object modules produced by ASM100 into a single load module for execution by the FPS-100 hardware or the simulator.

The user can separately code and assemble a main line program and the associated subroutines and later link them together for execution. LNK100 serves this purpose by performing the following tasks:

- relocating each object module and assigning absolute addresses
- linking the modules together by correlating global entry symbols defined in one module with external symbols referenced in another module
- selectively loading modules from program library
- optionally producing a load map showing the layout of the load module

1.2 RELATED MANUALS

The manuals in Table 1-1 may also be of interest to the user.

Table 1-1 Related Manuals

MANUAL	PUBLICATION NO.
FPS-100 Math Library Manual	FPS 860-7429-000
ASM100 Reference Manual	FPS 860-7428-000
SIM100/DBG100 Reference Manual	FPS 860-7424-000
FPS-100 Programmer's Reference Manual Volumes One and Two	FPS 860-7427-000
VFC100 Manual	FPS 860-7447-000
APX100 Manual	FPS 860-7426-000

CHAPTER 2

OPERATING PROCEDURE

2.1 SUMMARY

Program modules are linked interactively via a dialogue between the user and LNK100. The user enters a series of commands which direct the linking process.

When execution begins, LNK100 displays:

```
LNK100      version date
*
```

The version is the version number of LNK100, and date is the release date of LNK100. The asterisk (*) indicates that the program is ready to accept commands. After each user command, an * is displayed when that command has been executed and ASM100 is ready for a new command. An illegal command causes a ? to be displayed.

To load relocatable programs and prepare them for execution, the user would normally use the following procedure.

1. Using the L (load) command, load the file or files containing the desired main program, required subroutines, and library subprograms, if any. If a fatal error occurs during this step, reinitialize using the R command and repeat this step.

2. Using the U (undefined) command, check to see if any global symbols are still undefined. If nothing is listed from this command, continue to step 3. If any symbols are listed, it usually means that there was an error in one or more of the programs loaded or that the loading sequence was wrong. In these cases, correct the error and restart the loading operation from step 1.

3. Obtain the memory limits of the loaded program or a loader map by using the M (memory) or S (symbols) command.

4. Complete and generate the load module by using the E (end) command or the A command. Note the values of HIGH and START as well as the possible presence of any remaining undefined symbols.

5. Return to the operating system with an X (exit) command.

The individual LNK100 commands are described in the following sections, and a complete example loading session is given in section 2.14.

2.2 CONVENTIONS

The following abbreviations are used in the remainder of this manual:

Abbreviations	Meaning
(filename)	A user-specified input or output file. The (filename) follows whatever naming conventions exist for the particular host computer operating systems.
*	This is the terminal prompt indicating that the terminal is ready for input.
------	Indicates characters entered by the user. All user input is terminated with a carriage return.

The examples given are illustrative only, as file and I/O device names vary from system to system.

2.3 LOAD (L)

To load a program module, or a program library, enter:

```
L
(filename)
```

where (filename) is the name of the file containing the desired program or library.

Example:

```
*
L
FFT.RB
```

This example loads the program contained on file FFT.RB.

In loading routines, the first entry point defined becomes the name of the host source output. An entry point can be made the first entry point by entering a force (F) command without having loaded an object module previously.

2.4 SYMBOLS (S)

To list the global (external and entry) symbols, enter:

```
S
(filename)
```

where (filename) is the name of the file (or I/O device) to receive the symbol listing. The output of the loader map is as follows.

```
HIGH = aaaaaa
SYMBOL TABLE
SYMBOL              VALUE

ssssss              nnnnnn
  .                   .
  .                   .
  .                   .
```

where:

aaaaaa	Highest program address so far loaded. Normally, the next program is loaded starting at location HIGH+1.
ssssss	Symbol name.
nnnnnn	Symbol value. If undefined, this is the last location loaded which referenced this symbol. If defined as a constant (with the $GLOBAL pseudo-op), this is the value of the constant. If defined as an entry symbol, this is the program source address of the entry symbol.
U	If present, this indicates that the symbol is as yet undefined.

Example:

```
*
S
TTY
```

This example lists the loader symbol table at the terminal. (Some systems, however, may require a different parameter to indicate the user terminal.)

2.5 UNDEFINED (U)

To list at the terminal any presently undefined global symbols, enter:

```
U
(filename)
```

where (filename) is the file to receive the list of undefined symbols. The list format is:

```
ssssss          nnnnnn
```

where ssssss is the symbol name, and nnnnnn is the location of the last program instruction which referenced the symbol.

Example:

```
*
U
TTY
```

This example lists the names of any undefined symbols at the terminal.

2.6 NEXT BASE (B)

To specify a base address at which to load the next program, enter:

```
B
(loc)
```

where (loc) is the location specified.

Example:

```
*
B
200
```

This example sets the next location loaded to location 200.

2.7 RESET (R)

To reset LNK100, enter:

```
R
```

This reinitializes the program to its initial state. The symbol table is cleared, any previously loaded programs are disregarded, and the next location is set to zero. This command must be given following a fatal error.

2.8 FORCE (F)

To force loading of a program module from a library, enter:

```
F
(name)
```

where (name) is the name of the symbol to be forced. This command enters (name) into the symbol table as an external symbol. This causes the loading of a library program which has (name) as an entry symbol.

Example:

```
*
F
DOTPRD
```

This example forces the loading, from any subsequently loaded library file, of any program that defines the symbol DOTPRD as an entry symbol.

2.9 MEMORY (M)

To get the address of the highest program source memory location so far loaded, enter:

```
M
```

The information is printed as follows:

```
HIGH = aaaaaa
```

where aaaaaa is the highest address so far loaded.

2.10 END (E)

To end a load module and generate the completed load module for use with DBG100 or SIM100, enter:

```
E
(filename)
```

where (filename) is the name of the file to receive the loader output. The output is a core image which can be loaded by DBG100 and executed by either the simulator SIM100 or the hardware.

LNK100 lists the following information at the user terminal:

```
HIGH = aaaaaa
```

where aaaaaa is the highest program address loaded. If any symbols were still undefined, LNK100 lists:

```
(num) UNDEFINED SYMBOLS
```

where (num) is the number still undefined. A value of 0 is used in linking these undefined symbols.

Example:

```
*
E
SAVE
```

This example stores the completed load module into file SAVE.

The E (or A) command causes links between global symbols in the completed load module to be frozen. The load module can be generated again (with another E or A command), but no further links can be added (with an L command).

To work on another load module, a reset (R) command must be given to clear the linker.

2.11 END WITH ASSEMBLY CODE (A)

To end a load module and generate the completed load module as host computer code (for use with APX100), enter:

```
A
(filename)
```

where (filename) is the name of the file to receive the loader output. This output is a host FORTRAN (or possibly assembly) language subroutine, which is the linkage between host computer FORTRAN calls and the FPS-100 executive. The FPS-100 code in the load module follows the host subroutine and is in the form of data statements.

Information concerning the highest address loaded into, and any undefined symbols, are listed at the user terminal as described previously for the E command.

2.12 NUMBER RADIX (N)

To set the radix for numeric input/output to and from the user terminal, enter:

```
N
(radix)
```

where (radix) is either 8 (for octal), 10 (for decimal), or 16 (for hexadecimal). The default radix for user I/O is set to one of these choices at installation.

2.13 EXIT (X)

To exit to the operating system, enter:

```
X
```

Notice that the X command does not cause any output. The E or A command must be used to generate a load module.

2.14 AN EXAMPLE LOADING SESSION

This section contains an example loading session.

```
LNK100                          Call Linker
  LNK100  REL.  1.00 , 9/01/79
 *
L
PROG1.OBJ                       Load PROG1.OBJ
 *
U                               List any undefined symbols
TTY                             at the terminal
  DIV      000004  U
 *                              DIV subroutine is undefined
L                               Load DIV from subroutine
APLIB                           library
  LOAD COMPLETE
 *
S                               List global external and entry symbols
TTY
  HIGH=000042


  SYMBOL TABLE

  SYMBOL VALUE
  PROG1  000000
  DIV    000007
 *
E
PROG1.SIM100
 PROG1      HIGH=000042         Create PROG1.SIM100 to run on the simulator
 *
A
PROG1.SOURCE                    Create PROG1.SOURCE (host FORTRAN or assembler)
 PROG1      HIGH=000042         to run on host system
 *
X                               End (return to the operating system)

END LNK100
```

CHAPTER 3

OBJECT MODULES

3.1 INTRODUCTION

The relocatable object modules produced by the ASM100 assembler, which are used as input to LNK100, consist of numbers written as octal characters. Unlike most relocatable binary code, this code can be displayed at a terminal and edited with an ordinary text editor.

The relocatable object code is divided into a series of blocks. The order in which the blocks appear, if each type is present, is generally as follows (the octal block type number is in parentheses):

1. library start block (6)
2. title block (3)
3. data block descriptor blocks (10)
4. parameter block (12)
5. data block initialization blocks (11)
6. alternate entry block (13)
7. entry block (4)
8. code blocks (0)
9. external block (5)
10. end block (1)
11. library end block (7)

NOTE

The data block description, parameter, data block initialization, and relocatable entry blocks are not processed by LNK100. If encountered, LNK100 ignores these blocks.

An object module must contain a title block and an end block. The presence and ordering of other types of blocks depend on the particular program.

The first line of each block is a block header, which describes the remainder of the block. The block header is easily identified because it contains the characters "***" followed by the name of the block. The remainder of the block contains data records.

Blocks are described in the following paragraphs, in order of their block type numbers (again, in octal).

3.2 CODE BLOCK (0)

Header:

0 count location ***CODE

count	This specifies the number of data records that follow.
location	This specifies the address relative to the start of the routine where the code is loaded.

Data record:

* $code_1$ $code_2$ $code_3$ $code_4$ flddes type arg ...flddes type arg

*	The asterisk at the beginning of the line is optional, but it is present if any field of the instruction is to be relocated or contains an external reference.
$code_1$-$code_4$	These are four 16-bit unsigned octal numbers. They make up the code for one FPS-100 instruction word.

The optional triples at the end of the data record are used to define the fields in the instruction word that are to be relocated.

flddes	This is the field designator, specifying which field to relocate. Possible values are: 0 value field
type	This specifies the type of relocation. Possible values are: 1 program source relocatable 2 external reference (absolute) 3 DB reference 4 relocation via the .LOCAL block of a subroutine 5 external reference (relative) Type 5 is the only type processed by LNK100.
arg	The value of arg depends on the type specification. If type is 2, 4, or 5, arg specifies an external. If type is 3, arg specifies a data block. If type is 1, arg is ignored. Type 5 is the only type processed by LNK100.

3.3 END BLOCK (1)

Header:

1 ***END

Data record:

title

title	This specifies the title of the routine (the same as appears in the title block).

3.4 TITLE BLOCK (3)

Header:

```
3    ***TITLE
```

Data record:

```
title
```

title	This specifies the title of the routine.

3.5 ENTRY BLOCK (4)

Header:

```
4   count   ***ENTRY
```

count	This specifies the number of data records that follow.

Data record:

```
symbol   value   type   paramnum
```

symbol	This is a six-character entry symbol.
value	This specifies the value of the symbol. If the symbol is relocatable, this value is relative to the start of this routine.
type	This indicates the type of symbol. Possible values are: 0 absolute 1 relocatable (ignored by LNK100)
paramnum	This indicates the number of parameters associated with the entry point. It is not present if type is 0.

3.6 EXTERNAL BLOCK (5)

Header:

```
5   count   ***EXT
```

count	This specifies the number of data records that follow.

Data record:

```
symbol
```

symbol	This is a six-character external symbol name.

3.7 LIBRARY START BLOCK (6)

Header:

```
6    ***LSB
```

3.8 LIBRARY END BLOCK (7)

Header:

```
7    ***LEB
```

3.9 DATA BLOCK DESCRIPTOR BLOCK (10)

Header:

```
10   count   symbol   dest    ***DBDB
```

Data record:

```
type   number
```

3.10 DATA BLOCK INITIALIZATION BLOCK (11)

Header:

```
11   count   ***DBIB
```

Data record:

```
id   location   type   rc   value
```

3.11 FORMAL PARAMETER BLOCK (12)

Header:

```
12   count   ***FPB
```

Data record:

```
type   dest   size
```

3.12 ALTERNATE ENTRY BLOCK (13)

Header:

```
13   count   ***AENTRY
```

count	This specifies the number of data records that follow.

Data record:

```
symbol   value   type   paramnum
```

symbol	This is a six-character entry symbol.
value	This specifies the value of the symbol. If the symbol is relocatable, this value is relative to the start of this routine.
type	This indicates the type of symbol. Possible values are: 0 absolute 1 relocatable
paramnum	This indicates the number of parameters associated with the entry point. It is not present if type is 0.

CHAPTER 4

GENERATING LNK100 OUTPUT

4.1 INTRODUCTION

LNK100 generates two types of output. The E command generates a load module for use with SIM100 and DBG100; the A command generates a load module for use with APX100. This chapter shows how to produce each.

4.2 SOURCE PROGRAM AND OBJECT MODULE

Figure 4-1 contains an ASM100 subroutine for which both A and E output is to be generated. This subroutine is used as input to the ASM100 assembler. ASM100 generates the object module contained in Figure 4-2. The object module is used as input to LNK100.

```
        $TITLE VCADD
        $ENTRY VCADD, 4
"VECTOR ADD
"ADDS VECTOR A TO VECTOR B AND PUTS THE RESULT INTO VECTOR C
"C(M) = B(M) + A(M)      FOR M = 0 TO N-1

"S-PAD PARAMETERS
        A    $EQU   0                    "BASE ADDRESS OF VECTOR A
        B    $EQU   1                    "BASE ADDRESS OF VECTOR B
        C    $EQU   2                    "BASE ADDRESS OF C
        N    $EQU   3                    "NUMBER OF ELEMENTS IN C

VCADD:  MOV A,A; SETMA                   "FETCH A(0)
        MOV B,B; SETMA                   "FETCH B(0)
        DEC C; DPX(0)<MD                 "SAVE A(0)
LOOP:   INC A; SETMA                     "FETCH A(M+1)
        INC B; SETMA;                    "FETCH B(M+1)
          FADD DPX(0),MD                 "B(M) + A(M)
        DPX(0)<MD;                       "SAVE A(M+1)
          DEC N; FADD                    "  SEE IF DONE?????
        MI<FA; INC C; SETMA;             "STORE C(M)
          BNE LOOP                       "BRANCH IF NOT DONE
        RETURN
        $END
```

Figure 4-1 ASM100 Source

```
     3       ***TITLE
VCADD
    13       1       ***AENTRY
VCADD        0       2       4
     0      10       0       ***CODE
 40000       0       0       60
 40104       0       0       60
  1210       0   45004        0
  1100       0       0       60
  1105  124000     400       60
  1215  100000   45004        0
  1110     655       0      160
     0     340       0        0
     1     ***END
VCADD
```

Figure 4-2 Object Module

4.3 LOAD SESSION

The following procedure is used to load the object module shown in Figure 4-2. (The procedure to call LNK100 varies according to the host operating system but normally consists of entering the name LNK100.)

```
LNK100                                  Call LNK100

 LNK100     Version Date
 *

L                                       Load the object module which
VCADD.O                                 resides on file VCADD.O.
 LOAD COMPLETE
 *

E                                       Generate a load module for
SIMMOD                                  use with SIM100/DBG100.
 VCADD     HIGH=000007
 *

A                                       Generate a load module
LMOD                                    for use with APX100.
 VCADD     HIGH=000007
 *

X                                       Exit to the host operating
                                        system.
 END LNK100
```

4.4 OUTPUT FROM THE E COMMAND

The load session shown in section 4.3 generated a file SIMMOD with the E command. This file is a load module which can be used with a simulated FPS-100 (SIM100) or with the actual hardware for debugging (DBG100). Figure 4-3 contains this load module. The first line indicates that the program contains eight program words.

```
    8.
16384.  00000.  00000.  00048.
16452.  00000.  00000.  00048.
00648.  00000.  18948.  00000.
00576.  00000.  00000.  00048.
00581.  43008.  00256.  00048.
00653.  32768.  18948.  00000.
00584.  00429.  00000.  00112.
00000.  00224.  00000.  00000.
```

Figure 4-3 E Command Load Module

4.5 OUTPUT FROM THE A COMMAND

The load session shown in section 4.3 generated a file LMOD with the A command. This file is a load module which can be transferred to the FPS-100 with APX100 routines for execution there. This load module, which was produced on a Prime computer system, is shown in Figure 4-4. However, output is different for different host operating systems. For some systems, assembly code output is produced.

```
C* VCADD
      SUBROUTINE VCADD (I
     X    1,I 2,I 3,I
     X    4)
      INTEGER CODE(   33)
      INTEGER I 1,J 1
      INTEGER I 2,J 2
      INTEGER I 3,J 3
      INTEGER I 4,J 4
      INTEGER SLIST(16)
      COMMON /SPARY/SLIST
      EQUIVALENCE (J 1,SLIST( 1))
      EQUIVALENCE (J 2,SLIST( 2))
      EQUIVALENCE (J 3,SLIST( 3))
      EQUIVALENCE (J 4,SLIST( 4))
      DATA CODE(1) /   8/
      DATA CODE(    2),CODE(    3),CODE(    4),CODE(    5)/
     X  :040000,:000000,:000000,:000060/
      DATA CODE(    6),CODE(    7),CODE(    8),CODE(    9)/
     X  :040104,:000000,:000000,:000060/
      DATA CODE(   10),CODE(   11),CODE(   12),CODE(   13)/
     X  :001210,:000000,:045004,:000000/
      DATA CODE(   14),CODE(   15),CODE(   16),CODE(   17)/
     X  :001100,:000000,:000000,:000060/
      DATA CODE(   18),CODE(   19),CODE(   20),CODE(   21)/
     X  :001105,:124000,:000400,:000060/
      DATA CODE(   22),CODE(   23),CODE(   24),CODE(   25)/
     X  :001215,:100000,:045004,:000000/
      DATA CODE(   26),CODE(   27),CODE(   28),CODE(   29)/
     X  :001110,:000655,:000000,:000160/
      DATA CODE(   30),CODE(   31),CODE(   32),CODE(   33)/
     X  :000000,:000340,:000000,:000000/
      J 1=I 1
      J 2=I 2
      J 3=I 3
      J 4=I 4
      CALL APEX(CODE,    0,SLIST,    4)
      RETURN
      END
```

Figure 4-4 A Command Load Module

The source code generated by LNK100 consists of four basic parts: the SUBROUTINE statement, SLIST array, CODE array, and the APX100 call.

The subroutine statement contains the routine's arguments, the number of which corresponds to the s-pad parameter on the first $ENTRY psendo-op in the corresponding ASM100 code. The subroutine name is the same as the symbol on the $ENTRY pseudo-op in the corresponding ASM100 source code. When the user calls VCADD, control is passed to this host source routine. The arguments are transferred to the SLIST array. These are addresses of data already transferred to the FPS-100 via APPUT calls in the user-written host FORTRAN program. The code array contains the load module created by the user, in this case, VCADD. The first element of the array is the number of FPS-100 program source words; the following values correspond to the actual microcode.

The APX100 calls cause the microcode to be loaded into FPS-100 program source memory unless it still resides there from a previous call. The argument values are placed in their respective s-pad registers (16 is maximum), and control is transferred to the routine entry point in the FPS-100.

CHAPTER 5

ERROR MESSAGES

5.1 GENERAL INFORMATION

Any deviation from the prescribed command syntax causes LNK100 to display a ? at the user terminal. The illegal command is ignored, and LNK100 displays a * to indicate its readiness to accept a new command.

If a specified file cannot be found or is otherwise unavailable for use, the message:

```
FILE NOT FOUND!!!
```

is displayed and the command is ignored.

The specific error messages displayed by LNK100 are the result of loading errors detected during execution of an L (load) command. There are two classes of loading errors:

F - Fatal	Reinitialization of the loader (the R command) is required before loading can continue.
W - Warning	An advisory message indicating a possible error.

Any fatal error detected during loading causes immediate termination of the L (load) command following the error message. If the user attempts to execute another L command, the program displays the message:

```
RESET!!!
```

and ignores the command. After reinitializing the loader (R command), the user must reload any programs loaded up to that point.

5.2 MESSAGES

Following are the error messages, along with notes of explanation for each:

F SYMBOL TABLE OVERFLOW

The loader symbol table is full. The only recourse is to recompile LNK100 with a longer symbol table.

F PROGRAM MEMORY OVERFLOW nnnnnn

An attempt was made to load the upper limit of program source memory. The load module is too large to fit in program source memory. nnnnnn is the memory location involved.

F OVERWRITE nnnnnn

An attempt was made to overwrite a previously loaded program memory location. The loader does not permit any given program memory location to be loaded more than once. nnnnnn is the program memory location involved.

F ILLEGAL BLOCK TYPE nnnnnn

An illegal relocatable object code block type was encountered. The file specified does not contain legal object code. nnnnnn is the illegal block type, as read from the block header in question.

F TOO MANY EXTERNALS

The loader table of links is full. The only recourse is to recompile LNK100 with a longer LINKS array.

W MULTIPLE ENTRY

An $ENTRY symbol having the same name as one already defined was encountered during a load. The name and value of the symbol is listed at the terminal as follows:

```
ssssss          nnnnnn
```

where ssssss is the symbol name and nnnnnn the symbol value (refer to section 2.4). The loader proceeds by ignoring the latest definition.

W MISSING OR IMPROPER ENTRY

The user attempted to produce host assembly code (an A command) from a load module and the load module did not have any entry points (defined entry global symbols).

W $DBIB(S) IGNORED IN BINARY

The user attempted to load an FTN100 binary or a binary loaded from a library containing FTN100 entry points.

W $DBDB(S) IGNORED IN BINARY

The user attempted to load an FTN100 binary or a binary loaded from a library containing FTN100 entry points.

APPENDIX A

SUMMARY OF LNK100 COMMANDS

This appendix contains a summary of LNK100 commands. The abbreviations used in this section are listed in Table A-1. The commands are listed in Table A-2.

Table A-1 Abbreviations

Abbreviation	Meaning
(filename)	Name of a file, as appropriate for the host operating system being used.
(loc)	A location, octal or hexadecimal, as appropriate.
(name)	A symbol name, six characters or less.

Table A-2 Command Summary

Command	Effect
L (filename)	Load the program in file (filename); link with previously loaded programs.
S (filename)	Copy the loader symbol table to file (filename).
U (filename)	Copy any undefined symbols to file (filename).
B (loc)	Set LNK100 to load the next program at location (loc).
R	Reset the loader.
F (name)	Force loading of a program defining symbol (name) from any subsequent program libraries loaded.
M	list the highest program memory location used.
E (filename)	End the loading session; store the resultant load module into file (filename).
A (filename)	End the loading session; generate host computer assembly code for use with APX100 into file (filename).
N (number)	Set the radix for numeric user console I/O to either 8, 10, or 16.
X	Exit to the operating system.

INDEX

READERS COMMENT FORM

Your comments will help us improve the quality and usefulness of our publications. To mail: fold the form in three parts so that Floating Point Systems mailing address is visible, then seal.

Title of document _____

Name/Title _____ Date _____

Firm _____ Department _____

Address _____

Telephone _____

I used this manual...

- ☐ as an introduction to the subject
- ☐ as an aid for advanced training
- ☐ to instruct a class
- ☐ to learn operating procedures
- ☐ as a reference manual
- ☐ other _____

I found this material...

	Yes	No
accurate/complete	☐	☐
written clearly	☐	☐
well illustrated	☐	☐
well indexed	☐	☐

Please indicate below, listing the pages, any errors you found in the manual. Also indicate if you would have liked more information on a certain subject.

First Class
Permit No.A-737
Portland,
Oregon

BUSINESS REPLY

No postage stamp necessary if mailed in the United States

Postage will be paid by:

FLOATING POINT SYSTEMS, INC.

P.O. Box 23489

Portland, Oregon 97223

Attn: Technical Publications

FLOATING POINT
SYSTEMS, INC.

CALL TOLL FREE 800-547-1445
P.O. Box 23489, Portland, OR 97223
(503) 641-3151, TLX: 360470 FLOATPOINT PTL