ISIS-II
FORTRAN-80 COMPILER
OPERATOR'S MANUAL

Manual Order Number: 9800480B

Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

Printed in U.S.A./A106/0879/5K BL

PREFACE

This manual describes operating procedures for the ISIS-II FORTRAN-80 Compiler and run-time libraries. The compiler translates FORTRAN-80 source code into relocatable object code for the 8080 and 8085 microprocessors. The manual also describes the steps needed to execute the compiled program including all necessary linkage, relocation, and run-time requirements.

Manual Organization

Chapters 1 through 4 of this manual apply to all users, and describe compiler features and the general procedures for linking, locating, and execution of programs. The subsequent chapters (5-7) provide instructions applicable to particular run-time environments. The Appendices supply reference information pertinent to all systems.

The manual contains the following chapters and appendices:

"Chapter 1. Compiler Overview," which gives a general description of the compiler, its input and output files, and the run-time libraries provided with the compiler, plus a step-by-step outline of the compilation, linking, locating, and execution of a small sample program.

"Chapter 2. Compiler Controls," which describes each of the controls that may be used to modify the interpretation of source files, the use of compiler resources, and the format and content of output files.

"Chapter 3. Listing Formats," which explains each part of the listed output available from the compiler.

"Chapter 4. Program Linkage, Relocation, and Execution," which gives general instructions for linking, locating, and executing programs.

"Chapter 5. iSBC 310 Interface," which gives special instructions for using the run-time software interface that allows an iSBC 310 High-Speed Math Unit to be used for greater speed in performing floating-point operations.

"Chapter 6. Preparing Programs to Run Under RMX/80," which provides information applicable to running FORTRAN programs in the RMX/80 environment.

"Chapter 7. Preparing Programs to Run Without ISIS-II or RMX/80," which gives instructions needed in non-ISIS, non-RMX run-time environments, including instructions for programming custom low-level I/O drivers.

"Appendix A. The Compiler and the FORTRAN Language," which summarizes the limitations and extensions to the standard FORTRAN language assumed by the FORTRAN-80 compiler.

"Appendix B. Error Messages," which lists the error messages and error codes which may occur at compile time and at run time, including (for reference) errors that may be detected by ISIS-II, RMX/80, and the LINK and LOCATE programs. Information is also provided on how to program custom error handlers for floating-point operations.

"Appendix C. Number Formats," which explains the internal formats for integer and floating-point numbers assumed by the FORTRAN compiler, together with the schemes used for rounding floating-point numbers and exponent wraparound.

"Appendix D. Summary of LINK Options," which provides a diagram of all libraries that may be specified for various run-time environments, in the order in which they must be given in the LINK command.

"Appendix E. Execution Speeds and Stack Requirements for Floating-Point Operations," which lists the execution speed and minimum stack requirement for each FORTRAN operation and intrinsic function that involves floating-point (REAL) numbers.

"Appendix F. Providing Reentrancy for Non-RMX Floating-Point Libraries," which explains how two library procedures can be called to effect reentrancy for floating-point operations in non-RMX environments.

Related Publications

No discussion of the FORTRAN programming language is provided here. Please refer to the document

FORTRAN-80 Programming Manual 9800481

Use of the compiler and run-time libraries requires the ISIS-II software. (ISIS-II is the diskette operating system facility of Intel's Intellec or Intellec Series II Microcomputer Development System). This facility is described in the document

ISIS-II User's Guide 9800306

Object modules produced by the compiler may be run in the ISIS-II environment, under the iSBC 80 Real-Time Multitasking Executive (RMX/80), or without either operating system. The *ISIS-II User's Guide,* referenced above, provides all necessary information on the ISIS-II run-time environment. Use of RMX/80 is described in the document

RMX/80 User's Guide 9800522 (Rev. B or later)

If you are using the iSBC 310 High-Speed Math Unit interface option for faster floating-point operations, you will find information on the operation of this unit in the document

iSBC 310 High-Speed Mathematics Unit Hardware Reference Manual 9800410

For further information on linking FORTRAN and non-FORTRAN procedures together, refer to the document

How to use FORTRAN with other Intel Languages 9800778

(Application Note AP-44)

CONTENTS

ILLUSTRATIONS

CHAPTER 1
COMPILER OVERVIEW

The ISIS-II FORTRAN-80 Compiler converts FORTRAN source code into machine-executable form. It translates FORTRAN program units into relocatable object code modules for the 8080 and 8085 processors and (depending on the output options selected) can produce the object code module, a listing of the source and compiled code, and a symbol cross-reference listing.

NOTE

Relocatable object modules produced by version 1.0 of FORTRAN-80 are not compatible with this release. All program units must be recompiled using version 2.0 of the compiler before being linked with the current FORTRAN libraries and with other program units compiled using version 2.0.

The compiler runs under the ISIS-II operating system. Object modules produced by the compiler may be run in the ISIS-II environment, under the iSBC 80 Real-Time Multitasking Executive (RMX/80), or in a stand-alone environment without either operating system. Supporting the compiler is a set of relocatable library modules that supply a run-time environment including floating-point arithmetic and mathematical functions, sequential or direct access input/output (with or without formatting), and elementary interaction with ISIS-II or RMX/80. Floating-point operations may be performed by software routines supplied with the compiler or, alternatively, via a software interface provided for the iSBC 310 High-Speed Mathematics Unit.

Because of the variety of hardware and software configurations in which FORTRAN-80 programs can be run, a number of different run-time support libraries are provided. Which libraries you select at link time will depend upon your particular run-time environment.

In discussing FORTRAN-80, it is important to distinguish between the *compile-time* environment—that in which the FORTRAN compiler runs to translate your source program segments (and in which the resulting relocatable object program is linked and located) and the *run-time* environment in which your linked and located object program is executed. These two environments are defined in the following paragraphs.

Compile-Time Environment

No matter what your final application, the following environment is required to compile, link, and locate your FORTRAN programs:

- An Intellec or Intellec Series II Microcomputer Development System with 64K RAM Memory
- Console Device (TTY or CRT)
- Diskette unit with at least two drives
- The ISIS-II software

Run-Time Environment

Once your program modules have been compiled, linked, and located, you may run them on any 8080- or 8085-based system that is appropriate to your application. Your run-time environment may be an Intellec system and ISIS-II (the same environment in which your programs are compiled); an iSBC 80/10, 80/20, or 80/30 system running under RMX/80; or an 8080- or 8085-based iSBC or custom-designed system with no operating system software at all. Any of these environments can optionally be configured to include an iSBC 310 High-Speed Math Unit for faster floating-point operations. All that is *required* at run time is an 8080 or 8085 processor with hardware support and enough memory for your application.

Input/Output Files

Source File

The compiler expects a diskette-resident source file consisting of one or more *program units.* A program unit may be a main program or a BLOCK DATA, FUNCTION, or SUBROUTINE subprogram. The source file can also contain compiler *controls* embedded in the FORTRAN source code. These controls direct the exact operation of the compiler. In addition to the source file, the compiler will read any files specified by INCLUDE compiler controls (see Chapter 2).

A FORTRAN program, as defined in the *FORTRAN-80 Programming Manual,* consists of one and only one FORTRAN main program and any number of FUNCTION, SUBROUTINE, and/or BLOCK DATA subprograms. Under some circumstances, however, you may not need a FORTRAN main program at all—for instance, if your main program is in assembly language or PL/M, or if you are coding FORTRAN tasks to run under RMX/80. The FORTRAN-80 compiler does *not* require that your source file include a main program.

Object File

The primary output from the FORTRAN compiler is a file containing the relocatable object code. This file is linked with the FORTRAN run-time library routines (as well as user-supplied relocatable files) to produce a single, relocatable object file. This file is then located to form an absolute module ready for execution.

Each source file submitted to the compiler produces one object file. Each program unit in the source file produces one object module in the object file. Object modules have the same name as their corresponding input program unit. In the case of an unnamed main program or BLOCK DATA subprogram, the module names assigned are @MAIN and @BLOCKDATA, respectively.

List File

The compiler list file consists of the program listing and symbol cross-reference listing. The program listing can include introductory headings, a pseudo assembly-language listing of the object code, the source-code listing, symbol and cross-reference listings, and error messages. List file data is broken out separately for each program unit and is directed to the file or device specified by the PRINT compiler control (Chapter 2).

If you specify the CODE compiler control, the program listing includes a pseudo assembly-language version of all relocatable object code generated by the compiler. The source text is listed if the LIST control is active. Error messages are listed unless the NOPRINT control has suppressed all printed output. (If an error is detected within a specific source statement, that statement is printed even if the NOLIST compiler control is in effect.) The attributes of symbols are listed if the SYMBOLS control is active.

If you specify the XREF compiler control, a symbol cross-reference listing is produced. This is followed by the compilation summary.

See Chapter 2 for more complete descriptions of the various compiler controls, and Chapter 3 for details of the listing format.

Compiler Overlay/Scratch Files

In general, you need not concern yourself with the format of the compiler's overlay files or temporary scratch files. You must be aware of their existence, however, so you do not accidentally use one of these reserved file names for a file of your own.

The FORTRAN compiler is invoked by calling file 'FORT80,' which is the first phase of the compiler. This phase initializes the compiler and then calls other phases as overlays. These overlays are named 'FORT80.OVn,' where 'n' is a digit 0-4.

Compiler scratch files are deleted automatically by the compiler when they are no longer needed. The file names reserved for them are:

FORTT1.TMP
FORTT2.TMP
FORTXR.TMP
FORTAT.TMP
FORTER.TMP

FORTRAN Libraries

Several FORTRAN libraries are provided to do various mathematical and input/output operations. Those libraries appropriate to your particular run-time environment are to be linked to your program after it is compiled, using either the LINK command or the SUBMIT command. See the sample program development in the next subsection for an example. The libraries and various linking strategies are discussed in detail in Chapter 4. The following are the libraries provided:

F80RUN.LIB	Integer arithmetic, array indexing, and miscellaneous routines (all environments)
F80ISS.LIB	Input/output for the non-RMX environment (ISIS-II or stand-alone environment)
F80NIO.LIB	External reference library for programs using no FORTRAN input or output except port I/O (all environments)
FPEF.LIB	Floating-point intrinsic functions (all environments)
FPSOFT.LIB	Floating-point arithmetic library for the non-RMX (ISIS-II or stand-alone) environment
FPHARD.LIB	Floating-point interface to the iSBC 310 math unit for the non-RMX (ISIS-II or stand-alone) environment
FPNULL.LIB	External reference library for programs using no floating-point operations (all environments)

PLM80.LIB	Support for library modules coded in PL/M (all environments)

A separate package, the FORTRAN-80 Run-Time Package for RMX/80 Systems (iSBC 801), contains the additional libraries necessary to run FORTRAN programs in the RMX/80 environment. These libraries are described in detail in Chapter 6. The following libraries are included:

F80RMX.LIB	Input/output for the RMX/80 environment
FPSFTX.LIB	Floating-point arithmetic library for the RMX/80 environment
FPHRDX.LIB	Floating-point interface to the iSBC 310 math unit for RMX/80, iSBC 80/20 and 80/30 systems
FPHX10.LIB	Floating-point interface to the iSBC 310 math unit for RMX/80, iSBC 80/10 systems
F80NTH.LIB	External reference library for systems running under RMX/80 without the Terminal Handler
F80NDS.LIB	External reference library for systems running under RMX/80 without the Disk File System

Procedures in run-time support libraries have names that begin with either 'FQ0' or FQF.' You should avoid using program unit names beginning with these characters.

Preparing FORTRAN System Diskettes

The FORTRAN-80 compiler and libraries as delivered do not reside on system diskettes. Before following the procedure given in the next section to develop the sample program, you will generally want to create one or two FORTRAN system diskettes by deleting some other programs from existing ISIS-II system diskettes, then copying onto the diskettes the FORTRAN compiler files and run-time libraries. (The compiler can be run from a diskette on :F1: with an ISIS-II system diskette on :F0:, but this practice is usually inconvenient unless you have more than two diskette drives.) If you are using single-density diskettes, you will need two system diskettes, one for the compiler and one for the run-time libraries; for double-density diskettes, you can fit the compiler and other libraries on the same system diskette.

Figures 1-1, 1-2, and 1-3 show how you might create two single-density system diskettes suitable for use in compiling, linking, and locating FORTRAN programs. Figure 1-1 shows the directory listing for a standard ISIS-II (version 3.4) system diskette. Figure 1-2 shows an ISIS-II SUBMIT file that could be used to delete files from and add files to the standard system diskette, producing a system diskette containing the FORTRAN compiler; the figure also shows a directory listing of the files on the resulting diskette. Figure 1-3 shows a SUBMIT file that similarly converts another standard system diskette to one suitable for linking and locating programs with the FORTRAN libraries, and a directory listing of the files on the resulting diskette. Both SUBMIT files assume that a copy of the FORTRAN compiler and run-time libraries—e.g., the product as delivered—is on drive :F1:.

In both cases, you may either copy the whole SUBMIT file onto one of your diskettes and execute it with an ISIS-II SUBMIT command *or* enter the ISIS-II commands in the SUBMIT file one by one from the console. Either way, before you begin it is a good idea to prepare back-up copies of all your diskettes.

The two diskettes thus created may be used in the sample program development outlined in the following section. A similar SUBMIT file can be used to convert a single standard *double-density* diskette into one containing the FORTRAN compiler, run-time libraries, LINK, and LOCATE.

```
DIRECTORY OF :F0:IS00AS.SYS
NAME    .EXT BLKS    LENGTH ATTR
ISIS    .DIR   26      3200   IF
ISIS    .MAP    3       256   IF
ISIS    .TO    24      2944   IF
ISIS    .LAB    2       128   IF
ISIS    .BIN   94     11740  SIF
ISIS    .CLI   21      2548  SIF
ASM80         107     13374 WSI
ASM80   .OV0   20      2321 WSI
ASM80   .OV1   19      2280 WSI
ASM80   .OV2   18      2091 WSI
ASM80   .OV3  188     23679 WSI
ASXREF         35      4239 WSI
ATTRIB         38      4682 WSI
BINOBJ         28      3399 WSI
COPY           65      8042 WSI
DELETE         37      4506 WSI
DIR            46      5733 WSI
EDIT           56      6999 WSI
FORMAT         49      6093 WSI
HEXOBJ         35      4281 WSI
IDISK          50      6239 WSI
LIB            82     10227 WSI
LINK          114     14298 WSI
LINK    .OVL   29      3491 WSI
LOCATE        108     13505 WSI
OBJHEX         27      3284 WSI
RENAME         21      2487 WSI
SUBMIT         38      4629 WSI
FPAL    .LIB   71      8712 WS
PLM80   .LIB   45      5615 WS
SYSTEM.LIB     24      2846 WS
              1520
1520/2002 BLOCKS USED
```

Figure 1-1. Directory Listing of a Standard ISIS-II Version 3.4 System Diskette

```
ATTRIB *.* W0
DELETE LINK.*,LOCATE,LIB
DELETE ASM80.*,ASXREF
DELETE BINOBJ,HEXOBJ,OBJHEX,IDISK
DELETE *.LIB
COPY :F1:FORT80.* TO *.*
ATTRIB FORT80.OV* I1
ATTRIB *.* W1
```

```
DIRECTORY OF :F0:IS00AS.SYS
NAME    .EXT BLKS    LENGTH ATTR
ISIS    .DIR   26      3200 W IF
ISIS    .MAP    3       256 W IF
ISIS    .TO    24      2944 W IF
ISIS    .LAB    2       128 W IF
ISIS    .BIN   94     11740 WSIF
ISIS    .CLI   21      2548 WSIF
FORT80         36      4394 W
FORT80.OV0    259     32478 W I
FORT80.OV1     54      6752 W I
FORT80.OV2    241     30333 W I
FORT80.OV3    155     19412 W I
FORT80.OV4    156     19523 W I
ATTRIB         38      4682 WSI
COPY           65      8042 WSI
DELETE         37      4506 WSI
DIR            46      5733 WSI
EDIT           56      6999 WSI
FORMAT         49      6093 WSI
RENAME         21      2487 WSI
SUBMIT         38      4629 WSI
              1421
1421/2002 BLOCKS USED
```

Figure 1-2. SUBMIT File to Create a System FORTRAN Compile Diskette, and Directory Listing of the Resulting Diskette

```
ATTRIB *.* W0
DELETE ASM80.*,ASXREF
DELETE BINOBJ,HEXOBJ,OBJHEX,IDISK
DELETE *.LIB
COPY :F1:*.LIB TO *.*
COPY :F1:FLINK TO *
ATTRIB *.* W1
```

```
DIRECTORY OF :F0:IS00AS.SYS
NAME    .EXT BLKS   LENGTH ATTR
ISIS    .DIR   26     3200 W IF
ISIS    .MAP    3      256 W IF
ISIS    .TO    24     2944 W IF
ISIS    .LAB    2      128 W IF
ISIS    .BIN   94    11740 WSIF
ISIS    .CLI   21     2548 WSIF
F80ISS.LIB  420    52835 W
F80NIO.LIB    5      416 W
F80RUN.LIB  119    14936 W
FPEF    .LIB  100    12538 W
FPHARD.LIB   84    10402 W
FPNULL.LIB    3      212 W
ATTRIB       38     4682 WSI
FPSOFT.LIB  109    13638 W
COPY         65     8042 WSI
DELETE       37     4506 WSI
DIR          46     5733 WSI
EDIT         56     6999 WSI
FORMAT       49     6093 WSI
PLM80 .LIB   45     5615 W
LIB          82    10227 WSI
LINK        114    14298 WSI
LINK    .OVL   29     3491 WSI
LOCATE      108    13505 WSI
RENAME       21     2487 WSI
SUBMIT       38     4629 WSI
            1738
1738/2002 BLOCKS USED
```

Figure 1-3. SUBMIT File to Create a System FORTRAN Run-Time Library Diskette (for Linking and Locating), and Directory Listing of the Resulting Diskette

Sample Program Development

The following example shows the normal sequence of operations used to develop a FORTRAN program from system bootstrap to eventual program execution. The steps involved are as follows:

1. Power up the Intellec hardware.
2. Insert an ISIS-II system diskette containing the compiler into Drive 0.
3. Insert a second (data) diskette into Drive 1.
4. Bootstrap the ISIS-II operating system.
5. Enter your source program on Drive 1 using ISIS-II's EDIT program.
6. Compile the program with the FORTRAN compiler.
7. Exchange the compiler system diskette with a system diskette containing LINK, LOCATE, and the FORTRAN libraries. (This step is not necessary if you have a double-density diskette system.)
8. Link and locate the resulting object code program on Drive 1.
9. Execute your program.

Refer to the *ISIS-II User's Guide* to perform the first five steps in the above sequence. This manual explains how to compile, link, and locate programs.

The interactive sequence that follows is assumed to take place at your console terminal. The text in lower case represents your input to the system. The comments on the right are for clarification only, and do not represent material to be entered. This example shows how to create, compile, load, and execute a complete FORTRAN program for the ISIS-II run-time environment using the software floating-point routines.

The sample program assumes that the FORTRAN compiler and run-time libraries have been copied onto ISIS-II system diskettes (two diskettes for single density or one for double density) as described in the previous section. FLINK is an ISIS-II SUBMIT file that automatically links your object file ('myprog.obj' in the example) to the FORTRAN libraries required when the run-time environment is ISIS-II and the software floating-point routines (i.e., no iSBC 310 unit) are used.

Begin by bootstrapping ISIS-II.

```
ISIS-II, V3.4
-edit :f1:myprog.src
```
The system identifies itself.
Call the ISIS editor.

```
ISIS-II TEXT EDITOR, V1.6
NEW FILE
*i
```
Create program.

```
      PROGRAM GREETS
      PRINT 10
   10 FORMAT ('INTEL DELIVERS FORTRAN-80')
      END
$$
*e$$
```
'$' is escape key.
Exit editor.

```
-fort80 :f1:myprog.src
```
Invoke the compiler.

```
ISIS-II FORTRAN COMPILER V2.0
    0 PROGRAM ERROR(S) IN PROGRAM UNIT GREETS
    0 TOTAL PROGRAM ERROR(S)
FORTRAN COMPILATION COMPLETE
```
Compilation over; exchange compiler diskette with second system diskette if single-density system.

```
-submit flink(:f1:myprog.obj,:f1:myprog.lnk)
-LINK :F1:MYPROG.OBJ,F80RUN.LIB, &
**F80ISS.LIB,FPEF.LIB, &
**FPSOFT.LIB,PLM80.LIB &
**TO :F1:MYPROG.LNK
ISIS-II LINKER V2.1
-:F0:SUBMIT RESTORE :F0:FLINK.CS(:VI:)
-locate :f1:myprog.lnk
ISIS-II LOCATER V2.1
-:f1:myprog
INTEL DELIVERS FORTRAN-80
-
```

The program is linked to all FORTRAN libraries. . .

. . .located. . .

. . .and executed.

The compilation list and object files are written by default to a diskette file on the same diskette as the source file (:F1: in this case). By default, they have the same name as the source file except for the extensions LST and OBJ. Thus :F1:MYPROG.LST contains the compilation list file and :F1:MYPROG.OBJ contains the object code produced by compiling :F1:MYPROG.SRC.

This example provides enough information to use the compiler in its normal mode of operation (when your run-time environment is ISIS-II and you are using the software floating-point routines). The remainder of this manual describes additional features of the compiler, linker, and locater and the preparation of programs for other run-time environments.

CHAPTER 2
COMPILER CONTROLS

Operation of the compiler is directed by compiler controls. For example, these controls tell the compiler what kind of listing is to be produced or whether an object file is to be generated. While a large number of controls are available with the FORTRAN compiler, few need be specified for a typical compilation. Most control options have default values corresponding to the most common use of the compiler.

Specifying Compiler Controls

Compiler controls can be specified in two ways:

- As part of the ISIS-II command used to invoke the compiler (that is, at compile-time)
- As control lines in your source file

Controls Specified At Compile-Time

The FORTRAN compiler (FORT80) is invoked by an ISIS-II command. This command includes the name of your source file and any compiler controls you wish to specify. The format of the compiler invocation is

```
[drive] FORT80 source-file [control-list]
```

where the bracketed items are optional.

The 'drive' specified is the diskette drive containing FORT80. If 'drive' is not specified, ':F0:' is assumed.

The 'source-file' specified is the name of the file containing your sequence of FORTRAN program units. This file must reside on a diskette. The name specified can be a 1-6 character file name, a file name followed by a period and 1-3 character extension, or an ISIS-II diskette drive followed by a file name or extended file name.

Examples:

```
FILE20              (filename)
PROG.SRC            (filename.extension)
:F1:ASSMB.SRC       (:drive:filename.ext)
```

The 'control-list' indicates the compiler controls needed for this compilation. These controls are separated by blanks. The control itself consists of a control name, sometimes followed by a parenthesized control parameter.

Examples:

```
-FORT80 :F2:FPROG.SRC CODE XREF DATE (1978JAN15)
-:F1:FORT80 :F2:MYPROG SYMBOLS NOPAGING
```

Control Lines

Control lines embedded in your source file allow selective control over sections of your program. For example, you might want to suppress the compiler listing for certain sections of your program, or to cause page ejects at specific places.

Control lines are recognized in your source file by a dollar sign ($) in column 1.

Examples:

```
$NOCODE XREF PAGELENGTH(50)
$EJECT CODE
```

Primary And General Controls

Controls are classified as either *primary* or *general.* Both classes of controls can be specified when the compiler is invoked or in source file control lines. Control lines containing primary controls must precede all program units in the source file, however, and primary controls cannot be changed within a source program unit. Control lines containing only general controls can appear anywhere in your source file; general controls can be respecified at any time.

Summary Of Controls

The following list shows the controls available, the basic functions they control, and whether they are primary or general (P/G). Default controls are italicized. The remainder of this chapter describes each control in detail.

Controls	P/G	Function Area
OBJECT (source.OBJ)/NOOBJECT	P	Object File
DEBUG/ *NODEBUG*	P	Object File
OPTIMIZE(0)/ *OPTIMIZE(1)*	P	Object File
PRINT (source.LST)/NOPRINT	P	Compiler Listing
LIST/NOLIST	G	Compiler Listing
SYMBOLS/ *NOSYMBOLS*	P	Compiler Listing
CODE/ *NOCODE*	G	Compiler Listing
XREF/ *NOXREF*	P	Cross-Reference Listing
PAGING/NOPAGING	P	Listing Format
PAGELENGTH(*60*)	P	Listing Format
PAGEWIDTH(*120*)	P	Listing Format
DATE	P	Listing Format
TITLE	P	Listing Format
EJECT	G	Listing Format
REENTRANT	P	Procedure Reentrancy
DO77/DO66	P	DO Loop Interpretation
STORAGE(INTEGER*2, LOGICAL*1)	P	Default Data Length
FREEFORM/ *NOFREEFORM*	G	Source Line Format
INCLUDE	G	Source File Inclusion
WORKFILES(*:F1:,:F1:*)	P	Devices for Scratch Files
SAVE	G	Save Control Settings
RESTORE	G	Restore Control Settings

Object File Controls

These controls determine what type of object file is to be produced and where it is to be produced. The controls are:

OBJECT/NOOBJECT
DEBUG/ NODEBUG
OPTIMIZE(0)/OPTIMIZE(1)

OBJECT/NOOBJECT Controls

Type:	Primary
Form:	OBJECT(file)
	NOOBJECT
Default:	OBJECT(source-file.OBJ)

The OBJECT control specifies that one or more object modules are to be created during the compilation. The parameter 'file' is the object file name (an ISIS file name optionally preceded by a drive name as described earlier in this chapter). The NOOBJECT control specifies that no object modules are to be produced.

If neither control is specified, the default is as shown above. In this case, the file name is the same as the name of the source file with the extension OBJ, and the object file is created on the same drive used for the source file.

Example: OBJECT(:F1:FPROG.OBJ)

This example causes the object file FPROG.OBJ to be created on diskette drive :F1:.

DEBUG/NODEBUG Controls

Type:	Primary
Form:	DEBUG
	NODEBUG
Default:	NODEBUG

If an object file has been requested, the DEBUG control specifies that the object module is to contain the name and relative address of each symbol whose address is known at compile-time, plus the statement number and relative address of each source program statement. This information can be used later for symbolic debugging of the source program using the 8080/8085 in-circuit emulators, ICE-80 and ICE-85.

The NODEBUG control specifies that this symbolic debugging information is not to be included in the object module.

OPTIMIZE Controls

Type:	Primary
Form:	OPTIMIZE(0)
	OPTIMIZE(1)
Default:	OPTIMIZE(1)

The OPTIMIZE(1) control specifies that the compiler is free to perform certain time and/or space optimizations on the object program (such as eliminating repetitive evaluation of identical expressions where side effects cannot occur).

The OPTIMIZE(0) control specifies that the compiler is not to perform such optimizations.

Compiler Listing Controls

These controls determine what types of listings are to be produced, what they are to contain and on which device they are to appear. The controls are:

PRINT/NOPRINT
LIST/NOLIST
SYMBOLS/NOSYMBOLS
CODE/NOCODE
XREF/NOXREF

PRINT/NOPRINT Controls

Type:	Primary
Form:	PRINT(file)
	NOPRINT
Default:	PRINT(source-file.LST)

The PRINT control specifies that printed output is to be produced and names the file or output device to receive the printed output. The 'file' specified can be any name acceptable to ISIS-II. The NOPRINT control specifies that no printed output is to be produced, and overrides the LIST, SYMBOLS, CODE, XREF, and EJECT controls.

If neither control is specified, the default is as shown above. In this case, the file name is the same as the name of the source file with the extension LST, and printed output is directed to the diskette drive used for source input.

Example: PRINT(:LP:)

This example causes printed output to be directed to the line printer.

LIST/NOLIST Controls

Type:	General
Form:	LIST
	NOLIST
Default:	LIST

The LIST control specifies that listing of the source program is to begin or resume with the next source line read. The NOLIST control suppresses listing of the source program until the next occurrence of a LIST control.

When LIST is in effect, all input lines from the source file or from an INCLUDE file, including control lines, are listed. When NOLIST is in effect, only source lines associated with error messages are listed.

The NOPRINT control overrides the LIST control. If NOPRINT is in effect, no listing whatsoever is produced.

SYMBOLS/NOSYMBOLS Controls

Type:	Primary
Form:	SYMBOLS
	NOSYMBOLS
Default:	NOSYMBOLS

The SYMBOLS control specifies that a listing of all symbols (and their attributes) in the subsequent program unit(s) be printed. The NOSYMBOLS control suppresses this listing.

The NOPRINT control overrides the SYMBOLS control.

CODE/NOCODE Controls

Type:	General
Form:	CODE
	NOCODE
Default:	NOCODE

The CODE control causes the compiler to print a listing of the object code generated by the compiler in a form resembling 8080/8085 assembly language. The listing begins with the object code generated by the next following FORTRAN statement. The NOCODE control suppresses printing of the listing until the next occurrence of a CODE control.

The NOPRINT control overrides the CODE control.

XREF/NOXREF Controls

Type:	Primary
Form:	XREF
	NOXREF
Default:	NOXREF

The XREF control specifies that a cross-reference listing of all symbols, with their attributes and the locations at which they are referenced in the subsequent source program unit(s), is to be produced. The NOXREF control suppresses the cross-reference listing.

The NOPRINT control overrides the XREF control.

Listing Format Controls

These controls determine the format of the compiler output listing. The controls are:

PAGING /NOPAGING
PAGELENGTH
PAGEWIDTH
DATE
TITLE
EJECT

PAGING/NOPAGING Controls

Type:	Primary
Form:	PAGING
	NOPAGING
Default:	PAGING

The PAGING control specifies that printed output is to be formatted onto pages separated by page ejects. The pages are headed with the compiler identification, a page number, and possibly a user-specified title and/or date. Page numbering begins at '1' for each program unit.

The NOPAGING control specifies that page ejection, page heading, and page numbering is not to be performed, except at the beginning of the listing. Thus the listing appears on one long 'page,' as would be suitable for a slow printer without a page-eject mechanism. NOPAGING nullifies the effect of the EJECT control.

PAGELENGTH Control

Type: Primary
Form: PAGELENGTH(length)
Default: PAGELENGTH(60)

The PAGELENGTH control specifies the maximum number of lines to be printed per page (if the PAGING control is set). 'Length' is a nonzero, unsigned, decimal integer; '4' is the minimum length that can be specified.

The number of lines specified is assumed to include page headings.

PAGEWIDTH Control

Type: Primary
Form: PAGEWIDTH(width)
Default: PAGEWIDTH(120)

The PAGEWIDTH control specifies the maximum line width to be used for listed output. 'Width' is a nonzero, unsigned, decimal integer; its minimum value is 60 and its maximum value is 132.

DATE Control

Type: Primary
Form: DATE(date)
Default: None

The DATE control specifies the date to be included in the page heading if the PAGING control is active. The 'date' parameter is any sequence of nine or fewer characters not containing parentheses.

Example: DATE(25 JAN 78)

TITLE Control

Type: Primary
Form: TITLE('title')
Default: None

The TITLE control specifies the title to be printed in the first line of page headings. 'Title' can be any sequence of printable ASCII characters except the apostrophe (although an apostrophe can be printed by putting two consecutive apostrophes into the 'title' string).

The title is truncated on the right, if necessary, to fit the PAGEWIDTH specified.

Example: TITLE('SUBROUTINE TO PRINT TOTALS')

EJECT Control

Type: General
Form: EJECT
Default: None

The EJECT control causes the current control line and subsequent source lines to start printing at the next page. If the NOPRINT, NOLIST, or NOPAGE control is in effect, the EJECT control is ignored.

Source File Controls

These controls affect the interpretation of FORTRAN source code. The controls are:

REENTRANT
DO77/DO66
STORAGE
FREEFORM/NOFREEFORM
INCLUDE

REENTRANT Control

Type: Primary
Form: REENTRANT
Default: None; that is, subprogram is not reentrant unless this control is specified.

The REENTRANT control specifies that all SUBROUTINE or FUNCTION subprograms following it are to be reentrant. BLOCK DATA subprograms are not affected by this control. Main programs are not affected by this control; its use in a main program causes a warning message.

Local variables contained in reentrant subprograms are allocated dynamically on the stack (at run time); no statically-allocated storage (allocated at load time) is used. Local variables and arrays must not be initialized by DATA statements in reentrant subprograms. References to COMMON variables are allowed, but must be used with care.

If you want to specify reentrancy for selected subprograms only, compile those subprograms separately from the rest of the program.

DO77/DO66 Controls

Type: Primary
Form: DO77
DO66
Default: DO77

If the DO77 control is specified, DO loops in the FORTRAN source program conform to the explicit standards of the ANSI FORTRAN 77 subset. DO66 specifies that the 1966 ANS FORTRAN standard is in effect.

In particular, the 1966 standard implies that all DO loops must be executed at least once when encountered during program execution. The 1977 standard allows zero iterations to be specified by the values of initial and terminal expressions.

STORAGE Control

Type: Primary
Form: STORAGE(INTEGER*length,LOGICAL*length)
Default: STORAGE(INTEGER*2,LOGICAL*1)

The STORAGE control specifies the default lengths (in bytes) to be used for integer or logical variables, array elements, or constants. The default can be overridden by FORTRAN INTEGER or LOGICAL type statements or, in the case of integer constants, by an explicit number base specification.

'Length' can be 1, 2, or 4. If the STORAGE compiler control is not specified, the defaults are '2' for INTEGER and '1' for LOGICAL. INTEGER and LOGICAL lengths can also be specified separately in the form

```
STORAGE(INTEGER*length)
```

The default lengths for this control do not conform to the ANSI standard 'numeric storage unit' allocation. To be totally ANSI compatible, specify

```
STORAGE(INTEGER*4,LOGICAL*4)
```

FREEFORM/NOFREEFORM Controls

Type: General
Form: FREEFORM
NOFREEFORM
Default: NOFREEFORM

If the NOFREEFORM control is in effect, source code lines must be in the standard FORTRAN format. That is, comment line indicators are in column 1, statement labels in columns 1-5, continuation line indicators in column 6, and statements in columns 7-72.

The FREEFORM control allows you to begin statements in column 2 instead of column 7, simplifying console input of FORTRAN source programs. If FREEFORM is specified, statement labels must begin in column 1 and continuation lines must have an ampersand ('&') in column 1. Comments are indicated as in standard FORTRAN, that is, by a 'C' or '*' in column 1. If a statement does not contain a 'C' or '*' as its first character, it may actually begin in column 1.

INCLUDE Control

Type: General
Form: INCLUDE(file)
Default: None

The INCLUDE control causes subsequent source code to be input from the specified 'file' until an end-of-file is reached. At end-of-file, input resumes from the file being processed when the INCLUDE was encountered. The included file may not contain an END statement.

The included file may itself contain INCLUDE controls, up to a total of six files (that is, INCLUDE controls can be nested to a depth of five).

An INCLUDE control must be the rightmost control on a command line or control line.

The 'file' specified can be the name of any diskette-resident file.

Example: INCLUDE(:F1:TRIG.TWO)

This example causes the file 'TRIG.TWO,' located on diskette drive ':F1:,' to be included in the FORTRAN source file.

Compiler Resource Controls

These controls specify work files to be used by the compiler and also handle the saving/restoring of certain compiler controls. The controls are:

WORKFILES
SAVE
RESTORE

WORKFILES Control

Type:	Primary
Form:	WORKFILES (device, device)
Default:	WORKFILES(:F1:,:F1:)

The WORKFILES control specifies two diskette devices to be used as the compiler's temporary work files. For example, possible parameters are :F0:, :F1:, :F2:, and :F3:.

The parameters need not specify different devices. If only one parameter is specified, the effect is the same as specifying the same drive for both parameters.

SAVE Control

Type:	General
Form:	SAVE
Default:	None

The SAVE control stacks the current settings of the FREEFORM, LIST, and CODE controls (though the current settings remain valid until explicitly changed).

Controls can be stacked to eight levels.

RESTORE Control

Type:	General
Form:	RESTORE
Default:	None

The RESTORE control reestablishes the control settings saved on the top of the SAVE stack. The restored settings are then removed from the stack.

Default Control Settings

The FORTRAN compiler assumes the following defaults if the corresponding controls are not selected:

```
OBJECT(source-file.OBJ)
NODEBUG
OPTIMIZE(1)
PRINT(source-file.LST)
LIST
NOSYMBOLS
NOCODE
NOXREF
PAGING
PAGELENGTH(60)
PAGEWIDTH(120)
DO77
STORAGE(INTEGER*2,LOGICAL*1)
NOFREEFORM
WORKFILES(:F1:,:F1:)
```

CHAPTER 3
LISTING FORMATS

The compiler list file contains a variety of information. This chapter describes the information gathered in this file and the format in which it is listed.

Program Listing

Page Heading

During compilation a program listing may be produced. Unless the NOPAGING compiler control is active, each page of the listing has a numbered page heading identifying the compiler and optionally including a user-supplied title and/or date. If NOPAGING has been specified, only the first page of the listing contains this heading. The format of the page heading is

FORTRAN COMPILER [title] [date] PAGE nnn

where

title	is the string specified by the most recent TITLE compiler control
date	is the most recent date specified by the DATE compiler control
nnn	is the page number (beginning at 1 for each program unit).

The title is truncated on the right, if necessary, to fit the current PAGEWIDTH control setting, or is extended on the right with blanks to right-justify the date and page fields. The page heading line is followed by two blank lines.

Compilation Introductory Lines

The first part of the program listing acts as an introduction to the compilation beginning with the compiler identification and the name of the FORTRAN source module being compiled. For example:

```
ISIS-II FORTRAN-80 V2.0 COMPILATION OF PROGRAM UNIT MYPROG
```

The next line names the file receiving the object code. For example:

```
OBJECT MODULE PLACED IN :F1:MYPROG.OBJ
```

Finally, the command line used to invoke the compiler is reproduced. For example:

```
COMPILER INVOKED BY: FORT80 :F1:MYPROG.SRC CODE
```

The listing of the program itself follows this compilation summary information.

Assembly-Language Listing

If the CODE compiler control was specified, the next item in the program listing is the assembly-language equivalent of the object code generated. The assembly-language listing for each program unit begins a new page.

This part of the program listing has the form of a pseudo assembly-language listing resembling the output of the 8080/8085 assembler. The code listing for each program unit precedes the source text for that program unit. It appears in six columns of information conforming to the standard assembly language format:

1. Relocatable location counter (hexadecimal notation)
2. Resultant binary code (hexadecimal notation)
3. Label field
4. Symbolic operation code
5. Symbolic arguments
6. Comment field

Not all six of these columns will appear on any one line of the code listing.

The assembly-language code generated from each FORTRAN source statement is preceded by a comment line indicating the internal statement number of the source statement. Compiler-generated labels (e.g., those which mark the beginning and ending of a DO loop) are preceded by '@'; source program statement labels are preceded by '?' to distinguish them from numeric constants. The comments appearing on PUSH and POP instructions indicate the stack depth associated with the stack instruction.

Figure 3-1 shows a portion of the CODE listing for a sample FORTRAN program, followed by the source code from which it was generated.

FORTRAN Source Listing

The source listing contains a copy of the source input plus additional information.

Columns 1-4 are a sequential numbering of FORTRAN statements. Error messages, when present, refer to these internal numbers, not to statement labels coded as part of the FORTRAN program.

Columns 5-7 indicate whether the source line was included in the program with the INCLUDE control. If so, column 5 contains an equal sign (=). If the text was included as the result of a nested INCLUDE, the column contains a digit indicating the level of nesting.

The remainder of the line contains a copy of the source text, as coded, except that ASCII TAB characters are expanded with multiple blanks, as necessary, to the next character position that is a multiple of eight.

The sequence number in columns 1-4 is not applied to comment, control, and continuation lines. If a FORTRAN source line must be continued on another line in the listing because of a PAGEWIDTH limitation, the continued line has a hyphen (-) in column 7.

```
FORTRAN COMPILER                                                PAGE   1

ISIS-II FORTRAN-80 V2.0 COMPILATION OF PROGRAM UNIT @MAIN
OBJECT MODULE PLACED IN :F1:STOCKS.OBJ
COMPILER INVOKED BY:  FORT80 :F1:STOCKS.SRC CODE XREF PAGEWIDTH(67)


                                                    ; STATEMENT # 2
          0067  310000          LXI     SP,@STACK$ORIGIN
          006A  CD0000          CALL    FQ0GO
                                                    ; STATEMENT # 3
          006D  210600          LXI     H,6H
          0070  221600          SHLD    @IOPB
          0073  210400          LXI     H,?10
          0076  221A00          SHLD    @IOPB+4H
          0079  210000          LXI     H,0H
          007C  221C00          SHLD    @IOPB+6H
          007F  212800          LXI     H,ISTAT
          0082  222000          SHLD    @IOPB+0AH
          0085  218000          LXI     H,80H
          0088  221E00          SHLD    @IOPB+8H

                .
                .
                .

          012C  110100          LXI     D,1H
          012F  010400          LXI     B,4H
          0132  CD0000          CALL    FQ0164
          0135  110100          LXI     D,1H
          0138  010E00          LXI     B,CLOSE
          013B  CD0000          CALL    FQ0162
          013E  110100          LXI     D,1H
          0141  011200          LXI     B,CHNG
          0144  CD0000          CALL    FQ0162
          0147  CD0000          CALL    FQ0167
                                                    ; STATEMENT # 10
          014A  C39400          JMP     ?15
```

```
FORTRAN COMPILER                                                PAGE   3

        C PRINT FORMATTED LIST OF SELECTED STOCKS SHOWING
        C NAME, EXCHANGE, CLOSING PRICE, AND CHANGE FROM
        C PREVIOUS CLOSE
        C
  1           CHARACTER STOCK*10,EXCH*4
  2           REAL CLOSE,CHNG
  3           WRITE(6,10,IOSTAT=ISTAT)
  4        10 FORMAT('1','STOCK',7X,
             &'EXCHANGE  CLOSING PRICE  CHANGE'//)
  5        15 READ 20,STOCK,EXCH,CLOSE,CHNG
  6        20 FORMAT(A10,1X,A4,F6.2,F5.2)
  7           IF(STOCK.EQ.'DONE') STOP
  8           WRITE(6,30) STOCK,EXCH,CLOSE,CHNG
  9        30 FORMAT(1X,A10,9X,A4,F14.2,F12.2)
 10           GO TO 15
 11           END
```

Figure 3-1. Sample Program Listing

If an error is detected in the source code during compilation, a message is inserted into the source listing in the following format:

***ERROR *m*, STATEMENT *n*, [NEAR *symbol,*] *message*

where

m	is the error number
n	is the sequential number of the statement containing the error
symbol	is a pointer to the position of the error within the statement
message	is the error message.

Example:

***ERROR #71, STATEMENT #33, NEAR 'OP1', OPERAND EXPECTED

See Figure 3-1 for an example of a source listing. Error handling and error messages are discussed in greater detail in Appendix B.

Symbol And Cross-Reference Listings

A summary of symbol usage follows the program listing if either the SYMBOLS or XREF compiler control was specified. An entry is printed for each variable, array, function, subroutine, and statement label mentioned in each program unit of the source text. Each entry includes:

- The program identifier for the symbol
- The symbol's attributes and the relative hexadecimal address of the symbol (if meaningful)
- If XREF is active, a list of the statements in which the symbol is referenced or defined

The attributes for each symbol are:

- Its type (Integer, Real, Logical, Character, or none)
- Its length (the length in bytes or characters if appropriate; the length of an element for arrays)
- Its kind (variable, array, label, common block, intrinsic, statement function, program unit)
- Its scope (external, local, dummy parameter, common block name)

Figure 3-2 shows the symbol attribute and cross-reference listing for the sample program of Figure 3-1.

Compilation Summary And Signoff

For each program unit compiled, a compilation summary follows the program and symbol listings. The information provided in the summary is

- Code area size. The size in bytes of the code segment of the output module.
- Variable area size. The size in bytes of the data segment of the output module.
- Maximum stack size. The size in bytes of the stack segment allocated for the output module.

```
FORTRAN COMPILER

CROSS-REFERENCE LISTING
-----------------------

  DEFN  ADDR  SIZE  NAME, ATTRIBUTES, AND REFERENCES
 ----- ----- -----  --------------------------------

     4 0004H        10         LABEL
                                 3    4

     5 0094H        15         LABEL
                                 5   10

     6 0038H        20         LABEL
                                 5    6

     9 004DH        30         LABEL
                                 8    9

       0016H    18  @IOPB      INTEGER*2 DIMENSIONED
                                 3

       0012H     4  CHNG       REAL*4
                                 2    5    8

       000EH     4  CLOSE      REAL*4
                                 2    5    8

       000AH     4  EXCH       CHARACTER*4
                                 1    5    8

       0028H     2  ISTAT      INTEGER*2
                                 3

       0000H    10  STOCK      CHARACTER*10
                                 1    5    7    8
```

Figure 3-2. Sample Symbol-Attribute and Cross-Reference Listing

- Lines read. The number of source lines processed by the compiler.
- Program errors. The number of errors detected in this module.

All size information is shown in both hexadecimal and decimal. This summary is printed for each program unit unless the NOPRINT compiler control is in effect. Figure 3-3 shows the compilation summary for the sample program of Figure 3-1. Refer to the following chapter for a discussion of the various segment types.

```
MODULE INFORMATION:

     CODE AREA SIZE     = 014DH     333D
     VARIABLE AREA SIZE = 002AH      42D
     MAXIMUM STACK SIZE = 000AH      10D
     16 LINES READ

     0 PROGRAM ERROR(S) IN PROGRAM UNIT @MAIN

     0 TOTAL PROGRAM ERROR(S)
     END OF FORTRAN COMPILATION
```

Figure 3-3. Sample Compilation Summary

The summary listing of the last program unit is followed by the message

n TOTAL PROGRAM ERROR(S)
END OF FORTRAN COMPILATION

where 'n' is the total number of errors in *all* program units in the source file.

The program error summaries and the "END OF FORTRAN COMPILATION" message are also directed to the system console, regardless of the state of the PRINT/NOPRINT control.

CHAPTER 4
PROGRAM LINKAGE, RELOCATION, AND EXECUTION

Building An Executable Program

Once your FORTRAN program has been compiled, two tasks remain before it can be executed. First, your object module must be linked to the FORTRAN run-time libraries and any other required modules. The modules that make up your final program need not be translated from the same language. FORTRAN, PL/M, or assembly language can be used depending on your program needs. Relocatable modules produced by the FORTRAN-80 compiler, PL/M compiler, or 8080/8085 assembler can be input to LINK to build a program.

After all modules and libraries have been linked to form a new relocatable module, the relative addresses created in this module must be given absolute memory addresses by the LOCATE program.

The following software is needed to build your FORTRAN program:

- ISIS-II, version 3.0 or later, which includes:
  — LINK, version 2.1 or later
  — LOCATE, version 2.1 or later
- Selected FORTRAN-80 run-time libraries, from the standard set provided with the compiler, which include:
  — F80RUN.LIB
  — F80ISS.LIB
  — F80NIO.LIB
  — FPEF.LIB
  — FPSOFT.LIB
  — FPHARD.LIB
  — FPNULL.LIB
  — PLM80.LIB
- For RMX/80 users, selected run-time libraries from the FORTRAN-80 Run-Time Package for RMX/80 Users, which includes:
  — F80RMX.LIB
  — FPSFTX.LIB
  — FPHRDX.LIB
  — FPHX10.LIB
  — F80NTH.LIB
  — F80NDS.LIB

Memory Allocation

Memory for each compiled FORTRAN program unit is allocated in several independent, relocatable segments. These are called CODE, DATA, STACK, BLANK COMMON, NAMED COMMON, MEMORY, and ABSOLUTE segments.

Executable object code and data constants are placed in the CODE segment. This includes real, integer, Hollerith, character, and logical scalar constants and formats from FORMAT statements.

All local variables (except those in subprograms compiled while the REENTRANT control is in effect) are allocated memory in the DATA segment. Compiler-generated temporary storage for intermediate values and for copies of argument addresses are placed in the DATA segment also.

Memory for local arrays, variables, and compiler-generated intermediate values of reentrant subprograms is dynamically allocated at run time in the STACK segment. Subprogram calls passing more than two argument (or result) addresses allocate memory for them in the STACK segment as well.

All variables and arrays in blank common are allocated to the BLANK COMMON segment. In the case of named common, all variables and arrays are allocated to a NAMED COMMON segment corresponding to that name.

The MEMORY segment is assigned to RAM memory that is not allocated to CODE, DATA, STACK, or COMMON segments.

In addition to CODE, DATA, STACK, COMMON, and MEMORY segments, a relocatable object module can contain code or data with absolute addresses already assigned. These may be modules originally created in assembly language using the ASEG directive, modules created in PL/M using the AT attribute, or modules produced by an earlier LOCATE operation.

Linking Object Modules

The ISIS-II LINK program lets you combine object modules from several input files into one object module in one output file. While combining modules, LINK adjusts all addresses so they are relative to the beginning of the segments in the new output module. LINK also searches libraries for modules that resolve external references in the modules being combined and includes them in the new output module. If any unresolved external references remain in the output module, LINK puts a message in its link map.

The output module must be processed by LOCATE before it can be executed. LOCATE assigns absolute memory locations to the object module. The output module can also be used as input to LINK to be combined with other modules into a new and expanded output module.

FORTRAN modules can be linked using either the LINK command, listing individually the libraries and other modules to be linked, or the SUBMIT command (to access the FORTRAN submit file, FLINK.CSD, which is described in the following section). LINK and SUBMIT are described in detail in the *ISIS-II User's Guide* and are simply summarized here. Linking FORTRAN libraries, particularly F80ISS.LIB, does require some special considerations, however, and these are highlighted below.

LINK and SUBMIT Commands

The LINK program is invoked by entering the LINK command at the ISIS command level. The syntax of the LINK command is:

```
LINK input-list TO link-file [link-controls]
```

Your 'input-list' must include all required FORTRAN libraries in the sequence:

```
RMX8xx.LIB(START), object-files,F80RUN.LIB,&
{F80ISS.LIB | F80RMX.LIB | F80NIO.LIB},FPEF.LIB,{FPSOFT.LIB | FPHARD.LIB | FPSFTX.LIB | FPHRDX.LIB | FPHX10.LIB | FPNULL.LIB},RMX-files,PLM80.LIB
```

where braces { } indicate a choice of items and shading indicates items required only under RMX/80. (Four of the libraries in braces—F80RMX.LIB, FPSFTX.LIB, FPHRDX.LIB, and FPHX10.LIB—are to be selected only by RMX/80 users; however, these library names are not shaded here, because one selection from each group in braces is required for any environment.)

- 'xx' in 'RMX8xx.LIB' stands for 20, 30, or 10, for systems based on the iSBC 80/20, 80/30, and 80/10 respectively. (This is explained further in the *RMX/80 User's Guide* and is of no concern to the non-RMX user.)
- 'object-files' are one or more files containing the modules produced by compiling your FORTRAN program, plus other modules (if any) translated from PL/M or assembly language code. For RMX/80 systems, 'object-files' must include your configuration module.
- Use F80ISS.LIB if your program is to run under ISIS-II and perform I/O other than port I/O, F80RMX.LIB if your modules are to run under RMX/80 and perform I/O other than port I/O, or F80NIO.LIB if only port I/O (or no I/O) is used in FORTRAN.
- Link in FPSOFT.LIB to use software floating-point for non-RMX systems, FPHARD.LIB to use the iSBC 310 interface for non-RMX systems, FPSFTX.LIB to use software floating-point under RMX/80, FPHRDX.LIB to use the iSBC 310 interface under RMX/80 on an iSBC 80/20 or 80/30, FPHX-10.LIB to use the iSBC 310 interface under RMX/80 on an iSBC 80/10, or FPNULL.LIB if no floating-point operations are used.
- 'RMX-files' are other files required for RMX/80 systems. These are explained in Chapter 6.

The library modules included must be listed in the exact order shown. Any BLOCK DATA subprograms *residing in a library* must be linked explicitly also. They are not linked automatically with the programs that use them.

The 'link-controls' allowed in the LINK command are MAP, NAME, and PRINT, as described in the *ISIS-II User's Guide.*

The following is an example of a LINK command you might give if your program is to run under ISIS-II and use the software floating-point libraries.

```
-LINK :F1:FPROG.OBJ, F80RUN.LIB, F80ISS.LIB, FPEF.LIB, &
**FPSOFT.LIB, PLM80.LIB TO FPROG.LNK MAP
```

(Note: The double asterisks are prompts issued by the LINK command.)

If you plan to link only your object file and the five FORTRAN libraries normally required for execution under ISIS-II with software floating-point (i.e., F80RUN.LIB, F80ISS.LIB, FPEF.LIB, FPSOFT.LIB, and PLM80.LIB), and do not plan to use any of the LINK command controls, you can simplify the link operation by entering the following SUBMIT command at the ISIS command level.

```
-SUBMIT FLINK(object-file, link-file[, lib-drive])
```

where 'object-file' and 'link-file' are ISIS file-names and 'lib-drive' is of the form :Fx:, x being the number of the drive on which LINK and the FORTRAN run-time libraries reside. ('Lib-drive' may be omitted; the default is :F0:.)

This option was selected in the sample program development in Chapter 1, where linkage was performed by the command:

```
-SUBMIT FLINK(:F1:MYPROG.OBJ, :F1:MYPROG. LNK)
```

CAUTION

The LINK program uses a temporary file with the name LINK.TMP. The diskette drive used is the same specified by the output file. If you have a file with this name on the same drive as the output file, your file will be destroyed.

FORTRAN Library F80ISS.LIB

In certain situations you may want to avoid linking library F80ISS.LIB, which consists of input/output routines for the ISIS-II environment. You may be running on an Intellec system but prefer to call PL/M or assembly-language routines or the monitor to invoke ISIS directly and avoid the overhead of FORTRAN I/O. F80ISS.LIB can typically add 8,000-15,000 bytes to your program, depending on the operations used. Depending on your needs, you may be able to avoid linking this library entirely or, by judicious use of FORTRAN I/O statements within your program, to reduce the number of F80ISS.LIB modules actually linked.

If you use no standard FORTRAN I/O statements (READ, WRITE, PRINT, OPEN, CLOSE, BACKSPACE, REWIND, ENDFILE), and no STOP or PAUSE statements, you may omit F80ISS.LIB completely and substitute F80NIO.LIB, as described previously in the LINK command discussion.

Alternatively, you can reduce the space taken up by F80ISS.LIB by limiting the types of I/O operations your FORTRAN program performs, since only those modules actually required by your program are linked in. For instance, the total I/O system with all capabilities requires 18,000 bytes of storage; if you use only sequential, formatted I/O, you need 15,000 bytes.

FORTRAN Library F80RMX.LIB

For RMX/80 systems, you can often reduce the number of modules linked in from F80RMX.LIB (or avoid using F80RMX.LIB at all) by coding your I/O judiciously, just as you would for F80ISS.LIB.

If you use no standard FORTRAN I/O statements (READ, WRITE, PRINT, OPEN, CLOSE, BACKSPACE, REWIND, ENDFILE) and no STOP or PAUSE statements, you may omit F80ISS.LIB completely and substitute F80NIO.LIB, as described previously in this discussion.

Alternatively, you can reduce the space taken up by F80RMX.LIB by limiting the types of I/O operations your FORTRAN program performs, since only those modules actually required by your program are linked in. For instance, under RMX/80 the total I/O system with all capabilities requires 21,000 bytes of storage; if you use only sequential, formatted I/O, you need 16,000 bytes.

Linking with non-FORTRAN Procedures

The relocatable object modules produced by the ISIS-II FORTRAN-80 compiler are compatible with those produced by the ISIS-II PL/M compiler and the 8080/8085 macro assembler. Modules written in the three languages can be linked together. This compatibility allows you to use FORTRAN to code those segments of your application to which the features of FORTRAN are particularly well suited— multidimensional arrays, formatted I/O, floating-point arithmetic, and/or FORTRAN intrinsic functions— and write other segments in PL/M or assembly language if you desire.

The FORTRAN-80 compiler implements procedure (function and subroutine) calls in the same manner as PL/M-80. This allows FORTRAN program units to call PL/M procedures and vice-versa. FORTRAN-80 passes its arguments by reference (that is, by address). Furthermore, the FORTRAN-80 convention for calling a *function* with *n* arguments is to pass *n+1* addresses to the function routine (where the first address is a location for storing the result). Function and subroutine arguments are passed in the same locations as in PL/M, that is:

- For a single-argument function or subroutine, the argument address is passed in registers B (high-order byte) and C (low-order byte) of the 8080/8085;
- For a two-argument function or subroutine, the first argument address is passed as above and the second in registers D (high) and E (low);
- For a function or subroutine of more than two arguments, the last two argument address are passed as described above (next to last in B and C, last in D and E), and the remainder are passed on the stack (pushing them onto the stack in order from left to right in the argument list).

The relocatable modules produced by the ISIS-II FORTRAN-80 compiler are compatible with those produced by the ISIS-II PL/M compiler and the 8080/8085 macro assembler. Modules written in the three languages can be linked together. This compatibility allows you to use FORTRAN to code those segments of your application in which you need the features of FORTRAN—multidimensional arrays, formatted I/O, floating-point arithmetic, and/or FORTRAN intrinsic functions—and write other segments in PL/M or assembly language if you desire.

A CHARACTER argument in a FORTRAN procedure call is treated as two arguments in the generated object code. The first argument is the address of the character data; the second is a 2-byte integer value (passed by value, not by reference) indicating the length of the character string.

When using FORTRAN and non-FORTRAN program segments together, note that FORTRAN and non-FORTRAN I/O on the same file may interact improperly. You should use one of the other only on any given file.

If your main program is written in PL/M or assembly language rather than in FORTRAN and it calls FORTRAN subprograms, your FORTRAN subprograms must include calls to special procedures from F80RUN.LIB that perform initialization and termination actions for the FORTRAN arithmetic and input/output routines. (These calls are generated automatically by the compiler for a FORTRAN main program.) Before your program performs any floating-point operations or FORTRAN I/O, it must call the external procedure FQ0GO, which takes no parameters. Before exiting, it must call the external procedure FQ0END, which likewise takes no parameters.

For further information on linking FORTRAN and non-FORTRAN procedures together, refer to Intel Application Note AP-44, *How to Use FORTRAN with Other Intel Languages.*

Example:

```
      SUBROUTINE CRUNCH (ARG1,ARG2)
      REAL ARG1,ARG2
      REAL XTEMP,YTEMP
      CALL FQOGO
        .
        .
        .
C     ARITHMETIC AND FORMATTED I/O OPERATIONS GO HERE
        .
        .
        .
      RETURN
```

Object Module Relocation

The ISIS LOCATE program takes as input a relocatable object module and produces an output file containing the same object module with addresses fixed to absolute locations. The format of the LOCATE command is:

```
LOCATE input-file [TO output-file] [locate-controls]
```

Operation of the LOCATE program is described in Chapter 4 of the *ISIS-II User's Guide*. This section describes specific considerations when locating FORTRAN object modules.

ORDER Control

The ORDER control, which can be specified as part of the LOCATE command syntax, allows you to dictate the sequence of segment types in memory. The format of the ORDER control is:

```
ORDER(segids)
```

where 'segids' is some combination of the segment names CODE, DATA, /common name/,// (for blank common), MEMORY, and STACK. If ORDER is not specified, module segments are located sequentially in memory in the following order: CODE, STACK, COMMONs, DATA, and MEMORY, where the term 'COMMONs' means all COMMON segments in an arbitrary order.

The ORDER list may be partial; all module segments need not be listed. In this case, all segments specified in the ORDER control are taken in the order specified. The remaining segments are taken in the default order, *after* the modules specified in the ORDER control.

Base Address Controls

Segments can be explicitly located in memory by the base address controls. These controls assign an address for the first byte of the segment. The controls are:

```
CODE(addr)
DATA(addr)
STACK(addr)
MEMORY(addr)
/common name/(addr)
//(addr)
```

The last two controls refer to NAMED COMMON segments and the BLANK COMMON segment, respectively.

If you plan to locate some segments at specific addresses and let LOCATE place the others, you should use the ORDER control to modify the default sequence so the segments will be located in coordination with the specified base addresses. Also, be sure to specify the MAP control, which is your only notification of resulting conflicts.

When you locate FORTRAN common segments to specific addresses, you should also locate the MEMORY segment to an address above the top of the highest common segment. LOCATE handles common segments in an arbitrary order. Unless the MEMORY segment is located specifically using the ORDER control, it will follow the last common segment handled (which could be at a low memory address) and will conflict with all segments above the common segment.

Program Execution and I/O Unit Preconnection

Your linked and relocated program can now be loaded and executed by entering its name at the ISIS command level. For example:

```
—MYPROG
```

As discussed in the *FORTRAN-80 Programming Manual*, FORTRAN I/O statements operate on *units* that are *connected* to files on a one-to-one basis. A unit-file connection can be made when the file is opened (by the OPEN statement) or by *preconnecting* the unit and file via the UNIT run-time control.

As part of FORTRAN run-time conventions in the ISIS environment, two units are preconnected and need not be connected by the FORTRAN OPEN statement. These are:

Unit	Device
5	Console input
6	Console output

If you are running your programs under ISIS-II and wish to preconnect other units or override these default preconnections, you can specify the UNIT run-time control at the time you call your program for execution. The format of the UNIT control is

```
UNIT n = device
```

where 'n' is a number in the range 0-255 and 'device' is any device name recognized by ISIS-II.

Examples:

```
TRIG UNIT 4 = :LP:
TRIG.LOC UNIT1 = :CI:, UNIT0 = :CO:
```

Note that the preconnection feature applies only to FORTRAN programs prepared to run under ISIS-II; preconnection is not available for the RMX/80 run-time environment.

CHAPTER 5
iSBC 310 INTERFACE

To improve the speed of floating-point (REAL) arithmetic in your FORTRAN program, you may configure an iSBC 310 High-Speed Mathematics Unit in your system and use it, by means of the interface libraries provided with FORTRAN-80, to aid in performing floating-point arithmetic. This chapter provides instructions for using the FORTRAN software interface to the iSBC 310 unit.

All of the floating-point operations of the iSBC 310 are used via the interface provided in the appropriate floating-point run-time library. When the interface is used, the following operations will be performed with the aid of the iSBC 310 hardware: floating-point addition, subtraction, multiplication, division, square root, comparison, and conversions between floating-point and integer formats. The results of arithmetic operations are the same whether the all-software floating-point routines or the iSBC 310 interface is used. The iSBC 310 interface is designed to maximize concurrency between the CPU and the iSBC 310, to maximize overall execution speed, and to maintain a consistent error handling strategy.

Use of the interface requires only that you link in the appropriate run-time library and observe several hardware and software constraints. If these constraints are heeded, your FORTRAN programs will run with either the software floating-point routines or the iSBC 310 interface.

CAUTION

When installing the iSBC 310 in an Intel Intellec Microcomputer Development System (Model 800 or 888), the Intellec CPU board must be reconfigured to generate a Qualified Write signal. (Refer to the *Intellec Microcomputer Development System Hardware Reference Manual.*) This modification is not necessary for Intellec Series II systems.

Dedicated Use of iSBC 310

When using the FORTRAN-80 interface to the iSBC 310 unit, you should consider the iSBC 310 to be dedicated to use by FORTRAN only. Making other (direct) use of the 310 board in your system may cause the results of FORTRAN floating-point operations to be unreliable.

In non-RMX systems, this situation applies because the iSBC 310 unit is a non- reentrant resource. In RMX/80 systems, provision is made for saving the board's registers and memory work area so that one task using the board may interrupt another task that uses it. However, use of the iSBC 310 board under RMX/80 must be limited to FORTRAN tasks *within the RMX/80 system.*

I/O Base Address and Memory Base Address

The software interface routines assume that the iSBC 310 I/O base address is set *manually* (via switch on the 310 board) to 98H—which, of course, cannot be duplicated for any other device. The iSBC 310 memory base address will be assumed to be contained in the public address variable FQFMBA. This memory base address is set up at initialization by the routine FQ0GO (which is automatically called in a FORTRAN main program and must be called explicitly by your own program if you have no FORTRAN main program), so you, the user, need not be concerned with this address.

Error Handling

When using the FORTRAN interface, you should wire-wrap the iSBC 310 unit so that it initiates no interrupt request on completion of an operation with or without an error. FORTRAN-80 provides a means to transfer control to its own default error handler, to an ISIS-oriented error reporting routine, or to a user-supplied error routine. Refer to 'Run-Time Arithmetic Errors' in Appendix B for details.

An error handler (either one of the two error handlers provided in the run-time libraries, or a user-supplied error handler) is called whenever a floating-point operation signals an error and whenever an invalid floating-point result is not handled by the routine in which it has occurred. When the iSBC 310 interface is used, if the error handler is called it will be called just before the next floating-point operation is started on the iSBC 310 board, unless the function that produced the error was a test, comparison, or fix. In the case of a test, comparison, or fix error, the error handler is called right after the floating-point operation is performed on the board and before the test, comparison, or fix routine returns.

Activation and Deactivation of iSBC 310 Memory Mapping

The iSBC 310 unit communicates directly with an area of RAM memory. The starting address of this memory area is specified by the iSBC 310 interface software; however, your program must activate the memory mapping on the 310 board as part of system initialization, and deactivate it before exiting. For systems using the iSBC 310 interface, the initialization procedure FQ0GO (in F80RUN.LIB) includes a call to a routine that activates the iSBC 310 memory mapping, and the termination procedure FQ0END (also in F80RUN.LIB) includes a call to deactivate the memory mapping. For a FORTRAN main program, the compiler automatically generates calls to FQ0GO and FQ0END. However, if your main program is not a FORTRAN program, you should call FQ0GO before performing any floating-point operations, and call FQ0END before exiting. (These two procedures take no parameters.)

For instance, a PL/M main procedure might be written in this form:

```
PLMAIN:
DO;
  /* DECLARATIONS OF VARIABLES AND ARRAYS */
  /* DECLARATIONS OF EXTERNAL PROCEDURES */
  /*INITIALIZATION OPERATIONS*/
    .
    .
  CALL FQ0GO;
  /* arithmetic, I/O, and other operations*/
    .
    .
  CALL FQ0END;
END PLMAIN;
```

If the iSBC 310 memory mapping is *not* deactivated and is subsequently reactivated, some data in memory may be destroyed. For this reason, if you supply your own error handling routines (see 'Run-Time Arithmetic Errors' in Appendix B), it is important that these routines also include a call to FQ0END in case of a fatal error. If you exit to ISIS-II during program debugging, you can deactivate the iSBC memory mapping by simply re-BOOTing the system.

NOTE

In an iSBC 80/20-based system installed in a System 80/20 chassis, re-BOOTing the system as described above will deactivate the memory mapping only if the iSBC 80/20 unit is installed in the *bottom* slot of the cardcage. (This restriction does not apply to CPU boards other than the iSBC 80/20, or to iSBC 80/20 boards installed in chassis other than the System 80/20.)

Use of iSBC 310 Interface in iSBC 80-Based Systems (With or Without RMX/80)

In an iSBC 80/20, 80/30, or 80/10-based system, the iSBC 310 unit cannot be memory-mapped onto on-board RAM. You should take this into account when designing and coding your system. You can use LINK and LOCATE commands to ensure that the 310 is mapped onto existing off-board RAM; refer to the following section ('LINK Command') for details.

Under RMX/80, a user-supplied interrupt service (RQSETV) routine must not have access to the iSBC 310 unit—i.e., if you are using the iSBC 310 interface option, your RQSETV routines cannot perform any floating-point operations.

Also note that in an RMX/80 system, the iSBC 310 interface package disables interrupts for brief periods during floating-point operations. You must take this situation into account when programming interrupt-driven tasks.

LINK Command

When you use the iSBC 310 interface, you link in the same run-time libraries as if you were using the software floating-point routines, except that the floating-point arithmetic library will be different. Library FPHARD.LIB is provided to support the use of the iSBC 310 unit to perform floating-point arithmetic in the non-RMX (ISIS-II or stand-alone hardware) run-time environment. This library substitutes for FPSOFT.LIB. Two additional libraries are provided in the FORTRAN-80 Run-Time Package for RMX/80 Systems (iSBC 801) to perform the same functions under RMX/80: FPHRDX.LIB for iSBC 80/20 and 80/30 systems, and FPHX-10.LIB for iSBC 80/10 systems. You select one of these libraries in place of FPSFTX.LIB. For the order of all libraries in the LINK command, refer to Chapter 4 or Appendix D.

The following is an example of a LINK command that could be given to link FORTRAN programs for a system that runs under ISIS-II and includes an iSBC 310 unit dedicated to FORTRAN use.

```
-LINK :F1:FPROG.OBJ,F80RUN.LIB,F80ISS.LIB,FPEF.LIB,&
**FPHARD.LIB,PLM80.LIB TO FPROG.LNK MAP
```

(Note: The double asterisks are prompts issued by the LINK command.)

This example is identical to the one given in Chapter 4 except that the iSBC 310 interface is specified. For an example of a LINK command for an RMX/80 system in which FORTRAN tasks use the iSBC 310 interface, refer to 'Link Command' in Chapter 6.

In most cases, the LINK command above will result in correct memory mapping of the iSBC 310 unit onto existing on-board RAM. You can examine the LINK map to check that the mapping is correct—i.e., that the segment FPR.ABS has been mapped onto existing on-board RAM. If this LINK command does not result in correct memory mapping, you can assign a specific address for the 310 mapping by giving the following commands, in order:

```
LINK FPHARD.LIB (FPR) TO FPR.REL
LOCATE FPR.REL TO FPR.ABS DATA(address) STACKSIZE(0)
```

where 'address' is the address of a 16-byte segment of on-board RAM (the address must be on a 16-byte boundary). After giving these commands, you link all libraries together with a LINK command of the form

```
—LINK :F1: FPROG.OBJ, F80RUN.LIB, F80ISS.LIB, FPEF.LIB, &
**FPR.ABS, FPHARD.LIB, PLM80.LIB TO FPROG.LNK MAP
```

—i.e., the LINK command you would normally give, but with the addition of the "FPR.ABS" segment between FPEF.LIB and FPHARD.LIB.

CHAPTER 6
PREPARING PROGRAMS TO RUN UNDER RMX/80

This chapter describes the use of the special facilities provided to support FORTRAN programs in the RMX/80 run-time environment—specifically the FORTRAN-80 Run-Time Package for RMX/80 Systems (iSBC 801). Use of RMX/80 itself is covered only to the extent necessary to explain how to interface with it; for complete instructions, refer to the *RMX/80 User's Guide.*

Under RMX/80, FORTRAN-80 input and output operations normally use the full or minimal Terminal Handler and the Disk File System, rather than the ISIS-II functions used in the ISIS-II run-time environment. Alternatively, you can omit the Terminal Handler and/or Disk File System and use port input and output, or even write your own I/O device drivers for use with the FORTRAN I/O statements. (Instructions for doing the latter are provided in Chapter 7.) The RMX/80 run-time environment for FORTRAN offers the advantage of full interrupt capabilities.

Program Structure Under RMX/80

Recall that under RMX/80, programs run as a series of *tasks* under the control of the RMX/80 Nucleus, and that tasks communicate with each other by sending messages. You may use FORTRAN to code those tasks (or subroutines callable by tasks) that make use of formatted I/O, floating-point arithmetic, and other FORTRAN features. Tasks written in FORTRAN should be coded as SUBROUTINE subprograms. (Just as there must be no PL/M main procedure under RMX/80, there must be no FORTRAN main program.)

However, note that FORTRAN, since it has no address variables, cannot interface directly with RMX/80. All sending and waiting for messages must be done by tasks coded in PL/M or assembly language; likewise, the configuration module must be coded in PL/M or assembly language. If you wish to have one task perform floating-point arithmetic *and* send and/or wait for messages, you can do so by writing a short "skeleton" task in assembly language or PL/M to do the send and wait operations, and having it call one or more FORTRAN subroutines to perform the bulk of the processing.

Initialization and Termination

For FORTRAN main programs, the compiler automatically generates calls to perform the necessary initialization and termination actions for the FORTRAN library routines. In an RMX/80 system, however, there can be no FORTRAN main program, so these calls must be included in your code. Initialization must be performed once for the whole system and (when floating-point operations are used) for individual tasks.

All the required system initialization, including initialization of the floating- point and input-output routines, is performed by the external procedure FQ0GO, which resides in F80RUN.LIB. You can initialize your system by including in it a small, high- priority task that calls FQ0GO (which takes no parameters), then suspends itself by calling the RMX/80 procedure RQSUSP. This task must have a high enough priority to ensure that it runs before any floating-point arithmetic or FORTRAN I/O is performed.

CAUTION

It is presumed that FQ0GO will not be invoked twice. Unpredictable (and usually disastrous) results may occur if this assumption is violated.

In addition, each task that uses floating-point (REAL) operations or intrinsic functions must call FQFSET, which resides in the selected floating-point arithmetic library, before doing any REAL operations. This routine initializes the internal error handler address field. The calling sequence for FQFSET is:

```
CALL FQFSET(A,ERRH)      from a FORTRAN program or
CALL FQFSET(.A,.ERRH)    from a PL/M program
```

A is a two-byte integer and ERRH is the name of an error-handling routine. The least significant bit of the high-order byte of A is a flag which, when set to 1, indicates that a user-supplied routine at the address given in ERRH is now to serve as the floating-point error handler; if this flag is 0, the error handler named FQFERH will be activated. (Two error handlers by this name, a default error handler and an alternate one, are supplied in the FORTRAN run-time libraries.) The low-order byte of A will become the new value (normally 0) of the Error Field maintained internally by the floating-point arithmetic routines. Thus the standard settings for A are 0 and #100H. The routine FQFSET is identical to the FQFRST routine described under 'Run-Time Arithmetic Errors' in Appendix B, except that FQFSET also clears internal floating-point working accumulators and should be called only once per task.

To use the default or the alternate error handler, simply call FQFSET(0,ADDR), where the value of ADDR does not matter (e.g., it can be zero). Arithmetic error handlers are discussed in detail under 'Run-Time Arithmetic Errors' in Appendix B; refer to this section in Appendix B if you wish to supply your own error-handling routine.

The following table summarizes the meanings of the possible values of the 'A' parameter.

High-Order Byte of A	Low-Order Byte of A	Meaning
Low-order bit = 1 (01H, 3FH, C7H, etc.)	Zero	Use error handler at address ERRH, and set Error Field* to zero
Low-order bit = 0 (00H, 3EH, C6H, etc.)	Zero	Use FQFERH** as error handler, and set Error Field* to zero
Low-order bit = 1	Nonzero	Use error handler at address ERRH, and set Error Field* to value of low-order byte of A
Low-order bit = 0	Nonzero	Use FQFERH** as error handler, and set Error Field* to value of low-order byte of A

*For a description of the Error Field, see 'Error Monitoring' under 'Run-Time Arithmetic Errors' in Appendix B.

**Either the default or the ISIS- oriented error handler, depending upon the options specified in the LINK command. (See 'Run-Time Arithmetic Errors' in Appendix B for details.)

Calls to FQFSET are not required for non-RMX run-time environments, but these calls will not cause errors in such environments. This feature contributes to the portability of FORTRAN code between RMX and non-RMX systems.

Any routines that may terminate system operation, such as error handlers, should also call the termination routine FQ0END. (FQ0END also resides in F80RUN.LIB and takes no parameters.) This will ensure that all input and output files are closed and that the memory mapping on the iSBC 310 math unit, if any, is deactivated. In addition, FQ0END in F80RMX.LIB calls a user-defined routine named FQ0XIT, which you must supply as part of your code. This routine should perform any system exit functions that you desire, and must *not* return.

Input and Output

The RMX/80 run-time input/output support library, F80RMX.LIB, allows you to code regular FORTRAN statements (OPEN, CLOSE, READ, WRITE, PRINT, BACKSPACE, REWIND, ENDFILE) for input and output to the Terminal Handler and Disk File System. No sending of request messages to the Terminal Handler or DFS is required; this is all done by the routines in F80RMX.LIB.

The unit/file preconnection feature available in FORTRAN under ISIS-II cannot be used under RMX/80. As part of its initialization, the RMX/80 input/output library automatically connects :CI: (the terminal input file) to unit 5 and :CO: (the terminal output file) to unit 6. Any other connections must be specified in OPEN statements. (Note that the connections of :CI: to unit 5 and :CO: to unit 6 can be overridden by OPEN statements in your program.)

To read from or write to the terminal, you specify unit 5 in a READ statement or unit 6 in a WRITE statement. To perform I/O operations on a diskette file, you specify whatever unit number and file name you decide to assign to that file. The file name must be of the form :device:filename.ext, where 'device' is any two alphanumeric characters, 'filename' is from one to six alphanumeric characters, and 'ext' is from one to three alphanumeric characters. (This is identical to the form of an ISIS path-name.) Besides the unit specifier and the file name, you should also always include an error specifier in every I/O statement and provide an error action routine. Otherwise, if an error occurs, the I/O library routines will suspend the task performing the I/O.

The following "stub" example shows how you might code a routine to read in an 80-character unformatted direct-access record from a diskette file on :F1: called DATA1.

```
      $FREEFORM
      PROGRAM READIN
C     DECLARATIONS OF VARIABLES AND ARRAYS GO HERE
      OPEN (3,IOSTAT=ERRFLG,ERR=10,FILE= ':F1:DATA1',STATUS='OLD',
&     ACCESS='DIRECT',RECL=80)
      .
      .
10    CALL OPNERR
      .
      .
C     VALUE OF M MUST BE SET HERE
      READ(3,REC=M,IOSTAT=ERRFLG,ERR=20)
      .
      .
20    CALL RDERR
      .
      .
      END
```

```
C   "OPEN" ERROR ACTION ROUTINE
C   CHECKS ERRFLG AND PERFORMS ACTION DEPENDING ON ITS VALUE
    SUBROUTINE OPNERR
    .
    .
    END

C   "READ" ERROR ACTION ROUTINE
C   CHECKS ERRFLG AND PERFORMS ACTION DEPENDING ON ITS VALUE
    SUBROUTINE RDERR
    .
    .
    END
```

The RMX-based input/output library is a non-reentrant shared resource; generally, only one FORTRAN I/O operation can be in progress at a time anywhere in your system. While one task's input or output is in progress, a software lock (as described in Chapter 3 of the *RMX/80 User's Guide*) prevents any other task from performing FORTRAN I/O. The second task will wait until the first task's input or output has finished. As a consequence, tasks that handle interrupts must *not* perform FORTRAN I/O. This is not a serious restriction; since tasks that handle interrupts must be as short and speedy as possible, it is not advisable to do FORTRAN I/O in an interrupt-handling task anyway.

An exception to the non-reentrancy rule arises in the case of terminal input. A delay in terminal input—which may often occur, for instance when the operator leaves the console—will *not* halt disk input or output indefinitely. In this case, the Terminal Handler input buffers will be saved and the software lock on the I/O system will be removed, allowing other tasks to perform disk I/O.

Note that the non-reentrancy restriction applies only to regular FORTRAN I/O; port input and output (coded in FORTRAN, PL/M, or assembly language) may be performed concurrently with formatted I/O. However, FORTRAN and non-FORTRAN I/O on the *same file* may interact improperly; use one or the other only for any given file.

Using the iSBC 310 Option Under RMX/80

When using the iSBC 310 interface for FORTRAN floating-point operations under RMX/80, note that the iSBC 310 unit cannot be memory-mapped onto on-board RAM. You should take this into account when designing and coding your system. You can use LINK and LOCATE commands to ensure that the 310 is mapped onto existing OFF-board RAM; refer to 'LINK Command' in Chapter 5 for details.

A user-supplied interrupt service (RQSETV) routine under RMX/80 may not have access to the iSBC 310 unit—i.e., if you are using the iSBC 310 interface option, your RQSETV routines must not perform any floating-point operations.

Configuration Requirements

When you are using FORTRAN modules in your RMX/80 system, three types of requirements are imposed on your configuration module: enlarged Task Descriptors for floating-point operations, task stack requirements for floating-point routines, and Terminal Handler and Disk File System tasks and exchanges.

Each task in your system that performs any operations involving floating-point numbers must have a larger Task Descriptor than is usual for an RMX/80 task. If the software floating-point routines are used, 18 extra bytes must be added to the end of the Task Descriptor; if the iSBC 310 is used, 13 extra bytes. If you are coding your configuration module manually in PL/M, you need simply add these extra bytes when you declare your Task Descriptors. If you are using the assembly-language configuration macros, you must use the optional parameter *tdxtra* for the STD macro to specify the number of bytes to be added to the Task Descriptor for each task: 18 for software floating-point or 13 for hardware (iSBC 310) floating-point. Note that if you are supplying a value for *tdxtra* but are not providing a value for the preceding optional parameter, the initial exchange address *exch*, you must insert a comma to denote a null value for *exch*, as in this example:

```
STD FPTASK,60,150,,18
```

As discussed in Chapter 3 of the *RMX/80 User's Guide*, you must determine the stack size requirement for each of your tasks. For tasks written in FORTRAN that perform floating-point operations, you must take into account the bytes of stack required for the floating-point arithmetic operations and intrinsic functions you use. For a list of the stack requirements for all floating-point operations and intrinsic functions, refer to Appendix E. In addition, any task that uses FORTRAN I/O statements and/or STOP or PAUSE statements must include an extra 800 bytes of stack for I/O routines.

If you are using FORTRAN I/O for terminal input and/or output, your configuration module must include the input-output Terminal Handler and the exchanges RQINPX and RQOUTX. If you are doing FORTRAN I/O to disk files, you need the DFS services OPEN, READ, WRITE, SEEK, CLOSE, RENAME, and DELETE, and the exchanges RQOPNX, RQRNMX, RQDELX, and RQDSKX. (Note that the SEEK service is specified at link time and does not affect the configuration module.) Refer to Chapters 4 and 7 of the *RMX/80 User's Guide* for task names and other particulars. In addition, if you use any FORTRAN I/O at all you must declare the public exchange FQ0LOK, which is used by the FORTRAN I/O system, and include it in the Initial Exchange Table.

LINK Command

To run FORTRAN programs under RMX/80, you need libraries selected from two packages: the standard FORTRAN package and the RMX/80 run-time libraries. The RMX/80 package provides six libraries to support FORTRAN programs running under RMX/80:

FPSFTX.LIB	Software floating-point routines for the RMX/80 environment
FPHRDX.LIB	Routines to interface with the iSBC 310 math unit in iSBC 80/20 and 80/30 systems under RMX/80
FPHX10.LIB	Routines to interface with the iSBC 310 math unit in iSBC 80/10 systems under RMX/80
F80RMX.LIB	FORTRAN input/output routines for the RMX/80 environment
F80NTH.LIB	External reference library for RMX/80 systems that do not include the Terminal Handler
F80NDS.LIB	External reference library for RMX/80 systems that do not include the Disk File System.

Remember that even when you are using RMX/80 rather than ISIS-II at run time, you must first link your program segments together and locate them on an Intellec or Series II system using ISIS-II, which provides the LINK and LOCATE programs. If you are preparing your programs to run under RMX/80, your 'input-list' to the LINK command must include all required libraries in the sequence:

```
RMX8xx.LIB(START),object-file,F80RUN.LIB,&

{F80ISS.LIB}                {FPSOFT.LIB}                {DFS-libs  }
{F80RMX.LIB} ,.FPEF.LIB,    {FPHARD.LIB}  , {F80NDS.LIB} ,&
{F80NIO.LIB}                {FPSETX.LIB}
                            {FPHRDX.LIB}
                            {FPHX10.LIB}
                            {FPNULL.LIB}

{TH-libs     }
{mini-TH-libs} ,[ext-libs,]RMX8xx.LIB,UNRSLV.LIB,PLM80.LIB
{F80NTH.LIB  }
```

where braces { } indicate a choice of items and brackets [ ] indicate optional items.

- 'xx' in 'RMX8xx.LIB(START)' and 'RMX8xx.LIB' stands for 20, 30, or 10, for systems based on the iSBC 80/20, 80/30, and 80/10 respectively.
- 'object-files' are one or more files containing the modules produced by compiling your FORTRAN program, plus other modules (if any) translated from PL/M or assembly language code. For RMX/80 systems, 'object-files' must include your configuration module.
- Use F80ISS.LIB if your program is to run under ISIS-II and perform I/O other than port I/O, F80RMX.LIB if your modules are to run under RMX/80 and perform I/O other than port I/O, or F80NIO.LIB if only port I/O (or no I/O) is used in FORTRAN.
- Link in FPSOFT.LIB to use software floating-point for non-RMX systems, FPHARD.LIB to use the iSBC 310 interface for non-RMX systems, FPSFTX-.LIB to use software floating-point under RMX/80, FPHRDX.LIB to use the iSBC 310 interface under RMX/80 on an iSBC 80/20 or 80/30, FPHX10.LIB to use the iSBC 310 interface under RMX/80 on an iSBC 80/10, or FPNULL.LIB if no floating-point operations are used.
- If FORTRAN input or output is performed from or to a terminal, the libraries for the full input-output Terminal Handler ('TH-libs') or the minimal input-output Terminal Handler ('mini-TH-libs') must be linked in. Refer to Chapter 4 of the *RMX/80 User's Guide* for the names of these libraries. If the Terminal Handler is not needed, substitute F80NTH.LIB to resolve external references.
- If FORTRAN I/O is performed on disk files, include the 'DFS-libs' needed for the services OPEN, READ, WRITE, SEEK, CLOSE, RENAME, and DELETE. Refer to Chapter 7 of the *RMX/80 User's Guide* for the names of these libraries. If no disk I/O is performed, substitute F80NDS.LIB to resolve external references.
- 'Ext-libs' are libraries for any other RMX/80 extension services, such as the Free Space Manager or analog I/O, that you may need in your system.

For further information, refer to Appendix D.

The following sample LINK command links together an RMX/80 system that runs on an iSBC 80/20. This particular system uses the iSBC 310 interface, does FORTRAN I/O to the minimal Terminal Handler and to diskette files, and also requires the RMX/80 Free Space Manager. All the FORTRAN libraries, including PLM80.LIB, are on drive 0; the RMX/80 and user code libraries are on drive 1.

USRCOD.OBJ contains the translated code for the configuration module and all user tasks. CAMMOD.OBJ contains the translated controller-addressable memory module, which is described in Chapter 7 of the *RMX/80 User's Guide.*

```
-LINK :F1:RMX820.LIB(START),:F1:USRCOD.OBJ,F80RUN.LIB,&
**F80RMX.LIB,FPEF.LIB,FPHRDX.LIB,:F1:MT1820.LIB,:F1:MT0820.LIB,&
**:F1:DFSDIR.LIB(SEEK,DIRECTORY,DELETE,RENAME),:F1:DIO820.LIB ,&
**:F1:DFSUNR.LIB,:F1:CAMMOD.OBJ,:F1:TSK820.LIB,&
**:F1:RMX820.LIB,:F1:UNRSLV.LIB,PLM80.LIB TO :F1:USRCOD.LNK
```

(Note: The double asterisks are prompts issued by the LINK command.)

Unresolved External References

The LINK and LOCATE programs, ICE-80, and ICE-85 will generate 'error' messages for unused interrupt exchanges, as described in Chapter 3 of the *RMX/80 User's Guide*. These messages can be ignored. You should check, however, to make sure these messages refer only to interrupt exchanges and not to other unresolved external references that may be due to errors in your programs or in linking your system.

Example

The following pages provide listings of the program code for a simple RMX/80 system that displays pairs of numbers to be added, allows the user to type in answers from the terminal keyboard, checks the answers, and times the user's responses for those answers that were correct. Incorrect answers are logged on disk. At the end of the exercise, the problems missed are printed out for review, along with the average response time for correct answers.

The hardware environment is an iSBC 80/20-based system including a terminal and two disk drives (F0 and F1) on one controller (CN0) using interrupt level 2. The software environment includes the RMX/80 Nucleus, full Terminal Handler, and Disk File System.

Two user tasks are provided: TESTER, written in FORTRAN, which performs the arithmetic and I/O; and ITIMER, written in PL/M, which performs the interval timing for TESTER by means of the RMX/80 timed wait operation. Another PL/M module, called ITIMERINTERFACE, consists of two procedures (STARTT and STOPT) that are called by TESTER and that interface with ITIMER by sending messages to it to start or stop the timer.

The configuration module and the controller-addressable memory module have been written in both PL/M and assembly language; either version may be used when the programs are run. Listings of both versions are provided following the listings of TESTER, ITIMERINTERFACE, and ITIMER.

The last listing page gives the SUBMIT file used to link and locate the example system.

```
FORTRAN COMPILER    Arithmetic Testing Program

ISIS-II FORTRAN-80 V2.0 COMPILATION OF PROGRAM UNIT TESTER
OBJECT MODULE PLACED IN :F1:tester.OBJ
COMPILER INVOKED BY:  fort80 :F1:tester.ftn


   1     $date(78-Sep-14) title('Arithmetic Testing Program')
   2     $freeform

   3      subroutine Tester
         *---------------------------------------------------------------
         *---This Fortran-implemented RMX task prints 2 numbers to be
         *---added by the user, who must type an answer.  Several
         *---such pairs are typed and the number of correct answers is
         *---counted.  Incorrect answers are logged for later review.
         *---Correct answers are timed, and the average time is printed.
         *---------------------------------------------------------------

   4       external FpOGo, FpfSet, StartT, StopT, GetNum
         *---------------------------------------------------------------
         *---FpOGo and FpfSet are i/o and math initialization routines.
         *---StartT and StopT are PL/M routines to communicate to the timer.
         *---GetNum is a routine to generate the numbers for the test.
         *---------------------------------------------------------------

   5       integer N1, N2, Ans, Corect, NTries, Log
   6       real TotalT, Intrvl
         *---------------------------------------------------------------
         *---Log is the number of the i/o unit for logging wrong answers.
         *---TotalT and Intrvl are the accumulated and individual times
         *---  for responses.
         *---The rest of the variables are used in the problem itself.
         *---------------------------------------------------------------

         *---Initialize system transput (i/o) and system floating point:
   7       call FpOGo
         *---Initialize this task's floating point register:
   8       call FpfSet (0,0)

   9     1 write (*,2)
  10     2 format (///'This programs tests your ability to add pairs of numbers.'
         &  /'As soon as you see two numbers, add them and type your'
         &  /'answer.  The number of correct answers as well as the average'
         &  /'response time will be determined.  Type RETURN to start. )

  11       read (*, '()', err=1)
  12       Log = 99
  13       open (Log, file=':f0:test.log', form='unformatted'  err=1)
  14       Corect = 0
  15       NTries = 20
  16       TotalT = 0.0

  17       do 3, IthTry = 1, NTries
         *---Get two numbers for the example, print them, & start timer.
  18          call GetNum (N1)
  19          call GetNum (N2)
  20          write (*,'(//i6/1h+,i5)') N1, N2
```

```
FORTRAN COMPILER    Arithmetic Testing Program


  21          call StartT

              *---Get response, stop timer, check answer, & log bad response.
  22          read (*, '(i6)', err = 13) Ans
  23      13  call StopT (Intrvl)
  24          if (N1+N2 .eq. Ans) then
  25            Corect = Corect + 1
  26            TotalT = TotalT + Intrvl
  27          else
  28            write (Log) N1, N2, Ans
  29          end if
  30      3   continue

              *---Print results of the exercise.  Print problems missed.
  31        write (*, '(////''End of exercise.'')')
  32        if (Corect .eq. NTries) then
  33          write(*, '(''All ',i4,'' answers were correct! Very good!'')') Corect
  34        else
  35          write(*, '(i3,'' problem(s) wrong out of ',i4)') NTries-Corect, NTries
  36          write(*, '(///''Review the following problems:'')')
  37          rewind (Log)
  38          do 5, IthTry = 1, NTries-Corect
  39            read (Log) N1, N2, Ans
  40            write (*,4) N1, N2, N1+N2, Ans
  41      4       format (/i9/2h +,i7/2h =,i7,' not',i8)
  42      5     continue
  43          end if

              *---Print average response time for correct answers:
  44        write (*,6) TotalT / Corect
  45      6   format (///' Your average time per correct answer was ,f8.3,  sec.')
  46        close (Log, status='delete')

              *---Begin again:
  47        go to 1
  48        end


MODULE INFORMATION:

     CODE AREA SIZE     = 05CDH    1485D
     VARIABLE AREA SIZE = 0042H      66D
     MAXIMUM STACK SIZE = 0004H       4D
     87 LINES READ

     0 PROGRAM ERROR(S) IN PROGRAM UNIT TESTER
```

```
FORTRAN COMPILER    Arithmetic Testing Program

ISIS-II FORTRAN-80 V2.0 COMPILATION OF PROGRAM UNIT GETNUM
OBJECT MODULE PLACED IN :F1:tester.OBJ
COMPILER INVOKED BY:  fort80 :F1:tester ftn



   1      subroutine GetNum (N)
         *---This Fortran subroutine naively generates some numbers for Tester.

   2        integer N, LastN
   3        common LastN

   4       LastN = iabs(mod(13*LastN+1,1999))
   5       N = LastN
   6       end


MODULE INFORMATION:

     CODE AREA SIZE     = 0027H     39D
     VARIABLE AREA SIZE = 0002H      2D
     MAXIMUM STACK SIZE = 0002H      2D
     12 LINES READ

     0 PROGRAM ERROR(S) IN PROGRAM UNIT GETNUM

     0 TOTAL PROGRAM ERROR(S)
     END OF FORTRAN COMPILATION
```

```
PL/M-80 COMPILER    INTERVAL TIMER INTERFACE


ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE ITIMERINTERFACE
OBJECT MODULE PLACED IN :F2:testt.OBJ
COMPILER INVOKED BY:  plm80 :F2:testt.plm


                $date('78-Sep-12') title('Interval Timer Interface')

  1             ITimerInterface: do;
                  /*****************************************************************
                  /* This PL/M module interfaces the Fortran subroutine Tester to the
                  /* interval timer.  This module takes care of the RMX/80 message
                  /* sending/receiving for the Fortran subroutine.
                  /*
                  /* This module/task sends the following messages:
                  /*  sends     MSG          TO          VIA          COMMENTS
                  /*        StartTimer    ITimer      TCntrl  Signals timer to start
                  /*         StopTimer    ITimer      TCntrl  Signals timer to stop
                  /*
                  /* This module/task waits for the following messages:
                  /*  receives MSG          FROM         VIA          COMMENTS
                  /*  (ref'd by MsgAdr)  ITimer      TRslts  Indicates timer running
                  /*  (ref'd by MsgAdr)  ITimer      TRslts  Returns elapsed seconds (REAL)
                  /*****************************************************************/

                $include (:f1:Synch.Ext)
  2   1   =     RQSEND:
          =       PROCEDURE (EXCHANGE$POINTER,MESSAGE$POINTER) EXTERNAL;
  3   2   =         DECLARE (EXCHANGE$POINTER,MESSAGE$POINTER) ADDRESS;
          =
  4   2   =       END RQSEND;
          =
  5   1   =     RQWAIT:
          =       PROCEDURE (EXCHANGE$POINTER,DELAY) ADDRESS EXTERNAL;
  6   2   =         DECLARE (EXCHANGE$POINTER,DELAY) ADDRESS;
          =
  7   2   =       END RQWAIT;
          =
  8   1   =     RQACPT:
          =       PROCEDURE (EXCHANGE$POINTER) ADDRESS EXTERNAL;
  9   2   =         DECLARE EXCHANGE$POINTER ADDRESS;
          =
 10   2   =       END RQACPT;
          =
 11   1   =     RQISND:
          =       PROCEDURE (IED$PTR) EXTERNAL;
 12   2   =         DECLARE IED$PTR ADDRESS;
          =
 13   2   =       END RQISND;
                $include (:f1:Exch.Elt)
 14   1   =     DECLARE EXCHANGE$DESCRIPTOR LITERALLY 'STRUCTURE (
          =      MESSAGE$HEAD ADDRESS,
          =      MESSAGE$TAIL ADDRESS,
          =      TASK$HEAD ADDRESS,
          =      TASK$TAIL ADDRESS,
          =      EXCHANGE$LINK ADDRESS)';
                $include (:f1:Msg.Elt)
```

```
PL/M-80 COMPILER    INTERVAL TIMER INTERFACE

 15   1  =   DECLARE MSG$HDR LITERALLY
          =    'LINK ADDRESS,
          =    LENGTH ADDRESS,
          =    TYPE BYTE,
          =    HOME$EXCHANGE ADDRESS,
          =    RESPONSE$EXCHANGE ADDRESS';
          =
 16   1  =   DECLARE MSG$DESCRIPTOR LITERALLY 'STRUCTURE(
          =    MSG$HDR,
          =    REMAINDER(1) BYTE)';

 17   1      declare
               integer2 literally 'address',
               real literally '(2)address',
               TMsg structure (
                 Msg$Hdr,
                 Duration real)  /*-Message for communicating with timer-*/,
               TRslts Exchange$Descriptor public /*-Exchange for interval timer results-*/,
               TCntrl Exchange$Descriptor external /*-Exchange to start/stop timer.-*/,
               MsgAdr address /*-Pointer to 'result' message -*/;

 18   1      declare /*-Message types-*/
               StartTimer   literally '101',
               StopTimer    literally '100';

 19   1      StartT: procedure public;
               /*-------------------------------------------------------------------
               /* This PL/M procedure is called from the Fortran subroutine Tester when
               /* the interval timer is to be started.  This procedure forms the 'start'
               /* message, sends it to the exchange (TCntrl) controlling the timer, and
               /* waits for an acknowledgement before returning to Tester.
               /*-------------------------------------------------------------------*/
 20   2         TMsg.Length = 13;
 21   2         TMsg.Type = StartTimer;
 22   2         TMsg.Response$Exchange = .TRslts;
 23   2         call RqSend (.TCntrl, .TMsg);
                 /*- Send the message and wait for acknowledgement.-*/
 24   2         MsgAdr = RqWait (.TRslts, 0);
 25   2         end StartT;


 26   1      StopT: procedure (SecondsAdr) public;
               /*-------------------------------------------------------------------
               /* This PL/M procedure is called from the Fortran subroutine Tester in
               /* order to stop the interval timer and to obtain the elapsed time since
               /* the timer was started.  This procedure forms the 'stop' message,
               /* sends it to the exchange controlling the timer, and waits for
               /* the resultant elapsed time (a floating-point, or REAL, value) to be
               /* returned.
               /*-------------------------------------------------------------------*/
 27   2         declare SecondsAdr address,
                   Seconds based SecondsAdr real;
 28   2         TMsg.Type = StopTimer;
 29   2         call RqSend (.TCntrl, .TMsg);
 30   2         MsgAdr = RqWait (.TRslts, 0);
 31   2         Seconds(0) = TMsg.Duration(0); Seconds(1) = TMsg.Duration(1);
```

```
PL/M-80 COMPILER    INTERVAL TIMER INTERFACE

  33   2         end StopT;

  34   1       end ITimerInterface;


MODULE INFORMATION:

     CODE AREA SIZE      = 0062H      98D
     VARIABLE AREA SIZE  = 001BH      27D
     MAXIMUM STACK SIZE  = 0002H       2D
     111 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
PL/M-80 COMPILER     INTERVAL TIMER TASK

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE INTERVALTIMER
OBJECT MODULE PLACED IN :F2:itimer.OBJ
COMPILER INVOKED BY:  plm80 :F2:itimer.plm


                $date(78 Aug 9) title('Interval Timer Task')

   1            IntervalTimer: do;
                  /*************************************************************************
                  /* This PL/M procedure is the main procedure of a task that performs
                  /* interval timing for the Fortran subprogram Tester which runs as
                  /* another task.  The timing is simply performed by using the time-out
                  /* facility of the RMX/80 wait operation.  (The wait waits at an
                  /* exchange to which no messages are send.)
                  /*
                  /* This task/procedure receives the following messages:
                  /*  receives  MSG          FROM     VIA         COMMENT
                  /*         StartTimer     Tester  TCntrl  Starts operation of timer
                  /*         StopTimer      Tester  TCntrl  Requests elapsed time
                  /*
                  /* This task/procedure sends (returns) the following messages:
                  /*  sends      MSG          FROM     VIA         COMMENT
                  /*    (acknowledge Start) Tester (resp.exch) Returns message as confirmation
                  /*    (acknowledge Stop) Tester  (resp.exch) Returns message with time in it
                  /*************************************************************************/

                $include (:f1:Exch.lit)
   2    1  =    DECLARE EXCHANGE$DESCRIPTOR LITERALLY 'STRUCTURE (
             =     MESSAGE$HEAD ADDRESS,
             =     MESSAGE$TAIL ADDRESS,
             =     TASK$HEAD ADDRESS,
             =     TASK$TAIL ADDRESS,
             =     EXCHANGE$LINK ADDRESS)';
                $include (:f1:Msg.lit)
   3    1  =    DECLARE MSG$HDR LITERALLY '
             =     LINK ADDRESS,
             =     LENGTH ADDRESS,
             =     TYPE BYTE,
             =     HOME$EXCHANGE ADDRESS,
             =     RESPONSE$EXCHANGE ADDRESS';
             =
   4    1  =    DECLARE MSG$DESCRIPTOR LITERALLY 'STRUCTURE(
             =     MSG$HDR,
             =     REMAINDER(1) BYTE)';

   5    1       declare
                  zero address data (0),
                  integer2 literally 'address',
                  real literally '(2)address';

   6    1       declare /*Exchanges*/
                  Ticker Exchange$Descriptor public,
                  TCntrl Exchange$Descriptor public;

   7    1       declare /*Messages*/
                  ControlMsgAdr address,
```

```
PL/M-80 COMPILER    INTERVAL TIMER TASK

                  ControlMsg based ControlMsgAdr structure (
                    MsgHdr,
                    Duration real),
                  TickMsgAdr address;

   8    1       declare /*Message types*/
                  StartTimer   literally '101',
                  StopTimer    literally '100';

   9    1       ToSecs: /*Coversion routine for time format*/
                  procedure (R,M,S) external;
  10    2         declare (R,M,S) address; end;

  12    1       FqfSet: /*Task's floating point initialization*/
                  procedure (F,E) external;
  13    2         declare (F,E) address; end;

                  $include (:f1:Synch.Ext)
  15    1   =   RQSEND:
        =         PROCEDURE (EXCHANGE$POINTER,MESSAGE$POINTER) EXTERNAL;
  16    2   =       DECLARE (EXCHANGE$POINTER,MESSAGE$POINTER) ADDRESS;
        =
  17    2   =     END RQSEND;
        =
  18    1   =   RQWAIT:
        =         PROCEDURE (EXCHANGE$POINTER,DELAY) ADDRESS EXTERNAL;
  19    2   =       DECLARE (EXCHANGE$POINTER,DELAY) ADDRESS;
        =
  20    2   =     END RQWAIT;
        =
  21    1   =   RQACPT:
        =         PROCEDURE (EXCHANGE$POINTER) ADDRESS EXTERNAL;
  22    2   =       DECLARE EXCHANGE$POINTER ADDRESS;
        =
  23    2   =     END RQACPT;
        =
  24    1   =   RQISND:
        =         PROCEDURE (IED$PTR) EXTERNAL;
  25    2   =       DECLARE IED$PTR ADDRESS;
        =
  26    2   =     END RQISND;
```

```
PL/M-80 COMPILER    INTERVAL TIMER TASK


                $eject
  27    1       ITimer: procedure public;
                  /*****************************************************************
                  /* This procedure is the entry point of the interval timer task.
                  /* It is used to clock a period of time with a resolution of 50 msec
                  /* (under RMX/80/20 or 80/30).  The duration of such a period is
                  /* returned as a real (floating-point) value measured in seconds.
                  /*****************************************************************/

  28    2         declare
                    (Minutes, TwentiethSecs) integer2;

                  /*Initialize floating point for this task*/
  29    2         call FqfSet (.zero,.zero);
  30    2         do while 1;
                    /*Wait for some request to start*/
  31    3           ControlMsgAdr = RqWait (.TCntrl, 0);
  32    3           TwentiethSecs = 0; Minutes = 0;
  34    3           ControlMsg.Duration(0) = 0; ControlMsg.Duration(1) = 0;
  36    3           call RqSend (ControlMsg.Response$Exchange, ControlMsgAdr);
  37    3           ControlMsgAdr = 0;
  38    3           do while ControlMsgAdr = 0;
  39    4             TickMsgAdr = RqWait (.Ticker, 1); /*0.05 seconds on 80/20*/
  40    4             TwentiethSecs = TwentiethSecs + 1;
  41    4             if TwentiethSecs = 1200 then do;
  43    5               Minutes = Minutes + 1; TwentiethSecs = 0; end;
  46    4             ControlMsgAdr = RqAcpt (.TCntrl);
  47    4             end;

  48    3           call ToSecs (.ControlMsg.Duration, .Minutes, .TwentiethSecs);
  49    3           call RqSend (ControlMsg.ResponseExchange, ControlMsgAdr);
  50    3           end;
  51    2         end ITimer;

  52    1       end IntervalTimer;


MODULE INFORMATION:

     CODE AREA SIZE     = 00B9H    185D
     VARIABLE AREA SIZE = 001CH     28D
     MAXIMUM STACK SIZE = 0004H      4D
     127 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
FORTRAN COMPILER    Conversion routine for Interval Timer


ISIS-II FORTRAN-80 V2.0 COMPILATION OF PROGRAM UNIT TOSECS
OBJECT MODULE PLACED IN :F1:ToSecs.OBJ
COMPILER INVOKED BY:  fort80 :f1:ToSecs.ftn


   1      $date(78 Aug 9) title('Conversion routine for Interval Timer')
   2            real function ToSecs (Min, Sec20)
                *      ..This Fortran-80 function subprogram converts a pair of integer
                *      ..values representing some number of minutes and some multiple of
                *      ..twentieths of a second to a real value representing seconds.
   3            integer*2 Min, Sec20
   4            ToSecs = 60.0 * Min  +  Sec20 / 20.0
   5            end


MODULE INFORMATION:

     CODE AREA SIZE     = 004CH     76D
     VARIABLE AREA SIZE = 0006H      6D
     MAXIMUM STACK SIZE = 0006H      6D
     8 LINES READ

     0 PROGRAM ERROR(S) IN PROGRAM UNIT TOSECS

     0 TOTAL PROGRAM ERROR(S)
     END OF FORTRAN COMPILATION
```

```
PL/M-80 COMPILER

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE CONFIGURATIONMODULE
OBJECT MODULE PLACED IN :F2:config.OBJ
COMPILER INVOKED BY:  plm80 :F2:config.plm


   1          CONFIGURATION$MODULE: DO;

              /*************************************************************************/
              /* This configuration module describes the hardware environment to  */
              /* RMX/80.  The environment assumed includes two disk drives, F0 and
              /* F1, on one controller CN0 using interrupt level 2 on an iSBC 80/20.
              /* Also included is a terminal driven by the full terminal handler.
              /*
              /* The software environment assumed includes the Fortran-80 Run-time
              /* for RMX/80 Systems which uses the FCOLOK exchange.  Two user
              /* tasks are also specified, which use three exchanges.
              /*************************************************************************/

   2   1      DECLARE FP$TASK$DESCRIPTOR LITERALLY
                   /*************************************/
                   /* Note this new kind of TD for tasks */
                   /* performing floating-point math     */
                   /*************************************/ 'STRUCTURE(
              DELAY$LINK$FORWARD ADDRESS,
              DELAY$LINK$BACK ADDRESS,
              THREAD ADDRESS,
              DELAY ADDRESS,
              EXCHANGE$ADDRESS ADDRESS,
              SP ADDRESS,
              MARKER ADDRESS,
              PRIORITY BYTE,
              STATUS BYTE,
              NAME$PTR ADDRESS,
              TASK$LINK ADDRESS,
              FPR(16)BYTE)';

              $nolist

              /***** task entry points *****/
  11   1        RQTHDI: PROCEDURE EXTERNAL; END;    /*** terminal handler ***/

  13   1        RQPDSK: PROCEDURE EXTERNAL; END;    /*** disk input/output ***/
  15   1        RQPDEL: PROCEDURE EXTERNAL; END;      /*** file delete service ***/
  17   1        RQPDIR: PROCEDURE EXTERNAL; END;      /*** disk directory service ***/
  19   1        RQPRNM: PROCEDURE EXTERNAL; END;      /*** file rename service ***/
  21   1        RQHD1:  PROCEDURE EXTERNAL; END;      /*** controller for disks ***/

  23   1        ITIMER: PROCEDURE EXTERNAL; END;    /*** interval timer ***/
  25   1        TESTER: PROCEDURE EXTERNAL; END;    /*** user main task ***/


  27   1      DECLARE  /**** task stack lengths ****/
                 TH$STK$LEN  LITERALLY '36',

                 DSK$STK$LEN LITERALLY '48',
                 DEL$STK$LEN LITERALLY '64',
```

```
PL/M-80 COMPILER

                DIR$STK$LEN LITERALLY '48',
                RNM$STK$LEN LITERALLY '64',
                CNO$STK$LEN LITERALLY '80',

                ITIMER$STK$LEN LITERALLY '300',
                TESTER$STK$LEN LITERALLY '1000';


  28   1       DECLARE  /**** task stacks ****/
                TH$STK (TH$STK$LEN) BYTE,

                DSK$STK (DSK$STK$LEN) BYTE,
                DEL$STK (DEL$STK$LEN) BYTE,
                DIR$STK (DIR$STK$LEN) BYTE,
                RNM$STK (RNM$STK$LEN) BYTE,
                CNO$STK (CNO$STK$LEN) BYTE EXTERNAL,

                ITIMER$STK (ITIMER$STK$LEN) BYTE,
                TESTER$STK (TESTER$STK$LEN) BYTE;


  29   1       DECLARE  /**** task priorities ****/
                TH$PRI LITERALLY '112',

                DSK$PRI LITERALLY '129',
                DEL$PRI LITERALLY '140',
                DIR$PRI LITERALLY '135',
                RNM$PRI LITERALLY '145',
                CNO$PRI LITERALLY ' 33',

                ITIMER$PRI LITERALLY '100',
                TESTER$PRI LITERALLY '200';


  30   1       DECLARE  /**** task descriptors ****/
                TH$TD TASK$DESCRIPTOR,

                DSK$TD TASK$DESCRIPTOR,
                DEL$TD TASK$DESCRIPTOR,
                DIR$TD TASK$DESCRIPTOR,
                RNM$TD TASK$DESCRIPTOR,
                CNO$TD TASK$DESCRIPTOR,

                ITIMER$TD FP$TASK$DESCRIPTOR,
                TESTER$TD FP$TASK$DESCRIPTOR;


  31   1       DECLARE  /**** static task descriptors ****/
                N$TASKS LITERALLY '8',
                INITIAL$TASK$TABLE (N$TASKS) STATIC$TASK$DESCRIPTOR DATA(
                 /*_name____procedure___stack____stack_size___priority__default__TD_addr_*/

                     /**** std for input-output terminal handler ****/
                   'RQTHDI',  .RQTHDI,  .TH$STK,  TH$STK$LEN,  TH$PRI,  .RQOUTX,  .TH$TD,

                     /**** stds for disk file system services ****/
                   'DISKIO',  .RQPDSK,  .DSK$STK, DSK$STK$LEN, DSK$PRI, .RQDSKX, .DSK$TD,
```

```
PL/M-80 COMPILER

            'DELETE',  .RQPDEL,  .DEL$STK, DEL$STK$LEN, DEL$PRI, .RQDELX, .DEL$TD,
            'DIRSVC',  .RQPDIR,  .DIR$STK, DIR$STK$LEN, DIR$PRI, .RQDIRX, .DIR$TD,
            'RENAME',  .RQPRNM,  .RNM$STK, RNM$STK$LEN, RNM$PRI, .RQRNMX, .RNM$TD,
            'DC201A',  .RQHD1,   .CNO$STK, CNO$STK$LEN, CNO$PRI, .CNOX,   .CNO$TD,

               /**** stds for user tasks ****/
            'ITIMER',.ITIMER,.ITIMER$STK,ITIMER$STK$LEN,ITIMER$PRI,.TICKER,.ITIMER$TD,
            'TESTER',.TESTER,.TESTER$STK,TESTER$STK$LEN,TESTER$PRI,  0    ,.TESTER$TD);


 32  1      DECLARE  /**** system exchange descriptors ****/
              (RQINPX, RQOUTX
               , RQDBUG, RQWAKE, RQALRM)  EXCHANGE$DESCRIPTOR EXTERNAL  /** terminal **/,
              (RQDSKX, RQDELX, RQDIRX, RQRNMX)  EXCHANGE$DESCRIPTOR EXTERNAL  /** disk **/,
              (CNOX) EXCHANGE$DESCRIPTOR  /** exchange for disk controller **/,
              (RQL2EX) INT$EXCHANGE$DESCRIPTOR PUBLIC,
                (RQL0EX,RQL3EX,RQL4EX,RQL5EX) INT$EXCHANGE$DESCRIPTOR PUBLIC AT (.RQL2EX),
                (RQL6EX, RQL7EX) INT$EXCHANGE$DESCRIPTOR EXTERNAL  /** interrupts **/,
              FQOLOK EXCHANGE$DESCRIPTOR PUBLIC  /** fortran i/o interlock **/;

 33  1      DECLARE  /**** user exchange descriptors ****/
              (TICKER, TCNTRL, TRSLTS)  EXCHANGE$DESCRIPTOR EXTERNAL;


 34  1      DECLARE  /**** initial exchange table ****/
              N$EXCHANGES LITERALLY '17',
              INITIAL$EXCHANGE$TABLE (N$EXCHANGES) ADDRESS DATA(
                .RQINPX, .RQOUTX, .RQDBUG, .RQWAKE, .RQALRM,    /** terminal handler **/
                .RQDSKX, .RQDELX, .RQDIRX, .RQRNMX,   /** disk service exchanges **/
                .RQL2EX, .RQL6EX, .RQL7EX,   /** interrupt exchanges **/
                .CNOX,    /** controller exchanges **/
                .FQOLOK,    /** fortran i/o interlock exchange **/
                .TICKER, .TCNTRL, .TRSLTS)   /** user exchanges **/;


 35  1      DECLARE  /**** create table ****/
              RQCRTB  CREATE$TABLE PUBLIC DATA(
                .INITIAL$TASK$TABLE, N$TASKS,
                .INITIAL$EXCHANGE$TABLE, N$EXCHANGES);


 36  1      DECLARE  /**** controller specification table ****/
              SBC201 LITERALLY '0'  /** device type **/,
              RQCST (1) CST$ENTRY PUBLIC DATA(
                SBC201, /*port*/ 78H, /*int.level*/ 2, /*int.exchange*/ .RQL2EX
                   , /*request exchange*/ .CNOX);


 37  1      DECLARE  /**** device configuration table ****/
              RQNDEV ADDRESS PUBLIC DATA (2)   /** 2 drives **/,
              RQDCT (2) DCT$ENTRY PUBLIC DATA(
                'F0', SBC201, /*controller*/ 0, /*unit*/ 0,
                'F1', SBC201,                0,          1);
```

```
PL/M-80 COMPILER

  38   1       DECLARE  /**** buffer allocation block ****/
                  BUFPOL (3200) BYTE EXTERNAL,
                  RQBAE BAB$ENTRY PUBLIC DATA(
                    0,  0,  /*n files*/ 8, .BUFPOL)  /** statically allocated buffers **/;


  39   1       DECLARE  /**** public data ****/
                  RQRATE  ADDRESS PUBLIC DATA (28) /** 2400 baud **/;


  40   1       END CONFIGURATION$MODULE;


MODULE INFORMATION:

     CODE AREA SIZE     = 00CCH    204D
     VARIABLE AREA SIZE = 06FFH   1791D
     MAXIMUM STACK SIZE = 0000H      0D
     255 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
PL/M-80 COMPILER


ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE CONTROLLERACCESSMEMORY
OBJECT MODULE PLACED IN :F2:cammod.OBJ
COMPILER INVOKED BY:  plm80 :F2:cammod.plm


   1          CONTROLLER$ACCESS$MEMORY: DO;
              /*****************************************************************/
              /* This module describes a data area which must be in contoller
              /* accessable memory.  That is, the data segment of this module must
              /* be located for off-board memory (e.g. 0F000h).
              /*****************************************************************/

   2    1     DECLARE  /**** stack for controller task ****/
                CN0$STK$SIZE LITERALLY '60',
                CN0$STK (60) BYTE PUBLIC;


   3    1     DECLARE  /**** buffer for DIRSVC directory services ****/
                RQDBUF (700)BYTE PUBLIC;

   4    1     DECLARE  /**** buffers for up to 8 open files ****/
                BUFPOL (3200)BYTE PUBLIC;

   5    1     END CONTROLLER$ACCESS$MEMORY;


MODULE INFORMATION:

     CODE AREA SIZE      = 0000H       0D
     VARIABLE AREA SIZE  = 0F8CH    3960D
     MAXIMUM STACK SIZE  = 0000H       0D
     22 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
asm80 :f2:config.asm

ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0        CONFIG   PAGE    1

  LOC  OBJ          SEQ         SOURCE STATEMENT

                      1 $MACROFILE
                      2 $NOGEN
                      3         NAME CONFIG
                      4 ;*****************************************************************
                      5 ; This configuration module describes the hardware environment to
                      6 ; RMX/80.  The environment assumed includes two disk drives, F0 and
                      7 ; F1, on one controller CN0 using interrupt level 2 on an iSBC 80/20
                      8 ; Also included is a terminal driven by the full terminal handler.
                      9 ;
                     10 ; The software environment assumed includes the Fortran-80 Run-time
                     11 ; for RMX/80 Systems which used the FQOLOK exchange.  Two user
                     12 ; tasks are also specified, which use three exchanges.
                     13 ;*****************************************************************
                     14         CSEG
                     15 ;
                     16 ; INCLUDE MACROS
                     17 ;
                     18 $nolist
                    394 $list
                    395 ;
                    396 ;
0000                397 NTASK   SET     0
0000                398 NEXCH   SET     0
0000                399 NDEV    SET     0
0000                400 NCONT   SET     0
                    401 ;
                    402 ;
                    403 ;
                    404 ;
                    405 ;*****************************************************************
                    406 ; Static task descriptors:
                    407 ;
                    408         STD     RQTHDI,36,112,RQOUTX    ; terminal handler
0000 52515448       501+        DB      'RQTHDI'
0004 4449             +
0006 0000      E    512+        DW      RQTHDI
0008 0000      D    513+        DW      ??0001
000A 2400           514+        DW      36
000C 70             515+        DB      112
000D 0000      E    516+        DW      RQOUTX+0
000F 1806      D    517+        DW      TDBASE+TDAREA        ;**** CHANGED FROM RMX V1.3
0001                518+NTASK   SET     NTASK+1
0014                519+TDAREA  SET     +20+TDAREA      ;**** CHANGED FROM RMX V1.3
                    520 ;
                    521         STD     RQPDSK,48,129,RQDSKX    ; disk input/output services
                    522+        EXTRN   RQPDSK
                    523+        CSEG
                    528+        DSEG
0030                529+??0002: DS      48
                    530+        CSEG
0011 52515044       614+        DB      'RQPDSK'
0015 534B             +
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0        CONFIG   PAGE    2

  LOC  OBJ          SEQ          SOURCE STATEMENT

  0017 0000      E   625+         DW      RQPDSK
  0019 2400      D   626+         DW      ??0002
  001B 3000          627+         DW      48
  001D 81            626+         DB      129
  001E 0000      E   629+         DW      RQDSKX+0
  0020 2C06      D   630+         DW      TDBASE+TDAREA        ;**** CHANGED FROM RMX V1.3
  0002               631+NTASK    SET     NTASK+1
  0028               632+TDAREA   SET     +20+TDAREA      ;**** CHANGED FROM RMX V1.3
                     633          STD     RQPDEL,64,140,RQDELX      ; file delete service
                     634+         EXTRN   RQPDEL
                     635+         CSEG
                     640+         DSEG
  0040               641+??0003:  DS      64
                     642+         CSEG
  0022 52515044      726+         DB      'RQPDEL'
  0026 454C            +
  0028 0000      E   737+         DW      RQPDEL
  002A 5400      D   738+         DW      ??0003
  002C 4000          739+         DW      64
  002E 8C            740+         DB      140
  002F 0000      E   741+         DW      RQDELX+0
  0031 4006      D   742+         DW      TDBASE+TDAREA        ;**** CHANGED FROM RMX V1.3
  0003               743+NTASK    SET     NTASK+1
  003C               744+TDAREA   SET     +20+TDAREA      ;**** CHANGED FROM RMX V1.3
                     745          STD     RQPDIR,48,135,RQDIRX      ; disk directory service
                     746+         EXTRN   RQPDIR
                     747+         CSEG
                     752+         DSEG
  0030               753+??0004:  DS      48
                     754+         CSEG
  0033 52515044      838+         DB      'RQPDIR'
  0037 4952            +
  0039 0000      E   849+         DW      RQPDIR
  003B 9400      D   850+         DW      ??0004
  003D 3000          851+         DW      48
  003F 87            852+         DB      135
  0040 0000      E   853+         DW      RQDIRX+0
  0042 5406      D   854+         DW      TDBASE+TDAREA        ;**** CHANGED FROM RMX V1.3
  0004               855+NTASK    SET     NTASK+1
  0050               856+TDAREA   SET     +20+TDAREA      ;**** CHANGED FROM RMX V1.3
                     857          STD     RQPRNM,64,145,RQRNMX      ; file rename service
                     858+         EXTRN   RQPRNM
                     859+         CSEG
                     864+         DSEG
  0040               865+??0005:  DS      64
                     866+         CSEG
  0044 52515052      950+         DB      'RQPRNM'
  0048 4E4D            +
  004A 0000      E   961+         DW      RQPRNM
  004C C400      D   962+         DW      ??0005
  004E 4000          963+         DW      64
  0050 91            964+         DB      145
  0051 0000      E   965+         DW      RQRNMX+0
  0053 6806      D   966+         DW      TDBASE+TDAREA        ;**** CHANGED FROM RMX V1.3
  0005               967+NTASK    SET     NTASK+1
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0        CONFIG   PAGE    3


  LOC  OBJ          SEQ          SOURCE STATEMENT

  0064               966+TDAREA  SET     +20+TDAREA      ;**** CHANGED FROM RMX V1.3
                     969         EXTRN   RQHD1                   ; controller for disks
                     970         CONSTD  CNTRL1,RQHD1,80,CNOSTK,33,CNOX
  0055 434E5452     1062+        DB     'CNTRL1'
  0059 4C31            +
  005B 0000      E  1073+        DW      RQHD1
  005D 0000      E  1074+        DW      CNOSTK
  005F 5000         1075+        DW      80
  0061 21           1076+        DB      33
  0062 DC06      D  1077+        DW      CNOX+0
  0064 7C06      D  1078+        DW      TDBASE+TDAREA       ;**** CHANGED FROM RMX V1.3
                    1083 ;
                    1084         STD     ITIMER,300,100,TICKER,18        ; interval timer user task
                    1085+        EXTRN   ITIMER
                    1086+        CSEG
                    1091+        DSEG
  012C              1092+??0006: DS      300
                    1093+        CSEG
  0066 4954494D     1177+        DB      'ITIMER'
  006A 4552            +
  006C 0000      E  1188+        DW      ITIMER
  006E 0401      D  1189+        DW      ??0006
  0070 2C01         1190+        DW      300
  0072 64           1191+        DB      100
  0073 0000      E  1192+        DW      TICKER+0
  0075 9006      D  1193+        DW      TDBASE+TDAREA       ;**** CHANGED FROM RMX V1.3
  0007              1194+NTASK   SET     NTASK+1
  009E              1195+TDAREA  SET     18+20+TDAREA    ;**** CHANGED FROM RMX V1.3
                    1196         STD     TESTER,1000,200,,18     ; user main task
                    1197+        EXTRN   TESTER
                    1198+        CSEG
                    1203+        DSEG
  03E8              1204+??0007: DS      1000
                    1205+        CSEG
  0077 54455354     1289+        DB      'TESTER'
  007B 4552            +
  007D 0000      E  1300+        DW      TESTER
  007F 3002      D  1301+        DW      ??0007
  0081 E803         1302+        DW      1000
  0083 C8           1303+        DB      200
  0084 0000         1304+        DW      +0
  0086 B606      D  1305+        DW      TDBASE+TDAREA       ;**** CHANGED FROM RMX V1.3
  0008              1306+NTASK   SET     NTASK+1
  00C4              1307+TDAREA  SET     18+20+TDAREA    ;**** CHANGED FROM RMX V1.3
                    1308         GENTD
                    1309+        DSEG
  00C4              1310+TDBASE: DS      TDAREA      ;**** CHANGED FROM RMX V1.3
                    1311+        CSEG
                    1312 ;
                    1313 ;*************************************************************************
                    1314 ; Exchanges for disk controller and interrupts
                    1315 ;
                    1316         XCH     CNOX
                    1317+        DSEG
                    1318+        PUBLIC  CNOX
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0        CONFIG   PAGE    4

  LOC  OBJ          SEQ          SOURCE STATEMENT

  000A              1319+CNOX:   DS       10
                    1320         INTXCH   RQL2EX
                    1321+        XCH       RQL2EX
                    1322+        DSEG
                    1323+        PUBLIC   RQL2EX
  000A              1324+RQL2EX:   DS       10
  0005              1325+        DS       5
                    1326         PUBLIC   RQL3EX,RQL4EX,RQL5EX,RQLOEX
  06E6          D   1327 RQL3EX  EQU      RQL2EX
  06E6          D   1328 RQL4EX  EQU      RQL2EX
  06E6          D   1329 RQL5EX  EQU      RQL2EX
  06E6          D   1330 RQLOEX  EQU      RQL2EX
                    1331 ;
                    1332 ; Exchange for Fortran i/o system:
                    1333         XCH      FQOLOK
                    1334+        DSEG
                    1335+        PUBLIC   FQOLOK
  000A              1336+FQOLOK:   DS       10
                    1337 ;
                    1338 ;*************************************************************************
                    1339 ; Initial exchange table:
                    1340         XCHADR   RQINPX
                    1341+        EXTRN    RQINPX
                    1342+        CSEG
                    1344+IET:
  0086 0000     E   1346+        DW       RQINPX
  0001              1347+NEXCH   SET      NEXCH+1
                    1348         XCHADR   RQOUTX
                    1349+        EXTRN    RQOUTX
                    1350+        CSEG
  0088 0000     E   1354+        DW       RQOUTX
  0002              1355+NEXCH   SET      NEXCH+1
                    1356         XCHADR   RQWAKE
                    1357+        EXTRN    RQWAKE
                    1358+        CSEG
  008C 0000     E   1362+        DW       RQWAKE
  0003              1363+NEXCH   SET      NEXCH+1
                    1364         XCHADR   RQDBUG
                    1365+        EXTRN    RQDBUG
                    1366+        CSEG
  008E 0000     E   1370+        DW       RQDBUG
  0004              1371+NEXCH   SET      NEXCH+1
                    1372         XCHADR   RQALRM
                    1373+        EXTRN    RQALRM
                    1374+        CSEG
  0090 0000     E   1378+        DW       RQALRM
  0005              1379+NEXCH   SET      NEXCH+1
                    1380 ;
                    1381         XCHADR   RQDSKX
                    1382+        EXTRN    RQDSKX
                    1383+        CSEG
  0092 0000     E   1387+        DW       RQDSKX
  0006              1388+NEXCH   SET      NEXCH+1
                    1389         XCHADR   RQDELX
                    1390+        EXTRN    RQDELX
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0        CONFIG   PAGE    5

  LOC  OBJ          SEQ          SOURCE STATEMENT

                    1391+         CSEG
  0094 0000      E  1395+         DW       RQDELX
  0007              1396+NEXCH    SET      NEXCH+1
                    1397          XCHADR   RQDIRX
                    1398+         EXTRN    RQDIRX
                    1399+         CSEG
  0096 0000      E  1403+         DW       RQDIRX
  0008              1404+NEXCH    SET      NEXCH+1
                    1405          XCHADR   RQRNMX
                    1406+         EXTRN    RQRNMX
                    1407+         CSEG
  0098 0000      E  1411+         DW       RQRNMX
  0009              1412+NEXCH    SET      NEXCH+1
                    1413 ;
                    1414          PUBXCH   RQL2EX
  009A E606      D  1419+         DW        RQL2EX
  000A              1420+NEXCH    SET       NEXCH+1
                    1421          XCHADR   RQL6EX
                    1422+         EXTRN    RQL6EX
                    1423+         CSEG
  009C 0000      E  1427+         DW       RQL6EX
  000B              1428+NEXCH    SET      NEXCH+1
                    1429          XCHADR   RQL7EX
                    1430+         EXTRN    RQL7EX
                    1431+         CSEG
  009E 0000      E  1435+         DW       RQL7EX
  000C              1436+NEXCH    SET      NEXCH+1
                    1437 ;
                    1438          PUBXCH   CNOX
  00A0 DC06      D  1443+         DW        CNOX
  000D              1444+NEXCH    SET       NEXCH+1
                    1445 ;
                    1446          PUBXCH   FQOLOK
  00A2 F506      D  1451+         DW        FQOLOK
  000E              1452+NEXCH    SET       NEXCH+1
                    1453 ;
                    1454          XCHADR   TICKER
                    1455+         EXTRN    TICKER
                    1456+         CSEG
  00A4 0000      E  1460+         DW       TICKER
  000F              1461+NEXCH    SET      NEXCH+1
                    1462          XCHADR   TCNTRL
                    1463+         EXTRN    TCNTRL
                    1464+         CSEG
  00A6 0000      E  1468+         DW       TCNTRL
  0010              1469+NEXCH    SET      NEXCH+1
                    1470          XCHADR   TRSLTS
                    1471+         EXTRN    TRSLTS
                    1472+         CSEG
  00A8 0000      E  1476+         DW       TRSLTS
  0011              1477+NEXCH    SET      NEXCH+1
                    1478 ;
                    1479 ;*************************************************************************
                    1480 ; Create table:
                    1481          CRTAB
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0        CONFIG   PAGE    6

  LOC  OBJ          SEQ          SOURCE STATEMENT

                    1482+         CSEG
                    1483+         PUBLIC  RQCRTB
                    1484+RQCRTB:
  00AA 0000      C  1485+         DW      ITT
  00AC 08           1486+         DB      NTASK
  00AD 8800      C  1487+         DW      IET
  00AF 11           1488+         DB      NEXCH
                    1489 ;
                    1490 ;*****************************************************************
                    1491 ; Device configuration table:
                    1492 ;
                    1493          DCT     F0,0,0,0        ; unit 0, iSBC-201, controller 0
                    1497+         CSEG
                    1498+         PUBLIC RQDCT
                    1499+RQDCT:
  00B0 46           1530+         DB      CH1,CH2,0,0,0
  00B1 30               +
  00B2 00               +
  00B3 00               +
  00B4 00               +
                    1534          DCT     F1,0,0,1        ; unit 1, ...
  00B5 46           1571+         DB      CH1,CH2,0,0,1
  00B6 31               +
  00B7 00               +
  00B8 00               +
  00B9 01               +
                    1575 ;
                    1576 ;*****************************************************************
                    1577 ; Controller specification table:
                    1578 ;
                    1579          CST     0,78H,2,RQL2EX,CNOX
                    1583+         PUBLIC RQNDEV
  00BA 02           1584+RQNDEV:  DB      NDEV
                    1585+         PUBLIC RQCST
                    1586+RQCST:
  00BB 00           1590+         DB      0,78H,2
  00BC 78               +
  00BD 02               +
  00BE E606      D  1591+         DW      RQL2EX,CNOX
  00C0 DC06      D      +
                    1595 ;
                    1596 ;*****************************************************************
                    1597 ; Buffer allocation block:
                    1598 ;
                    1599          BAB     8,BUFPOL
                    1600+         CSEG
                    1611+         PUBLIC RQBAB
  00C2 0000         1612+RQBAB:   DW      0,0
  00C4 0000             +
  00C6 08           1613+         DB      8
                    1614+         EXTRN   BUFPOL
  00C7 0000      E  1615+         DW      BUFPOL
                    1619 ;
                    1620 ;*****************************************************************
                    1621 ; Public data
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0        CONFIG   PAGE    7


  LOC  OBJ          SEQ          SOURCE STATEMENT

                    1622 ;
                    1623           PUBLIC  RQRATE
  00C9 1C00         1624 RQRATE:  DW      28       ; 2400 baud terminal
                    1625 ;
                    1626           END

PUBLIC SYMBOLS
CNOX    D 06DC    FQOLOK D 06F5    RQBAB  C 00C2    RQCRTB C 00AA    RQCST  C 00BB    RQDCT  C 00B0    RQL0EX D 06E6
RQL2EX D 06E6    RQL3EX D 06E6    RQL4EX D 06E6    RQL5EX D 06E6    RQNDEV C 00BA    RQRATE C 00C9

EXTERNAL SYMBOLS
BUFPOL E 0000    CNOSTK E 0000    ITIMER E 0000    RQALRM E 0000    RQDBUG E 0000    RQDELX E 0000    RQDIRX E 0000
RQDSKX E 0000    RQHD1  E 0000    RQINPX E 0000    RQL6EX E 0000    RQL7EX E 0000    RQOUTX E 0000    RQPDEL E 0000
RQPDIR E 0000    RQPDSK E 0000    RQPRNM E 0000    RQRNMX E 0000    RQTHDI E 0000    RQWAKE E 0000    TCNTRL E 0000
TESTER E 0000    TICKER E 0000    TRSLTS E 0000

USER SYMBOLS
ADDCHR + 0000    BAB    + 0000    BUFPOL E 0000    CH1    A 0046    CH2    A 0031    CNOSTK E 0000    CNOX   D 06DC
CONSTD + 0012    CRTAB  + 0006    CST    + 000D    CTR    A 0002    DCT    + 0010    DNC4   + 001C    FQOLOK D 06F5
GENDNC + 001D    GENTD  + 0007    IET    C 0088    INTXCH + 0009    ITIMER E 0000    ITT    C 0000    LITCHA + 000B
NCONT  A 0001    NDEV   A 0002    NEXCH  A 0011    NTASK  A 0008    PUBXCH + 000A    RQALRM E 0000    RQBAB  C 00C2
RQCRTB C 00AA    RQCST  C 00BB    RQDBUG E 0000    RQDCT  C 00B0    RQDELX E 0000    RQDIRX E 0000    RQDSKX E 0000
RQHD1  E 0000    RQINPX E 0000    RQL0EX D 06E6    RQL2EX D 06E6    RQL3EX D 06E6    RQL4EX D 06E6    RQL5EX D 06E6
RQL6EX E 0000    RQL7EX E 0000    RQNDEV C 00BA    RQOUTX E 0000    RQPDEL E 0000    RQPDIR E 0000    RQPDSK E 0000
RQPRNM E 0000    RQRATE C 00C9    RQRNMX E 0000    RQTHDI E 0000    RQWAKE E 0000    STD    + 0000    TCNTRL E 0000
TDAREA A 00C4    TDBASE D 0618    TESTER E 0000    TICKER E 0000    TRSLTS E 0000    XCH    + 0000    XCHADR + 0000

ASSEMBLY COMPLETE,  NO ERRORS
```

```
asm80 :f2:cammod.asm

ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0         CAMMOD   PAGE    1

  LOC  OBJ          SEQ          SOURCE STATEMENT

                      1          NAME CAMMOD
                      2 ;*************************************************************************
                      3 ; This assembly language module describes a data area which must be in
                      4 ; controller accessible memory.  That is, the data segment of this
                      5 ; module must be located for off-board memory.
                      6 ;*************************************************************************
                      7          DSEG
                      8          PUBLIC CNOSTK,RQDBUF,BUFPOL
                      9 ;
  0050                10 CNOSTK: DS       80       ; stack for controller task
                      11 ;
  02BC                12 RQDBUF: DS       700      ; buffer for DIRSVC directory services
                      13 ;
  0C80                14 BUFPOL: DS       3200     ; buffers for up to 8 open files
                      15 ;
                      16          END

PUBLIC SYMBOLS
BUFPOL D 030C     CNOSTK D 0000     RQDBUF D 0050

EXTERNAL SYMBOLS


USER SYMBOLS
BUFPOL D 030C     CNOSTK D 0000     RQDBUF D 0050

ASSEMBLY COMPLETE,  NO ERRORS
```

```
$MACROFILE
$NOGEN
        NAME CONFIG
;*********************************************************************
; This configuration module describes the hardware environment to
; RMX/80.  The environment assumed includes two disk drives, F0 and
; F1, on one controller CN0 using interrupt level 2 on an iSBC 80/20
; Also included is a terminal driven by the full terminal handler.
;
; The software environment assumed includes the Fortran-80 Run-time
; for RMX/80 Systems which uses the FQOLOK exchange.  Two user
; tasks are also specified, which use three exchanges.
;*********************************************************************
        CSEG
;
; INCLUDE MACROS
;
$nolist
$INCLUDE(:F1:STD.MAC)
$INCLUDE(:F1:XCHADR.MAC)
$INCLUDE(:F1:CRTAB.MAC)
$INCLUDE(:F1:GENTD.MAC)
$INCLUDE(:F1:RMXXCH.MAC)
$INCLUDE(:F1:DFSCFG.MAC)
$list
;
;
NTASK   SET     0
NEXCH   SET     0
NDEV    SET     0
NCONT   SET     0
;
;
;
;*********************************************************************
; Static task descriptors:
;
        STD     RQTHDI,36,112,RQOUTX
terminal handler
;
        STD     RQPDSK,48,129,RQDSKX
disk input/output services
        STD     RQPDEL,64,140,RQDELX
; file delete service
        STD     RQPD1R,48,135,RQDIRX
; disk directory service
        STD     RQPRNM,64,145,RQRNMX
; file rename service
        EXTRN   RQHD1
; controller for disks
        CONSTD  CNTRL1,RQHD1,80,CNOSTK,33,CN0X
;
        STD     ITIMER,300,100,TICKER,18
interval timer user task
        STD     TESTER,1000,200,,18
user main task
        GENTD
;
;*********************************************************************
; Exchanges for disk controller and interrupts
;
        XCH     CN0X
        INTXCH  RQL2EX
        PUBLIC  RQL3EX,RQL4EX,RQL5EX,RQL0EX
RQL3EX  EQU     RQL2EX
RQL4EX  EQU     RQL2EX
RQL5EX  EQU     RQL2EX
RQL0EX  EQU     RQL2EX
;
; Exchange for Fortran i/o system:
        XCH     FQOLOK
;
```

```
;*****************************************************************
; Initial exchange table:
        XCHADR  RQINPX
        XCHADR  RQOUTX
        XCHADR  RQWAKE
        XCHADR  RQDBUG
        XCHADR  RQALRM
;
        XCHADR  RQDSKX
        XCHADR  RQDELX
        XCHADR  RQDIRX
        XCHADR  RQRNMX
;
        PUBXCH  RQL2EX
        XCHADR  RQL6EX
        XCHADR  RQL7EX
;
        PUBXCH  CNOX
;
        PUBXCH  FQOLOK
;
        XCHADR  TICKER
        XCHADR  TCNTRL
        XCHADR  TRSLTS
;
;*****************************************************************
; Create table:
        CRTAB
;
;*****************************************************************
; Device configuration table:
;
        DCT     F0,0,0,0        ; unit 0, iSBC-201, controller 0
        DCT     F1,0,0,1        ; unit 1, ...
;
;*****************************************************************
; Controller specification table:
;
        CST     0,78H,2,RQL2EX,CNOX
;
;*****************************************************************
; Buffer allocation block:
;
        BAB     8,BUFPOL
;
;*****************************************************************
; Public data
;
        PUBLIC  RQRATE
RQRATE: DW      28      ; 2400 baud terminal
;
        END
```

```
        NAME CAMMOD
;*****************************************************************
; This assembly language module describes a data area which must be in
; controller accessible memory.  That is, the data segment of this
; module must be located for off-board memory.
;*****************************************************************
        DSEG
        PUBLIC CNOSTK,RQDBUF,BUFPOL
;
CNOSTK: DS      80      ; stack for controller task
;
RQDBUF: DS      700     ; buffer for DIRSVC directory services
;
BUFPOL: DS      3200    ; buffers for up to 8 open files
;
        END
```

```
;LLTest(source_drive, RMX_library_drive, Fortran_library_drive)

Locate :%0:CAMMod.Obj to :%0:CAMMod.Abs Data(0F000h) Publics Map StackSize(0)
Link :%1:RMX820.Lib(Start), :%0:CAMMod.Abs, :%0:Config.Obj,&
 :%0:Tester.Obj, :%0:TestT.Obj, :%0:ITimer.Obj, :%0:ToSecs.Obj,&
 :%2:F80Run.Lib, :%2:F80RMX.Lib, :%2:FPEF.Lib, :%2:FPSftx.Lib,&
 :%1:THi820.Lib, :%1:THo820.Lib,&
 :%1:DFSDir.Lib(Directory,Seek,Delete,Rename), :%1:Dio820.Lib, :%1:DFSUnr.Lib,&
 :%1:RMX820.Lib, :%1:Unrslv.Lib, :%2:PLM80.Lib &
 to :%0:Tester.Rel
Locate :%0:Tester.Rel to :%0:Tester Print(:%0:Tester.Map) Map Symbols Restart0&
 Code(1000h) StackSize(0)
Delete :%0:Tester.Rel
```

CHAPTER 7
PREPARING PROGRAMS TO RUN WITHOUT ISIS-II OR RMX/80

For some applications, you may wish to run FORTRAN programs in a strictly hardware (iSBC or custom-wired) environment, without either ISIS-II or RMX/80. In this type of run-time environment, there are two parts of your programming task that will differ from programming for other environments: initialization and input/output. This chapter will provide the information you need on these topics, and also the libraries required in the LINK command for this environment.

Initialization

Under ISIS-II or RMX/80, the operating system performs all the required initialization of hardware features and devices such as interrupts and timers. In a stand-alone iSBC or custom-wired system, user-supplied code must perform this initialization. Some of this may have to be done in assembly language and/or PL/M, since FORTRAN does not have the capability to communicate this directly with hardware.

Input and Output

ISIS-II and RMX/80 both provide considerable I/O-handling software facilities. In the absence of these facilities, there are several ways you can perform I/O. One is to use port I/O exclusively; this entails doing your own buffering and formatting, but it may be practical in cases where the data to be handled is relatively simple in form.

A second method, one which allows you to gain some of the advantages of formatted I/O, is to use internal files, which are discussed in Chapter 6 of the *FORTRAN-80 Programming Manual.* However, a formatted READ to or WRITE from an internal file merely formats the data; port I/O, or a separate PL/M procedure or assembly language subroutine, must be used to perform the actual data transfer from or to an external device, making every I/O operation a two-step process.

A third method is to write your own I/O drivers to provide the basic low-level I/O functions—those which are most environment-specific. The standard library routines that provide these capabilities represent only a small part of the I/O system software; by writing your own drivers to replace them and interfacing these with the rest of the I/O software, you can take advantage of the bulk of the I/O system. This method allows you to then code formatted I/O statements to transfer data directly to and from external files.

In order to write your own I/O drivers, you need to know how these drivers communicate with the rest of the I/O system. This information is provided in the following paragraphs. In this discussion, the term "the I/O system" is used to refer to the bulk of the FORTRAN I/O routines, with which your drivers are to interface.

FORTRAN I/O involves more than transferring blocks of data between memory and an external device. The bulk of FORTRAN I/O processing concerns such actions as scanning format strings, converting the internal binary representation of numbers to strings of digits and vice versa, and keeping track of the file referenced in the program. The actual transfer of data between memory and external devices is actually a small part of I/O processing.

The procedures which actually transfer data and communicate with the external devices in other ways (e.g., SEEKing on a diskette file, OPENing a new file) are called *drivers*. The FORTRAN I/O system assumes that each file has eight basic actions, or *capabilities*, that can be performed on it. The mnemonics for these capabilities are OPEN, CLOSE, READ, WRITE, FBACK1, REWIND, MV2REC, and MAKEOF. The OPEN, CLOSE, READ, WRITE, and REWIND capabilities perform the operations required by the FORTRAN statements of the same names. FBACK1 backspaces a sequential-access file one byte. MV2REC positions a direct-access file immediately before the specified relative record in preparation for a READ or WRITE operation. MAKEOF marks the current position as being the end of the file, deleting any part of the file beyond that point. A detailed discussion of each of these capabilities is included later in this chapter.

Associated with each open file is a set of eight routines, one for each of the eight capabilities. These associated routines are supplied dynamically (i.e., under program control) at run time. Both F80ISS.LIB and F80RMX.LIB include drivers that will be associated with each file by default. These drivers are part of their respective libraries. However, it is also possible for the user to supply custom-written drivers that will be used in place of the library-supplied drivers. Such special drivers can be supplied for just one particular file or for a number of different files.

Because the I/O system requires that addresses (procedure entry points) be manipulated as data, you will need to write these drivers in PL/M or assembly language rather than in FORTRAN.

Providing I/O Capabilities for Files

If you wish the FORTRAN I/O system to use special drivers for some or all of your files, you must:

- write a routine called FQ0LVL that recognizes each filename for which you are supplying drivers and gives the entry points of its drivers; and
- write the drivers themselves.

The following sections supply the information you need to do both these steps.

FQ0LVL Procedure

When a unit/file connection is first established by the OPEN statement, the memory record that represents that connection is initialized with the starting address for those routines that implement the eight basic I/O capabilities for that file. In doing this, the I/O system calls the routine named FQ0LVL:

PL/M-80

```
FQ0LVL:
    PROCEDURE(FILENAME$PTR,BUFFER$PTR)
        BYTE PUBLIC;
    DECLARE(FILENAME$PTR,BUFFER$PTR)
        ADDRESS;

    .
    .
    END;
```

ASM80

```
        CSEG
        PUBLIC  FQ0LVL
FQ0LVL:         ;FILENAME ADDR
                ;IS IN BC REGS
                ;BUFFER ADDR IS
                ;IN DE REGS

        .
        .
        END;
```

FILENAME$PTR and BUFFER$PTR are both address values. FQ0LVL returns a byte value. FILENAME$PTR is the starting address of the name of a file. This file name must be a string of 1 to 15 ASCII characters, the last character being a

'%' symbol that serves as a delimiter. BUFFER$PTR is the starting address of a 16-byte buffer that is to become defined with the starting addresses of the routines implementing the eight basic I/O capabilities for this file. If you wish to supply I/O drivers for some or all of your files, you can do so by linking in an FQ0LVL procedure that examines the file name and determines whether this file is one for which you are supplying drivers. If you do wish to supply the drivers for a given file, your FQ0LVL must do the following upon recognizing the filename:

- Copy the starting addresses of the procedures supplying the eight I/O capabilities into the buffer pointed to by BUFFER$PTR. The addresses must be stored in this order: OPEN, CLOSE, READ, WRITE, FBACK1, MV2REC, REWIND, and MAKEOF.
- Return the value 255 to indicate that the drivers have been supplied for this file.

If you do not wish to supply the drivers for this file, FQ0LVL should return the value 0. In this case the system will use the default drivers which are supplied as part of the I/O library.

If you do not specifically link in an FQ0LVL procedure, a default version will be linked in from the I/O library. This default version does nothing but return 0. This causes the default drivers to be used for every file.

I/O Drivers

The user-supplied drivers for the capabilities OPEN, CLOSE, READ, WRITE, FBACK1, MV2REC, REWIND, and MAKEOF must be defined with the calling sequence conventions that are presumed by the I/O system. The calling sequences and general functional descriptions for these operations are given below.

The parameter list, in order, for the OPEN capability is:

```
(STATUS$PTR,OWN$PTR,FILENAME$PTR,MODE,EXISTS)
```

STATUS$PTR is the address of a two-byte variable in which the status of the operation is to be returned upon completion. Allowed status values will be explained later. OWN$PTR is the starting address of an 8-byte block of memory that will always be associated with this file. The address of this block of memory will always be passed to any driver, either default or user-supplied. These memory locations are not manipulated by the I/O system; they are totally devoted to the routines which implement the I/O capabilities for this file. As an example of how this block of memory may be used, in F80ISS.LIB two bytes of this area are used to hold the ISIS-II active file number (AFTN). In F80RMX.LIB, two bytes of this area are used to hold the address of the appropriate exchange for a diskette file. FILENAME$PTR is the starting address of the name of the file to be opened. This name must be a string of from 2 to 15 characters, the last of which must be a '%' delimiter. MODE is a single- byte value of 1, 2, or 3, indicating that the file is to be opened in READ only mode, WRITE only mode, or UPDATE mode, respectively.

EXISTS is a byte with a value of 0, 1, or 2, indicating respectively whether a file with this name may *not* already exist, that it *must* already exist, or that it does not matter. The OPEN driver should initialize a file so that subsequent I/O operations can access the file. The OPEN operation can return a status value of 0, 1, or 2. 0 means that the operation was successful; 1 means that the OPEN was unsuccessful because the EXISTS parameter could not be accommodated—e.g., EXISTS was 0 but there was already a file with this name. A status of 2 means that the EXISTS parameter could be accommodated, but that the OPEN was still unsuccessful.

The CLOSE capability has the following parameter list:

```
(STATUS$PTR,OWN$PTR,DELETE)
```

The STATUS$PTR and OWN$PTR values are the same as for the OPEN capability. DELETE is a byte value which is 255 if the file should be deleted after being closed, and 0 otherwise. The CLOSE driver should undo the initialization actions performed by the OPEN driver, so that I/O operations can no longer access the file. The status value returned is 0 if the operation finished correctly and non-zero otherwise.

The READ capability has the following parameter list:

```
(STATUS$PTR,OWN$PTR, TARGET,LEN)
```

STATUS$PTR and OWN$PTR are the same as for the OPEN capability. TARGET and LEN are address values. TARGET specifies the starting address of a buffer which is LEN bytes long. The READ operation should transfer the next LEN bytes from the file to the buffers. The possible values for the returned status are 0 if the operation finished correctly, −1 if an end-of-file was encountered, and some other value otherwise.

The parameter list for the WRITE capability is as follows:

```
(STATUS$PTR,OWN$PTR ,SOURCE,LEN)
```

STATUS$PTR and OWN$PTR are the same as for the OPEN capability. SOURCE and LEN are address values. SOURCE is the starting address of a buffer which is LEN bytes long. The WRITE driver should transfer these LEN bytes to the file. A returned status of 0 indicates that the WRITE operation was successful; a non-zero status indicates an error.

FBACK1 has the following parameter list:

```
(STATUS$PTR,OWN$PTR)
```

STATUS$PTR and OWN$PTR are the same as for the OPEN capability. This capability must position the file backwards one byte. A returned status of 0 indicates that the operation was successful, a status of −1 indicates that the file was already positioned at the front, and any other non-zero status indicates an error. This capability is used only for backspacing a file.

MV2REC has this parameter list:

```
(STATUS$PTR,OWN$PTR,REL$REC,REC$LEN)
```

STATUS$PTR and OWN$PTR are the same as for the OPEN capability. REL$REC and REC$LEN are address values. The file is assumed to be divided into equal-length records, each having the length given in the value addressed by REC$LEN. The MV2REC driver must position the file immediately before the relative record addressed by REL$REC, where the beginning of the file is relative record 0. The returned status value is 0 for a successful operation and non-zero otherwise. MV2REC is used only for direct-access READ or WRITE operations on a file.

The parameter list for REWIND is as follows:

(STATUSPTR,OWNPTR)

STATUS$PTR and OWN$PTR are the same as for the OPEN capability. This capability must cause the file to be positioned at its beginning. The returned status is 0 for a successful operation and non-zero otherwise. REWIND is used only for implementing a FORTRAN REWIND statement.

MAKEOF has the following parameter list:

(STATUSPTR,OWNPTR)

STATUS$PTR and OWN$PTR are the same as for the OPEN capability. This operation must mark the current position as being the end of the file, with any part of the file after that current position being deleted. The returned status value is 0 for a successful operation and non-zero otherwise.

If you are certain that some capability will never be used for a particular file—e.g., REWIND for a line printer—then you need not supply the corresponding starting address to the I/O system; i.e., that position in the list of starting addresses need not be defined. However, the results may be disastrous if that capability is inadvertently referenced for this file. A better alternative is to supply a null routine, or a routine which signals an error, as the driver for such a (supposedly) unused capability.

The following sample FQ0LVL procedure and set of "stub" drivers, written in PL/M, shows how you might implement custom I/O drivers for a direct-access diskette file called D1. Note that since backspacing, rewinding, and end-of-file capabilities apply only to sequential-access files, no drivers are needed for FBACK1, REWIND, or MAKEOF. As a safeguard against errors, we have included dummy drivers for these capabilities; these "drivers" print an error message and then return.

```
FQ0LVL:  PROCEDURE(FILENAME$PTR,BUFFER$PTR) BYTE PUBLIC;
         DECLARE(FILENAME$PTR,BUFFER$PTR) ADDRESS;
         DECLARE(BUFFER BASED BUFFER$PTR) (8) ADDRESS;
         DECLARE(FILENAME BASED FILENAME$PTR) (15) BYTE;
         IF FILENAME = 'D1' THEN DO;
         BUFFER(0) = .OPND1;
         BUFFER(1) = .CLSD1;
         BUFFER(2) = .RDD1;
         BUFFER(3) = .WRTD1;
         BUFFER(4) = .FBKD1;
         BUFFER(5) = .MVD1;
         BUFFER(6) = .REWD1;
         BUFFER(7) = .EOFD1;
         RETURN(255);
         END;
         /*OPERATIONS TO PROCESS OTHER FILENAMES AND
         TO DETECT UNKNOWN FILENAMES GO HERE*/
             .
             .
         END FQ0LVL;
```

```
OPND1:  PROCEDURE(STATUS$PTR,OWN$PTR,FILENAME$PTR,MODE,EXISTS,);
        DECLARE(STATUS$PTR,OWN$PTR,FILENAME$PTR) ADDRESS;
        DECLARE(MODE,EXISTS)BYTE;
        /* OPERATIONS TO OPEN FILE GO HERE */
            .
            .
        END OPND1;

CLSD1:  PROCEDURE(STATUS$PTR,OWN$PTR,DELETE);
        DECLARE(STATUS$PTR,OWN$PTR) ADDRESS;
        DECLARE DELETE BYTE;
        /* OPERATIONS TO CLOSE FILE GO HERE */
            .
            .
        END CLSD1;

RDD1:   PROCEDURE(STATUS$PTR,OWN$PTR,TARGET,LEN);
        DECLARE(STATUS$PTR,OWN$PTR,TARGET,LEN) ADDRESS;
        /* OPERATIONS TO READ FROM FILE GO HERE */
            .
            .
        END RDD1;

WRTD1:  PROCEDURE(STATUS$PTR,OWN$PTR,SOURCE,LEN);
        DECLARE(STATUS$PTR,OWN$PTR,SOURCE,LEN) ADDRESS;
        /* OPERATIONS TO WRITE TO FILE GO HERE */
            .
            .
        END WRTD1;

FBKD1:  PROCEDURE(STATUS$PTR,OWN$PTR);
        DECLARE(STATUS$PTR,OWN$PTR) ADDRESS;
        /* STATEMENTS TO PRINT ERROR MESSAGE GO HERE */
            .
            .
        END FBKD1;

MVD1:   PROCEDURE(STATUS$PTR,OWN$PTR,REL$REC,REC$LEN);
        DECLARE(STATUS$PTR,OWN$PTR,REL$REC,REC$LEN) ADDRESS;
        /* OPERATIONS TO POSITION FILE BEFORE SPECIFIED RECORD GO
        HERE */
            .
            .
        END MVD1;

REWD1:  PROCEDURE(STATUS$PTR,OWN$PTR);
        DECLARE(STATUS$PTR,OWN$PTR) ADDRESS;
        /* STATEMENTS TO PRINT ERROR MESSAGE GO HERE */
            .
            .
        END REWD1;

EOFD1:  PROCEDURE(STATUS$PTR,OWN$PTR);
        DECLARE(STATUS$PTR,OWN$PTR) ADDRESS;
        /* STATEMENTS TO PRINT ERROR MESSAGE GO HERE */
            .
            .
        END EOFD1;
```

If you do not wish to use the library supplied default file capabilities (i.e., those in F80ISS.LIB or F80RMX.LIB) for *any* of your files, you can avoid linking in the module that contains these capabilities. Since this is a sizeable module—i.e., greater than 4K bytes—you will probably wish to do so. To avoid linking in this module, you must code a routine called FQ0DL0, which must return a value of zero in the A register, and cause it to be unconditionally linked in before F80ISS.LIB or F80RMX.LIB is linked. (If you include this module in the same library file with the rest of your code, the LINK command order of libraries prescribed at the end of this chapter will cause it to be linked in correctly.) The effect of these actions is to make the I/O library (F80ISS.LIB or F80RMX.LIB) recognize only those file names for which you have explicitly supplied device drives—i.e., no default device drivers are recognized.

FQ0DL0 will never be called so long as you supply I/O drivers for all files in your system.

Directly Callable I/O Drivers

The purpose of separating out the drivers from the rest of the I/O libraries is to make the libraries as environment-independent as possible. There are other functions of FORTRAN I/O, besides communication with user files, that depend heavily on the run-time environment of the program. These functions include processing a STOP statement, processing a PAUSE statement, handling fatal errors, and specifying preconnections. Each of these functions is performed by a particular publicly defined procedure. If the procedure included in the library is unsuited to the environment in which your program must run, it is a simple matter to write an environment-specific driver which correctly performs the function of any one of these procedures. The following paragraphs provide a more detailed explanation of these four routines.

FQ0007 is the routine for processing STOP statements. A user routine could be written as follows:

PL/M-80

```
FQ0007:
        PROCEDURE(MSG$PTR,MSG$LEN)PUBLIC;
        DECLARE(MSG$PTR,MSG$LEN)ADDRESS;

        .
        .
        END FQ0007;
```

ASM80

```
        CSEG
        PUBLIC  FQ0007
FQ0007:         ;MESSAGE ADDR
                ;IS IN BC REGS
                ;MESSAGE LENGTH
                ;IS IN DE
        .
        .
        END
```

Both parameters are address values. The first parameter is a pointer to the beginning of a character string; the second parameter is the length of that character string. This string is the message which was in the STOP statement. Control must not return to the user program from this routine.

FQ0008 is the routine for processing PAUSE statements. A user routine could be written as follows:

```
          PL/M-80                                ASM80

FQ0008:                                          CSEG
       PROCEDURE(MSG$PTR,MSG$LEN)PUBLIC;         PUBLIC   FQ0007
       DECLARE(MSG$PTR,MSG$LEN)ADDRESS;  FQ0008:          ;MESSAGE ADDR IS IN BC
                                                          ;REGS
                                                          ;MESSAGE LENGTH IS IN DE
       .                                          .
       .                                          .
       END FQ0008;                                END
```

where the parameters are the same as for FQ0007. It *is* legitimate for control to return to the user program from FQ0008.

FQ0FER is the routine for handling fatal errors. A user routine could be written as follows:

```
          PL/M-80                                ASM80

FQ0FER:                                          CSEG
       PROCEDURE(ERRNUM,CALLEDFROM)PUBLIC;       PUBLIC   FQ0FER
       DECLARE(ERRNUM,CALLEDFROM)ADDRESS;                 ;ERRNUM IS IN BC REGS
                                                          ;CALLEDFROM IS IN DE
       .                                          .
       .                                          .
       END FQ0FER;                                END
```

ERRNUM is a byte value which is the FORTRAN run-time error number (as listed in Appendix B). CALLEDFROM is an address value which is an address near the code for that FORTRAN I/O statement which is in error. Control must *not* return to the calling program from FQ0FER.

FQ0PRC is a routine which specifies unit/file pairs which are to be preconnected. A user routine could be written as follows:

```
          PL/M-80                                ASM80

FQ0PRC:                                          CSEG
       PROCEDURE(BUFFER$PTR)PUBLIC;              PUBLIC   FQ0PRC
       DECLARE BUFFER$PTR ADDRESS;       FQ0FRC:          ;BUFFER ADDR IS IN BC REGS
       .                                          .
       .                                          .
       END FQ0PRC;                                END
```

BUFFER$PTR is the starting address of an eight-element array of structures. Each structure in the array has a two-byte UNIT field and a 15-byte FILENAME field. The PL/M declaration of this structure is:

```
DECLARE BUFFER (8) STRUCTURE (
    UNIT ADDRESS,
    FILENAME (15) BYTE);
```

FQ0PRC should fill in those unit/filename pairs that are to be preconnected. The first entry whose UNIT field is 65,535 indicates that this entry and all later entries are to be ignored; i.e., they are not filled in. If there is a conflict between the preconnections specified (two or more different entries have identical UNIT or FILENAME fields), then only the last of the conflicting specifications will be used. (The "last" entry is that entry with the greatest index in BUFFER.)

LINK Command

Remember that even when you are not using ISIS-II or RMX/80 at run time, you must first link your program segments together and locate them on an Intellec or Intellec Series II System using ISIS-II, which provides the LINK and LOCATE programs.

If you are running your FORTRAN programs in a non-ISIS, non-RMX environment, your 'input-list' to the LINK command must still include F80ISS.LIB if you perform any FORTRAN I/O statements (other than port I/O), since this library contains I/O routines with which your drivers must interface. For a non-ISIS, non-RMX system, three possibilities exist:

- Your program uses FORTRAN I/O statements on external files, and/or STOP or PAUSE statements. In this case, you include your own drivers in the library with the rest of your code, then link in F80ISS.LIB as the I/O library (between F80RUN.LIB and FPEF.LIB in the LINK command).
- Your program uses FORTRAN I/O statements, but only to internal files, and no STOP or PAUSE statements. In this case, you need supply no drivers of your own, but should link in *both* F80NIO.LIB and F80ISS.LIB (F80NIO.LIB first) between F80RUN.LIB and FPEF.LIB in the LINK command.
- Your program uses no FORTRAN I/O statements (except port I/O) and no STOP or PAUSE statements. In this case, you simply link in F80NIO.LIB between F80RUN.LIB and FPEF.LIB in the LINK command.

To supply floating-point arithmetic routines, link in FPSOFT.LIB to use the software floating-point operations, FPHARD.LIB to use the iSBC 310 interface, or FPNULL.LIB if no floating-point operations are needed. For the order in which all libraries must be specified to the LINK command, refer to Chapter 4 or Appendix B.

Example:

```
-LINK :F1:FPROG.OBJ,F80RUN.LIB,F80ISS.LIB,FPEF.LIB,&
**FPSOFT.LIB,PLM80.LIB TO FPROG.LNK MAP
```

(Note: The double asterisks are prompts issued by the LINK command.)

This LINK command is identical to the example given in Chapter 4. For the non-ISIS, non-RMX environment, you include your own I/O driver routines in FPROG.OBJ, and the LINK command will automatically substitute these for the corresponding default routines in F80ISS.LIB.

APPENDIX A
THE COMPILER AND THE FORTRAN LANGUAGE

The language translated by the FORTRAN-80 compiler includes the ANSI FORTRAN 77 subset, as defined in the *FORTRAN-80 Programming Manual.* In the programming manual, several aspects of the language were said to be 'processor dependent' or 'compiler dependent.' This chapter summarizes the limitations and extensions to the FORTRAN language assumed by the FORTRAN-80 compiler.

Compiler Limitations On Language

Most constraints imposed on the FORTRAN language are related to data lengths and the permissible range of data values. The following indicates the range of values possible for a given variable or array element length.

Length	Value Range
INTEGER*1	−128 to +127
INTEGER*2	−32,768 to +32,767
INTEGER*4	−32,768 to +32,767
LOGICAL*1	.TRUE. or .FALSE.
LOGICAL*2	.TRUE. or .FALSE.
LOGICAL*4	.TRUE. or .FALSE.
REAL	Approximately −3.37E+38 to +3.37E+38 (The handling of magnitudes less than 1.17E-38 is not defined.)

If no length is specified, the compiler defaults are INTEGER*2 and LOGICAL*1.

The maximum field width, '*w*,' in the *Fw.d, Ew.d, Iw,* and *Lw* edit descriptors of the FORMAT statement is 32,767. The maximum length of the format string, 'flist',in a FORMAT statement is 255 characters.

The length and interpretation of integer expression values is determined as follows:

- Addition, subtraction, multiplication, division, or exponentiation is performed modulo 256 for two INTEGER*1 operands and modulo 65536 otherwise.
- Assignment is performed modulo 256 if the variable whose value is being assigned has type INTEGER*1 and modulo 65536 otherwise.
- The length of the value of integer expressions used as actual arguments (but which are not variables or array elements) is at least the default length of an integer variable.
- Subscript expression values are taken modulo 65536.

In all of the cases listed above, overflow is ignored.

Statement Functions

The FORTRAN-80 compiler does not support statement functions. Functions must be coded as FUNCTION subprograms.

Compiler Extensions To Language

The FORTRAN compiler provides a number of features that extend the capabilities of the FORTRAN language.

Lowercase Letters

Except within Hollerith and character constants, a lowercase letter is considered to be identical to its corresponding uppercase letter.

Record Length Specifier for Sequential Access Files

The compiler provides an additional feature for input from sequential access files. If the record length specifier is given in an OPEN statement for a *sequential* access file, whenever that file is READ the input line is extended with blanks as necessary to provide the specified record length. This means, for instance, that if the record length is specified as 80 but the line read in has fewer than 80 characters, the line will be extended with blanks to make it 80 characters long, and no error will be registered. This feature applies to input only; output to such a sequential file is not affected.

Port Input/Output

The compiler provides two intrinsic subroutines for handling input/output through 8080/8085 I/O ports. When these subroutines are called, they generate 8080 IN and OUT instructions.

The form of the subroutine call is

```
CALL INPUT(port, var)
CALL OUTPUT(port, exp)
```

where

port	is an integer constant in the range $0 \leqslant port \leqslant 255$
var	is an integer variable
exp	is an integer expression

The value read or written through the specified port is always a single-byte integer (INTEGER*1).

Examples:

```
CALL INPUT(1, TEST1)
CALL OUTPUT(2, 100)
```

Reentrant Procedures

External procedures can be defined to be reentrant by setting the REENTRANT compiler control (Chapter 2). When this control is used, local variables are allocated dynamically on the stack when the procedure is entered, rather than being statically allocated.

Freeform Line Format

Normally, FORTRAN source file lines must be in the standard format. If the FREEFORM compiler control (Chapter 2) is set, however, the following rules apply:

- If a statement has a label, the label must begin as the first character of the statement's initial line.
- The first character of a continuation line must be an ampersand (&).
- Control lines must have a dollar sign ($) as their first character.
- Statements can begin in column 2, or in column 1 if the first character is not 'C.'

Comment lines are the same in both formats. The first character must be a 'C' or an asterisk (*).

The freeform line format simplifies entering FORTRAN programs through a console terminal.

Interpretation Of DO Statements

The 1966 ANSI FORTRAN standard implies that all DO loops must be executed at least once when encountered during program execution. The 1977 ANSI standard allows zero iterations, if so specified by the values of the initial and terminal expressions in the DO statement format. You can select the interpretation you prefer by specifying either the DO66 compiler control or the DO77 compiler control (Chapter 2).

Including Source Files

You can 'include' specified files in your source file using the INCLUDE compiler control (Chapter 2). This control causes subsequent source code to be input from the specified 'file' until an end-of-file is reached. At end-of-file, input resumes from the file being processed when the INCLUDE was encountered.

Flexibility In Standard Restrictions

The ANSI FORTRAN 77 standard prohibits certain constructions that cannot be checked (or are not economical to check) by the compiler, or that cannot be implemented by other processors. Although the FORTRAN-80 compiler generally follows the standard in prohibiting these constructions, it does allow certain meaningful constructions even though they are nonstandard. While this affords the programmer some additional flexibility, be aware that future compilers may implement checks in these areas.

Association Of Storage Units

Character, logical, and numerical items can be freely declared within the same common block and can be equivalenced. In particular, the compiler does not check whether character variables of different lengths are associated.

Partially Initialized Arrays

The DATA statement can be used to initialize arrays *partially* (starting at the first element). If the 'nlist' in the DATA statement format contains several unsubscripted array names, initialization begins with the first array and continues until all items in 'clist' have been used.

Transfers Into IF Blocks

The compiler does not check the formal restriction against transfers into an IF, ELSE IF, or ELSE block.

Unit Preconnection

The UNIT run-time control (Chapter 4) is used to preconnect units to your program so they need not be connected by the OPEN statement. Two units, the console input and output devices, are preconnected automatically to unit numbers 5 and 6, respectively.

Interrupt Processing

Interrupt processing for the 8080 and 8085 is not supported by FORTRAN. To process interrupts, you must write separate assembly language or PL/M drivers and call your FORTRAN program as a subroutine.

In the ISIS-II run-time environment, no FORTRAN subprogram that uses floating-point (REAL) arithmetic operatons (or intrinsic functions involving REAL numbers) may interrupt another FORTRAN subprogram that also uses floating-point operations. Users who need to do this should consider operating in the RMX/80 environment, where fuller interrupt capabilities are supported.

APPENDIX B
ERROR MESSAGES

This appendix lists all error messages produced by the FORTRAN-80 compiler as well as run-time errors encountered when your program is executed. Error messages and codes issued by ISIS-II, RMX/80, LINK, and LOCATE are also summarized here for your convenience. ISIS-II, LINK, and LOCATE errors are explained in detail in the *ISIS-II User's Guide*; RMX/80 errors, in the *RMX/80 User's Guide.*

FORTRAN Compiler Error Messages

The compiler may issue five kinds of error messages.

- FORTRAN source program errors
- Compiler control errors
- Input/output errors
- Insufficient memory errors
- Compiler failure errors

Source Program Error Messages

Source program errors are not fatal. The error messages are interspersed in the program listing at the points of error. They are listed in the format:

***ERROR *m*, STATEMENT *n*, NEAR *symbol, message*

where

m	is the error number from the list below
n	is the sequential number of the statement where the error occurred
symbol	is the source text near the point of error
message	is the error explanation from the list below

Source program error totals are summarized at the end of the program listing for each program unit as shown in Figure 3-3.

NOTE

Some error numbers —e.g., 126 through 149— do not appear in this list. No errors corresponding to these numbers exist, and they should never appear in an error message.

Error No.	Message
1	FIRST LINE OF A STATEMENT IS A CONTINUATION LINE
2	TOO MANY CONTINUATION LINES
3	END OF FILE ENCOUNTERED ON SOURCE INPUT
4	A LINE IN THIS STATEMENT IS TOO LONG
5	NON-DIGIT IN STATEMENT NUMBER FIELD

Error No.	Message
6	TOO MANY NESTED INCLUDES
7	SYNTAX PRECLUDES STATEMENT CLASSIFICATION
8	BLANK STATEMENT
9	STATEMENT HAS UNBALANCED PARENS
10	STATEMENT CONTAINS UNCLOSED CHARACTER CONSTANT
11	STATEMENT IS OF UNKNOWN TYPE
12	INVALID EXPLICIT LENGTH
13	"END" STATEMENT MISSING—CANNOT PROCESS NEXT LINE
14	STATEMENT IS OUT OF PLACE
15	STATEMENT IS INCORRECTLY NESTED WITH RESPECT TO A "DO"
16	STATEMENT CANNOT END A "DO"
17	"DO" OR "IF" BLOCK NOT CLOSED AT END OF PROGRAM UNIT
18	STATEMENT NOT PROCESSED AS DO'S OR BLOCK-IF'S ARE TOO DEEPLY NESTED
19	"THEN" CLAUSE MISSING
20	TOO MANY LABELS IN LABEL LIST
21	THREE LABELS MUST FOLLOW ARITHMETIC-IF
22	NO PRIOR "IF"
23	A DO-LOOP IS STILL OPEN
24	MAY NOT FOLLOW AN "ELSE"
25	MISSING PROGRAM UNIT NAME
26	PROGRAM UNIT NAME LONGER THAN SIX CHARACTERS
27	COMMON BLOCK NAME MUST BE AN IDENTIFIER
28	VARIABLE IS ALREADY IN A COMMON BLOCK
29	CANNOT PUT A PARAMETER INTO COMMON
30	DIMENSION LIST REQUIRED
31	ALREADY DECLARED "EXTERNAL"
32	PREVIOUSLY DECLARED IN ANOTHER CONTEXT
33	PARAMETER LIST REQUIRED
34	MISSING TYPE SPECIFICATION
35	INCORRECT LETTER RANGE
36	DUPLICATE IMPLICIT
37	SINGLE LETTER EXPECTED
38	NOT AN INTRINSIC FUNCTION
39	ALREADY ASSIGNED A MODE
40	MORE THAN SEVEN DIMENSIONS
41	ALREADY DIMENSIONED
42	DUPLICATE PARAMETERS
43	IDENTIFIER EXPECTED
44	NON-ZERO, UNSIGNED, INTEGER CONSTANT LESS THAN 32,768 REQUIRED
45	VARIABLE NAME REQUIRED
46	ILLEGAL PUNCTUATION
47	NUMERIC CONSTANT REQUIRED AFTER "+" OR "–"
48	CONSTANT EXPECTED IN VALUE LIST
49	"TO" MISSING
50	INTEGER SCALAR VARIABLE REQUIRED
51	LABEL REQUIRED
52	LABEL MUST NOT BE LARGER THAN FIVE DIGITS
53	LABEL MUST NOT BE ZERO
54	FORMAT STATEMENTS MUST BE LABELLED
55	DUPLICATE LABEL
56	FORMAT STATEMENT BODY MUST BEGIN WITH "("
57	FORMAT STATEMENT IS TOO LONG
58	STATEMENT NOT ALLOWED IN BLOCK DATA PROGRAM UNIT

Error No.	Message
59	STATEMENT NOT ALLOWED AS PART OF LOGICAL-IF
60	SYNTAX ERROR—PROCESSING TERMINATED BEFORE END OF STATEMENT
61	ASSIGNMENT STATEMENT DOES NOT START WITH AN IDENTIFIER
62	TERMINAL LABEL OF "DO" ALREADY DEFINED
63	ILLEGAL PAUSE OR STOP VALUE
64	EXPECTED KEYWORD MISSING IN I/O CONTROL LIST
65	UNKNOWN KEYWORD IN I/O CONTROL LIST
66	ILLEGAL KEYWORD IN I/O CONTROL LIST
67	REQUIRED KEYWORD MISSING IN I/O CONTROL LIST
68	SUBSCRIPTING A NON-ARRAY
69	FUNCTION USED IN A CONTEXT WHICH REQUIRES A SUBROUTINE
70	ILLEGAL USE OF RELATIONAL OPERATOR
71	OPERAND EXPECTED
72	SUBROUTINE USED IN A CONTEXT WHICH REQUIRES A FUNCTION
73	ILLEGAL AS A FUNCTION
74	INCORRECT NUMBER OF ARGUMENTS
75	PROCEDURE USED WITHOUT ARGUMENTS
76	SUBSCRIPT EXPRESSION MUST BE INTEGER
77	INCORRECT NUMBER OF SUBSCRIPTS
78	INCORRECT MODE FOR ARGUMENT OF AN INTRINSIC
79	EXPRESSION IS TOO COMPLEX
80	ILLEGAL MIXED MODE EXPRESSION
81	CALL STATEMENT MUST BEGIN WITH AN IDENTIFIER
82	PROCEDURE NAME REQUIRED IN THIS CONTEXT
83	MISSING OR ILLEGAL ARGUMENT LIST FOR "INPUT" OR "OUTPUT"
84	INVALID LEFTHAND SIDE OF AN ASSIGNMENT
85	EQUAL SIGN MISSING WHERE EXPECTED
86	ARRAY NAME USED WITHOUT SUBSCRIPTS
87	INTEGER EXPRESSION REQUIRED
88	LOGICAL EXPRESSION REQUIRED
89	STATEMENT IS TOO COMPLEX TO PROCESS
90	TOO MANY IDENTIFIERS
91	ASSUMED-SIZE ARRAY CANNOT BE TRANSMITTED IN AN I/O STATEMENT
92	TOO MANY EXTERNALS
93	INSUFFICIENT SPACE TO DISPLAY ALL ERROR MESSAGES
94	FORMAT STATEMENT MUST END WITH ")"
95	ILLEGAL EXPONENT FORMAT
96	ZERO COUNT FOR HOLLERITH CONSTANT IS ILLEGAL
97	END OF STATEMENT INSIDE HOLLERITH CONSTANT
98	NON-FORTRAN CHARACTER
99	UNPRINTABLE ASCII CHARACTER
100	END OF STATEMENT INSIDE CHARACTER CONSTANT
101	ILLEGAL USE OF "."
102	NON-ALPHANUMERIC INSIDE PRESUMED KEYWORD
103	END OF STATEMENT IN PRESUMED KEYWORD
104	IDENTIFIER, CONSTANT, OR KEYWORD LONGER THAN 255 CHARACTERS
105	ILLEGAL MACHINE-BASED CONSTANT
106	STATEMENT NOT YET IMPLEMENTED
107	UNKNOWN KEYWORD

Error No.	Message
108	DUPLICATE STATEMENT NUMBER
109	TOO MANY COMMON BLOCKS
110	QUOTED STRING REQUIRED IN CONTROL
111	CONTROL LINE ENDS INSIDE QUOTED STRING
112	QUOTED STRING IS TOO LONG
113	NON-DECIMAL DIGIT IN CONTROL VALUE
114	INCORRECT DEVICE FOR INCLUDE CONTROL
115	FILE NAME IS TOO LONG
116	ILLEGAL VALUE FOR OPTIMIZE CONTROL
117	ILLEGAL VALUE FOR PAGELENGTH OR PAGEWIDTH VALUE
118	ILLEGAL "STORAGE" SPECIFICATION
119	UNKNOWN CONTROL
120	PRIMARY CONTROLS MUST OCCUR BEFORE FIRST NON-CONTROL LINE
121	PRIMARY CONTROLS CANNOT BE CHANGED ONCE SET
122	TARGET OF "ASSIGN" MUST NOT BE INTEGER*1
123	ILLEGAL ITEM IN I/O LIST
124	ILLEGAL USE OF A PROCEDURE NAME
125	QUOTED STRING MAY NOT BE NULL
150	NUMBER OF ARRAY ELEMENTS EXCEEDS 65,535
151	SIZE OF AN ARRAY EXCEEDS 65,535 BYTES
152	PROGRAM UNIT USES TOO MANY EXTERNAL PROCEDURES
153	PROGRAM UNIT HAS TOO MANY COMMON BLOCKS
154	ATTEMPT TO EXTEND A COMMON BLOCK ON THE LEFT
155	ATTEMPT TO EQUIVALENCE A NAME TO TWO DIFFERENT LOCATIONS
156	ATTEMPT TO EQUIVALENCE NAMES IN DIFFERENT COMMON BLOCKS
157	AN EQUIVALENCE LIST MUST HAVE AT LEAST TWO VALID ENTRIES
158	CANNOT USE A DUMMY ARGUMENT IN AN EQUIVALENCE LIST
159	IMPROPER SUBSCRIPT VALUE
160	RUN-TIME STORAGE OVERFLOW
161	THIS STATEMENT CANNOT BE REACHED
162	THE FORMAT STRING DOES NOT LOGICALLY END BEFORE THE PHYSICAL END OF THE STRING
163	A REPETITION FACTOR (BEFORE A REPEATABLE EDIT DESCRIPTOR OR BEFORE A PARENTHESIZED FORMAT SUB-STRING) IS ZERO
164	THE FIRST NON-BLANK CHARACTER OF A FORMAT STRING IS NOT '('
166	THIS FORMAT STRING CONTAINS NESTED FORMAT SUBSTRINGS WHICH ARE NESTED MORE THAN THREE DEEP
167	A SIGN ('+' OR '−') IS NOT FOLLOWED BY A STRING OF ONE OR MORE DIGITS IN THIS FORMAT STRING
168	THIS FORMAT STRING CONTAINS A SIGNED NUMBER WHICH IS NOT IMMEDIATELY FOLLOWED BY A SCALE FACTOR EDIT DESCRIPTOR ('P')
172	AN EDIT DESCRIPTOR IN THIS FORMAT STRING IS NOT IMMEDIATELY FOLLOWED BY EITHER ',' OR '/' OR ')'
173	IN THIS FORMAT STRING, A FIELD WIDTH SPECIFIER (W = UNSIGNED, POSITIVE INTEGER) DOES NOT IMMEDIATELY FOLLOW AN IW, LW , BW, ZW, FW.D, EW.D OR EW.DEE REPEATABLE EDIT DESCRIPTOR
175	IN THIS FORMAT STRING, A PERIOD ('.') DOES NOT IMMEDIATELY FOLLOW THE FIELD WIDTH SPECIFIER (W) IN A FW.D, EW.D, OR EW.DEE REPEATABLE EDIT DESCRIPTOR

Error No.	Message
176	IN THIS FORMAT STRING, A DECIMAL FRACTION WIDTH SPECIFIER (D = UNSIGNED, POSITIVE INTEGER) DOES NOT IMMEDIATELY FOLLOW THE PERIOD ('.') IN A FW.D, EW.D OR EW.DEE REPEATABLE EDIT DESCRIPTOR
177	IN THIS FORMAT STRING, A REPEATABLE EDIT DESCRIPTOR WAS NOT FOUND WHERE ONE WAS EXPECTED
179	IN THIS FORMAT STRING, AN EXPONENT FIELD WIDTH SPECIFIER (UNSIGNED, POSITIVE INTEGER) DOES NOT IMMEDIATELY FOLLOW THE 'E' IN AN EW.DEE REPEATABLE EDIT DESCRIPTOR
180	IN THIS FORMAT STRING, THE DECIMAL FRACTION WIDTH SPECIFIER (D) IS GREATER THAN THE FIELD WIDTH SPECIFIER (W) FOR AN FW.D, EW.D OR EW.DEE REPEATABLE EDIT DESCRIPTOR
181	IN THIS FORMAT STRING, THE FIELD WIDTH SPECIFIER (W) IS NOT LARGE ENOUGH FOR THE SPECIFIED DECIMAL FRACTION SUB-FIELD WIDTH (D) AND THE EXPONENT SUB-FIELD (E) IN AN EW.DEE REPEATABLE EDIT DESCRIPTOR
182	THE FORMAT STRING ENDS LOGICALLY BEFORE THE LAST NONBLANK CHARACTER
193	INTEGER VARIABLE REQUIRED AS I/O KEYWORD ARGUMENT
194	CHARACTER EXPRESSION REQUIRED AS I/O KEYWORD ARGUMENT
195	INTEGER EXPRESSION REQUIRED AS I/O KEYWORD ARGUMENT
196	NOT A FORMAT STATEMENT NUMBER
197	ILLEGAL FORMAT SPECIFICATION
198	ILLEGAL UNIT SPECIFICATION
199	ILLEGAL COMBINATION OF I/O KEYWORDS
200	STATEMENT TOO LONG FOR BUFFER
201	THE NATURE OF OTHER ERRORS PROHIBITS OBJECT PRODUCTION FOR THIS PROGGRAM UNIT
203	TOO MANY NESTED CALLS
206	UNDEFINED LABEL
209	A MAIN PROGRAM MAY NOT BE DECLARED REENTRANT
210	CONSTANT TOO LARGE
211	THE CODE SEGMENT LENGTH EXCEEDS 65,535 BYTES
220	SUBSCRIPT ON A NON-ARRAY IN A DATA STATEMENT
221	INCORRECT NUMBER OF SUBSCRIPTS FOR AN ARRAY IN A DATA STATEMENT
222	SUBSCRIPT LARGER THAN DIMENSION OF ARRAY IN A DATA STATEMENT
223	ATTEMPT TO INITIALIZE A LOCAL VARIABLE IN A REENTRANT PROGRAM UNIT
224	ATTEMPT TO INITIALIZE A LOCAL REENTRANT VARIABLE IN A DATA STATEMENT
225	NUMBER OF INITIAL VALUES IN A DATA STATEMENT EXCEEDS NUMBER OF DESTINATIONS
226	TYPE OF VARIABLE DOES NOT MATCH INITIAL VALUE IN A DATA STATEMENT
227	CHARACTER STRING LENGTH LONGER THAN SIZE OF DESTINATION
228	REAL NUMBER OUT OF RANGE
229	TOO MANY NESTED SAVE CONTROLS
230	NO ACTIVE SAVE CONTROL FOR THIS RESTORE CONTROL
246	INTERNAL UNIT CANNOT BE USED
247	ILLEGAL USE OF HOLLERITH CONSTANT
248	SYMBOL IS LONGER THAN 6 CHARACTERS

Error No.	Message
249	AN END STATEMENT MAY NOT APPEAR ON AN INCLUDE FILE
250	ADJUSTABLE DIMENSION MUST BE INTEGER SCALAR DUMMY ARGUMENT OR IN COMMON
251	ADJUSTABLE ARRAY MUST BE A DUMMY ARGUMENT
252	INTEGER OR REAL EXPRESSION REQUIRED
253	I/O DO'S ARE TOO DEEPLY NESTED
254	APPARENT END OF I/O LIST INSIDE I/O DO

Compiler Control Error Messages

If an error is detected in a compiler control (whether in a control line or in the command tail of the compiler invocation), the compilation may be terminated and an error message is issued to the console and list file. The form of the message is

***FORTRAN COMPILATION TERMINATED. *message*

where 'message' is one of the following:

ILLEGAL COMMAND TAIL SYNTAX
ILLEGAL OR INCORRECT OPTION IN COMMAND TAIL
INCORRECT DEVICE SPEC
INVOCATION COMMAND DOES NOT END WITH <CR><LF>
SOURCE FILE EXTENSION INCORRECT
SOURCE FILE NAME INCORRECT
SOURCE FILE NOT A DISKETTE FILE
WORKFILES ALREADY OPEN

Input/Output Error Messages

Fatal input/output errors occur if you should incorrectly specify a file or device name for compiler input or output. The error messages issued are:

ATTEMPT TO OPEN AN ALREADY OPEN FILE
ATTEMPT TO READ PAST EOF
DEVICE TYPE NOT COMPATIBLE WITH INTENDED USE
FILE IS NOT ON A DISKETTE
FILE IS WRITE PROTECTED
FILENAME REQUIRED ON A DISKETTE FILE
ILLEGAL FILENAME SPECIFICATION
ILLEGAL OR UNRECOGNIZED DEVICE SPECIFICATION IN FILENAME
NO SUCH FILE
NULL FILE EXTENSION

Insufficient Memory Error Messages

A fatal error occurs if the system configuration does not have enough RAM memory to support the compiler. The error messages issued in this case are as follows:

DYNAMIC STORAGE OVERFLOW
NOT ENOUGH MEMORY FOR COMPILATION

Compiler Failure Errors

Fatal compiler failure errors are internal errors that should never occur. If you encounter one of these errors, please report it to Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051, Attention: Software Marketing Department

The two errors falling into this category are:

```
208  COMPILER ERROR:  AN OPERAND HAS A DISALLOWED FORM
214  COMPILER ERROR:  SOME OPERATOR CAN'T GET ITS
                      OPERANDS INTO AN ACCEPTABLE FORM
```

FORTRAN Run-Time Error Messages

Certain run-time errors like floating-point overflow, improper format specification, and inappropriate I/O operations invoke error routines. Unless specified otherwise by the program, the error routines are library routines that output an error message to the console and return control to ISIS. The form of such a message is

```
***EXECUTION ERROR. message
```

The asterisks are separated by ASCII BELL characters.

Run-Time Arithmetic Errors

Errors in operations and intrinsic functions involving floating-point (REAL) numbers are normally handled by a PUBLIC error-handling routine named FQFERH. Two error handlers by that name are provided in the FORTRAN run-time libraries: a default error handler included in all the non-null arithmetic libraries (FPSOFT.LIB, FPHARD.LIB, FPSFTX.LIB, FPHRDX.LIB, and FPHX10.LIB), and an alternate error handler in F80ISS.LIB. You can use the default error handler, select the alternate routine by linking it in explicitly, or provide your own error handler if you wish.

The arithmetic libraries recognize the following error codes:

1	Attempted division by zero
2	Argument range exceeded
3	Overflow (value too large to represent internally)
4	Underflow (value too small to represent internally)
5	First argument is invalid
6	Second argument is invalid

The following codes are used to denote the operations and intrinsic functions that use floating-point (REAL) numbers:

1	Addition
2	Subtraction
3	Multiplication
4	Division
5	Conversion of real to 32-bit integer
6	Comparison of two values
7	Comparison of a single value to zero

8	Negation (sign complementation)
9	Absolute value
10	Square root
11	Conversion of real to 16-bit integer
101	SIGN
102	DIM
103	AINT
104	ANINT
105	NINT
106	AMOD
108	EXP
109	ALOG
110	ALOG10
111	SIN
112	COS
113	TAN
114	ASIN
115	ACOS
116	ATAN
117	ATAN2
118	SINH
119	COSH
120	TANH

Operations 101-120 are intrinsic functions from the list in Appendix B of the *FORTRAN-80 Programming Manual.* The other operations are floating-point library routines.

Default Error Handler

The default error handler will attempt to recover from an error and continue. If 'arg1' and 'arg2' represent the first and second arguments, respectively, the recovery action of the default error handler can be represented as follows:

On attempted division by zero (error 1), arg1:= indefinite.

On argument range exceeded (error 2), arg1:= indefinite.

On overflow (error 3) for a *real* number, argl:=sign (argl) real MAX.

On overflow (error 3) for an *integer,* arg1:= sign (arg1) integer MAX .

On underflow (error 4), argl:=0.

On invalid operand(s) (error 5 and/or 6), no action will be taken.

Upon return from the error handler, 'arg1' contains a (possibly new) REAL result for any operation other than 5 or 11, and 'arg2' contains an INTEGER result for operation 5 or 11.

F80ISS.LIB Error Handler

If you prefer to have all floating-point errors trapped and reported to the console (:CO: device), you may use the alternate error handler provided in F80ISS.LIB. When this error handler is linked in explicitly and an error in a floating-point (REAL) arithmetic operation or intrinsic mathematical function occurs, the FORTRAN program stops. Control returns to ISIS-II if ISIS controls the run-time environment, and the following message is sent to the console:

***EXECUTION ERROR. REAL ARGUMENT ERROR *m* IN FUNCTION *n*.

where

m is a digit indicating the error code
n is an integer indicating the operator or function

These error and operation codes are as listed previously under "FORTRAN Run-Time Error Messages."

To link in the alternate error handler (for the ISIS-II run-time environment and software floating-point routines), use the following LINK command:

```
LINK input-list,F80ISS.LIB(FQFERH),F80RUN.LIB,&
F80ISS.LIB,FPEF.LIB,FPSOFT.LIB,PLM80.LIB,&
TO link-file
```

If the iSBC 310 interface is to be used, substitute FPHARD.LIB for FPSOFT.LIB in the LINK command above. The alternate error handler may be used only in the ISIS-II run-time environment.

User-Supplied Error Handlers

If you prefer not to use either of the error-handling routines provided in the run-time libraries, you may write your own error handler. (Note that the same error handler may be used whether you are performing arithmetic via software routines or via the iSBC 310 and interface.) Your error handler can interface with the FORTRAN floating-point routines in one of two ways:

- It can be labeled with the public name FQFERH and linked before the arithmetic library, thus substituting for the FORTRAN-provided error handlers.
- In addition, error handling can be "reset" dynamically (i.e., another error handler can be substituted during execution) by inserting into your FORTRAN code a call to the external subroutine FQFRST, as described below.

The calling sequence for FQFRST is:

```
CALL FQFRST(A,ERRH)
```

where A is a two-byte integer variable and ERRH is the address of your error- handling routine. The least significant bit of the high-order byte of A is a flag which, when set to 1, indicates that the user-supplied subroutine designated by ERRH will now be used as the floating-point error handler; if this flag is reset to 0, then the error handler named FQFERH will be activated. The low-order byte of A will become the new value (normally 0) of the Error Field, a byte which is explained more fully later under "Error Handling." Thus the standard settings of A are 0 and #100H. Note that by using FQFRST, you may change the error handler more than once, and at execution time rather than at link time.

Under RMX/80, each task must initialize the internal error handler address field by calling the external subroutine FQFSET. This routine is identical to FQFRST except that it also clears internal floating-point working accumulators and should be called only once per task. By using combinations of FQFSET and FQFRST, you can dedicate (and re-dedicate) error handlers to individual tasks or groups of tasks.

The error handler takes four parameters, in the order (ARG1, ARG2, OPCODE, ERROR$CODE). ARG1 and ARG2 are the REAL arguments of the floating-point operation. OPCODE and ERROR$CODE are one-byte INTEGER values representing the operation code and the error code as listed previously.

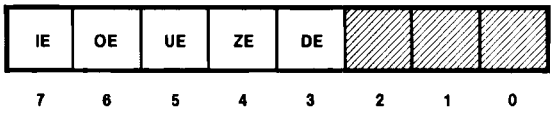
The following is an example of a routine you might supply in place of FQFERH to handle floating-point arithmetic errors. You can establish this error-handling routine as the arithmetic error handler dynamically at run time by using the following statement in your FORTRAN program:

```
CALL FQFRST(#100H,ALTERH)
```

```
$REENTRANT
*
*  A SAMPLE USER-SUPPLIED FORTRAN MATH ERROR HANDLER.

      SUBROUTINE ALTERH(ARG1,ARG2,FNCODE,ERCODE)

      REAL ARG1,ARG2
      INTEGER*1 FNCODE,ERCODE
      REAL RTEMP
      INTEGER*2 I2TEMP
      EQUIVALENCE (RTEMP,I2TEMP)

*  CASE 1: Square-root of a negative number (domain error).
*          Return the negative square-root of the argument s absolute value.
      IF ((ERCODE.EQ.2).AND.(FNCODE.EQ.10)) THEN
         ARG1 = -SQRT(ABS(ARG1))

*  CASE 2: Overflow on multiply or divide.
*          Return a very large real number, positive if ARG1 and ARG2
*          have the same sign, negative otherwise.
      ELSE IF ((ERCODE.EQ.3) .AND. ((FNCODE.EQ.3).OR.(FNCODE.EQ.4))) THEN
         ARG1 = SIGN(3.4E+38,ARG1) * SIGN(1.0,ARG2)

*  CASE 3: Too big a number to fix to a two-byte integer (overflow error)
*          Return 7FFFH, the largest two-byte positive integer, to the low
*          two bytes of ARG2 when ARG1 is non-negative, and 8000H, the
*          largest two-byte negative integer, otherwise.  (Note that
*          this is a particularly tricky situation.  You want to over-
*          write the low-addressed two bytes of ARG2 with 7FFFH or 8000H
*          without destroying the data in the high-addressed two bytes
*          of ARG2.  To do this, use the equivalenced REAL and INTEGER*2
*          temporary variables, RTEMP and I2TEMP.)
      ELSE IF ((ERCODE.EQ.3).AND.(FNCODE.EQ.11)) THEN
         RTEMP = ARG2
         IF (ARG1.GE.0) THEN
            I2TEMP = #7FFFH
         ELSE
            I2TEMP = #8000H
         END IF
         ARG2 = RTEMP

*  CASE 4: Domain error on ATAN2 (i.e., ARG1 = ARG2 = 0).
*          Return 0.
      ELSE IF ((ERCODE.EQ.2).AND.(FNCODE.EQ.117)) THEN
         ARG1 = 0

*  CASE 5: Division by zero.
*          Return ARG1 divided by a very small number.  (Note that such a
*          division may result in an overflow  causing this error handler
*          to be reinvoked for CASE 2 -- thus the REENTRANCY requirement )
      ELSE IF (ERCODE.EQ.1) THEN
         ARG1 = ARG1 / 1.2E-38

*  End of error-handling examples.  Note that no action will be taken on
*  errors that do not fall into one of the above cases.
      ENDIF

      RETURN
      END
```

Error Monitoring

The arithmetic routines maintain, as part of an internal floating-point record, a one-byte Error Field containing flags to indicate the occurrence of the various types of errors. Independently of the error handler, you can monitor accumulated error information between successive initializations of this Error Field by means of the external integer function FQFERR. FQFERR returns the current value of the Error Field byte, which is interpreted as shown below.



Bit	Interpretation
IE	Invalid argument
OE	Overflow
UE	Underflow
ZE	Attempted division by zero
DE	Domain error (argument range exceeded)

Bits 0, 1, and 2 are currently unused. Setting any of these bits to one causes undefined results.

The Error Field byte reflects the floating-point error situation since the last time the Error Field was cleared, which will generally be since the last call to FQFRST or FQFSET.

Run-Time I/O Errors

If an ISIS I/O error occurs at run-time, the message from the ISIS 'ERROR' routine precedes the FORTRAN 'EXECUTION ERROR' message. The ISIS message lists the ISIS-supplied STATUS for the I/O operation causing the error.

In the ISIS-II run-time environment, FORTRAN I/O error routines can issue the following message at run time:

```
***EXECUTION ERROR. FORTRAN I/O ERROR nnn

        NEAR LOCATION xxxH
```

In the RMX/80 environment, the error message is as follows:

```
***EXECUTION ERROR. FORTRAN I/O ERROR nnn TASK = taskname
```

The error message gives the memory location of the current instruction being executed (for ISIS-II) or the name of the active task (for RMX/80) at the time the error occurred.

The I/O error number 'nnn' is also returned as the value of the symbol designated by IOSTAT, if the IOSTAT specifier is included in a FORTRAN READ or WRITE statement. I/O error numbers are given in the following list.

NOTE

Some error numbers —e.g., 117, 122, 123—are omitted from this list. No errors corresponding to these numbers exist, and they should never appear in an error message.

Error No.	Message
112	A syntax error exists in a formatted binary or hexadecimal input field.
113	A syntax error exists in a list-directed alphanumeric input field.
114	A syntax error exists in a formatted or list-directed logical input field.
115	A syntax error exists in a formatted or list-directed real input field.
116	A syntax error exists in a formatted or list-directed integer input field.
118	The RECL specifier is invalid in an OPEN statement for a sequential, unformatted connection.
119	The STATUS = 'SCRATCH' specifier is invalid in an OPEN statement with a named FILE.
120	The string supplied for the FILE specifier of an OPEN statement or for the device specifier in the UNIT preconnection control is not a valid ISIS path-name.
121	An illegal string has been passed as the CARRIAGE specifier of an OPEN statement.
124	A WRITE statement is attempting to write too many characters to a record in an internal file.
125	A READ or WRITE statement is attempting to read from or write to more records than were specified to be in the internal file.
126	A READ statement is attempting to read too many characters from a record in an internal file.
128	An ISIS, RMX/80 Terminal Handler, or RMX/80 Disk File System error has occurred during a WRITE operation.
129	An ISIS, RMX/80 Terminal Handler, or RMX/80 Disk File System error has occurred during a READ operation.
130	The specified unit is not connected.
131	The unit specified by a formatted READ or WRITE is not connected with the proper ACCESS and FORMAT attributes.

Error No.	Message
132	No BACKSPACE or REWIND has occurred since an ENDFILE was specified for the unit selected by a sequential READ or WRITE.
134	A READ statement is attempting to read from a unit whose CARRIAGE attribute is 'CONSOLE'.
135	A READ statement is attempting to read from a unit whose CARRIAGE attribute is 'FORTRAN'.
136	The edit descriptor in the format string matches a variable in the I/O list whose type is inappropriate for this descriptor.
141	An ISIS or RMX/80 Disk File System error has occurred while closing a pre-existing file whose length was shortened during the execution of this program. The file cannot be copied.
142	An ISIS or RMX/80 Disk File System error has occurred during a BACKSPACE operation.
143	An ISIS or RMX/80 Disk File System error has occurred during a REWIND operation.
144	An ISIS or RMX/80 Disk File System error has occurred during a OPEN operation.
145	An ISIS or RMX/80 Disk File System error has occurred when attempting to move to the beginning of a record during a direct-access READ or WRITE.
146	An ISIS or RMX/80 Disk File System error has occurred when ending an output record for a sequential disk file.
152	An ISIS or RMX/80 Disk File System error has occurred during a CLOSE operation.
155	An error has occurred while attempting to OPEN, in update or read mode, a file whose STATUS = 'OLD'.
156	The OPEN statement cannot specify STATUS = 'OLD' for a nonexistent file.
158	An error has occurred while trying to OPEN, in update or write mode, a file whose STATUS = 'NEW'.
160	The OPEN statement cannot specify STATUS = 'NEW' for a file that already exists.
162	The physical end of the format string occurs before the logical end of the format string.
163	A repetition specifier preceding a repeatable edit descriptor or parenthesized format substring cannot be zero.
164	The format string must start with '('.
166	The nesting of parenthesized format substrings cannot exceed three levels.

Error No.	Message
167	A nondigit follows a '+' or '−' in the format string.
168	A 'P' must follow a '+' <number> in the format string.
171	The substring of the format string that should be used to revert format control does not contain any repeatable edit descriptor.
172	A ',' or ')' or '/' is expected to terminate an edit descriptor.
173	An I, L, F, E, B, or Z edit descriptor is not followed by a field width specification.
175	A period must follow the field width in an E or F edit descriptor.
176	A number must follow the period in an E or F edit descriptor.
177	An expected repeatable edit descriptor is missing.
179	A number must follow the exponent field specifier 'E' in an E edit descriptor.
180	The fraction specifier is greater than the field-width specifier in an E or F edit descriptor.
181	The width specified for an E edit descriptor is not large enough for the specified fraction and exponent fields.
182	The format string ends logically before the last non-blank character.
191	A filename must be specified in the OPEN statement for a file whose STATUS = 'OLD'.
192	A filename must be specified in the OPEN statement for a file whose STATUS = 'NEW'.
193	An error has occurred while trying to OPEN a file in update mode whose STATUS = 'SCRATCH'. The file is known to be nonexistent.
194	The program is trying to OPEN an existing file with STATUS = 'SCRATCH'.
196	An ISIS error has occurred while trying to OPEN a file whose STATUS = 'UNKNOWN'.
198	A syntax error exists in preconnection specifications in the command tail.
200	A CLOSE statement specifies STATUS = 'KEEP' for a file whose current STATUS = 'SCRATCH'.
201	The STATUS specifier for a CLOSE statement must be either 'KEEP' or 'DELETE'.
202	The UNIT specifier in an OPEN statement cannot have a value greater than 255.

Error No.	Message
203	The FILE named in an OPEN statement is already connected to a different unit.
204	An OPEN statement requires a new connection but the connection table is full.
205	An OPEN statement for a direct-access file is missing a RECL specifier (or RECL = 0).
206	The string passed in the ACCESS specifier of an OPEN statement is illegal.
207	The string passed in the FORMAT specifier of an OPEN statement is illegal.
208	The string passed in the BLANK specifier of an OPEN statement is illegal.
209	An OPEN statement referencing a file connected for unformatted I/O cannot include a BLANK specifier.
210	The string passed in the STATUS specifier of an OPEN statement is illegal.
211	An OPEN statement is attempting to change the attributes of a previously-connected file. Only the BLANK attribute for a formatted file or the RECL attribute for a sequential, formatted file may be changed.
217	A formatted READ statement is attempting to read more characters than are present in the record.
221	An end-of-file has occurred on the file being read and no END specifier has been supplied.
222	A Hollerith or literal edit descriptor cannot appear in the format string matching an input list.
225	An invalid scale factor has been found when trying to write a real value.
232	A READ or WRITE statement is attempting to read from or write to a direct-access file that is not on diskette.
233	A WRITE statement is attempting to write to a unit that cannot be written to; e.g., :CI:.
234	A READ statement is attempting to read from a unit that cannot be read from; e.g., :LP:.
235	The program has tried to REWIND, BACKSPACE, or ENDFILE a unit not connected for sequential I/O.
238	The attempted REWIND is inappropriate for this file.
242	The attempted BACKSPACE is inappropriate for this file.

250 A direct-access READ statement is attempting to read more bytes than are present in the record.

251 An unformatted READ statement is attempting to read more bytes than are present in the record.

252 A <CR, LF> has been found in the middle of a formatted, direct-access record.

253 A direct-access, formatted record must be terminated with a <CR, LF>.

255 A direct-access WRITE statement is trying to write more data than will fit into a single direct-access record.

'ERR' Specifier

If a FORTRAN I/O statement includes the ERR specifier in its control list, control is transferred to the statement designated by ERR when an error is detected. No library routine is invoked in this case.

'IOSTAT' Specifier

If a FORTRAN I/O statement includes the IOSTAT specifier in its control list, I/O operations return a numerical code as the value of a symbol designated by IOSTAT. This code is the same number shown in the 'FORTRAN I/O ERROR' message (0 means no error detected). See 'Run-Time I/O Errors' above.

ISIS-II Error Messages

The following error messages are discussed in more detail in the *ISIS-II User's Guide*, but are summarized here for your convenience. By convention, error numbers 1-99 are reserved for errors that originate in or are detected by the resident routines of ISIS-II or by RMX/80. In the following list an asterisk precedes fatal errors. The other errors are generally nonfatal unless they are issued by the CONSOL system call.

0 No error detected.
* 1 Insufficient space in buffer area for a required buffer.
2 AFTN does not specify an open file.
3 Attempt to open more than six files simultaneously.
4 Illegal filename specification.
5 Illegal or unrecognized device specification in filename.
6 Attempt to write to a file open for input.
* 7 Operation aborted; insufficient diskette space.†

† Error number 7 (insufficient diskette space) may sometimes occur when it appears that there should be enough space on the disk. This happens because when a file has been truncated in the course of processing (e.g., via a BACKSPACE followed by an ENDFILE), the FORTRAN I/O routines copy that file onto a scratch file called FTCHOP.TMP, which consumes additional disk space.

8	Attempt to read from a file open for output.
9	No more room in diskette directory.
10	Filenames do not specify the same diskette.
11	Cannot rename file; name already in use.
12	Attempt to open a file already open.
13	No such file.
14	Attempt to open for writing (output or update) or to delete or rename a write-protected file.
*15	Attempt to load into ISIS area or buffer area.
*16	Incorrect ISIS binary format.
17	Attempt to rename or delete a file not on diskette.
18	Unrecognized system call.
19	Attempt to seek in a file not on diskette.
20	Attempt to seek backward past beginning of file.
21	Attempt to rescan a file not line edited.
22	Illegal ACCESS parameter to OPEN or access mode impossible for file specified (input mode for :LP:, for example).
23	No filename specified for a diskette file.
*24	Input/output error on diskette (see below).
25	Incorrect specification of echo file to OPEN.
26	Incorrect SWID parameter in ATTRIB system call.
27	Incorrect MODE parameter in SEEK system call.
28	Null file extension.
*29	End of file on console input.
*30	Drive not ready.
31	Attempted seek on file open for output.
32	Can't delete an open file.
33	Illegal system call parameter.
34	Bad RETSW parameter to LOAD.
35	Attempt to extend a file opened for input by seeking past end-of-file.
201	Unrecognized switch.
202	Unrecognized delimiter character.
203	Invalid command syntax.
204	Premature end of file.
206	Illegal diskette label.
207	No END statement found in input.
208	Checksum error.
209	Illegal record sequence in object module file.
210	Insufficient memory to complete job.
211	Object module record too long.
212	Bad object module record type.
213	Illegal fixup specified in object module file.
214	Bad parameter in a SUBMIT file.
215	Argument too long in a SUBMIT file.
216	Too many parameters in a SUBMIT file.
217	Object module record too short.
218	Illegal object module record format.
219	Phase error.
220	No end of file record in object module file.
221	Segment exceeds 64K bytes.
222	Unrecognized record in object module file.
223	Fixup record pointer is incorrect.
224	Illegal record sequence in object module file.
225	Illegal module name specified.
226	Module name exceeds 31 characters.
227	Command syntax requires left parenthesis.
228	Command syntax requires right parenthesis.
229	Unrecognized control specified in command.
230	Duplicate symbol found.
231	File already exists.

232 Unrecognized command.
233 Command syntax requires a "TO" clause.
234 File name illegally duplicated in command.
235 File specified in command is not a library file.
236 More than 249 common segments in input files.
237 Specified common segment not found in object file.
238 Illegal stack content record in object file.
239 No module header in input object file.
240 Program exceeds 64K bytes.

When error number 24 occurs, an additional message is sent to the console:

FDCC = 00nn

where *nn* is a hexadecimal number with the following meaning:

01 Deleted record.
02 CRC error (data field).
03 Invalid address mark.
04 Seek error.
08 Address error.
0A CRC error (ID field).
0E No address mark.
0F Incorrect data address mark.
10 Data overrun or data underrun.
20 Write protect.
40 Write error.
80 Not ready.

RMX/80 Error Codes

The following error codes are discussed in more detail in the *RMX/80 User's Guide*, but are summarized here for your convenience. By convention, error numbers 1-99 are reserved for errors that originate in or are detected by the resident routines of ISIS-II or by RMX/80. On completion of a requested service, RMX/80 indicates errors by returning the appropriate error code in the low-order byte of the STATUS field of the response message.

The following error codes may be returned by the RMX/80 Disk File System. Many of these errors correspond to ISIS-II errors; in these cases, the codes are the same. Note, however, that errors considered fatal in ISIS are not so considered in DFS.

0 No error detected.
4 Illegal FILENAME specified in File Name Block.
5 DEVICENAME in File Name Block not in Device Configuration Table.
6 Attempt to write to a file opened for input.
7 No more space on disk.*

* Error number 7 (no more space on disk) may sometimes occur when it appears that there should be enough space on the disk. This happens because when a file has been truncated in the course of processing (e.g., via a BACKSPACE followed by an ENDFILE), the FORTRAN I/O routines copy that file onto a scratch file called FTCHOP.TMP, which consumes additional disk space.

8 Attempt to read a file opened for output.

9 No more room in disk directory.

10 File Name Blocks in RENAME request do not specify same device.

11 Cannot rename file; name already in use.

12 File already open.

13 No such file.

14 Attempt to open a write-protected file for output or update, or attempt to delete or rename a write-protected file.

16 Incorrect object program format.

18 Unrecognized message TYPE.

20 Attempt to seek backwards past beginning of file.

22 Illegal ACCESS in OPEN message.

24 Input/output error on disk.

26 Illegal SWID in ATTRIB message.

27 Illegal MODE in SEEK message.

30 Drive not ready.

31 Attempt to seek on file open for output.

32 Attempt to delete an open file.

35 Attempt to seek past end of file opened for input.

40 Request sent to wrong exchange.

41 Insufficient free memory space to open file.

42 DRIVE specified in DISKIO request is not in Device Configuration Table.

43 Drive timeout—the drive has not responded to an I/O request within a set period of time (10 seconds for iSBC 80/20 or 80/30 systems; for 80/10 systems, refer to *RMX/80 User's Guide*).

44 SEEK request with SEEK not present in system.

45 Format driver missing.

If error 24 (input/output error) occurs, DFS places one (or more, if multiple errors occur) hexadecimal codes in the *high-order* byte of STATUS in the response message to identify the type of I/O error, as follows:

01 Deleted record.

02 Cyclic Redundancy Check character error (data field).

03 Invalid address mark.

04 Seek error.

08 Address error.

0A Cyclic Redundancy Check character error (ID field).

0E No address mark.

0F Incorrect data address mark.

10 Data overrun or underrun.

20 Write protect.

40 Write error.

80 Not ready.

The RMX/80 Terminal Handler returns only one possible error code: 18, which denotes an invalid read or write request message type. For error codes returned by other RMX/80 extensions (e.g., Free Space Manager or Analog Handlers), refer to the *RMX/80 User's Guide.*

LINK Error Messages

The following LINK error messages indicate a fatal error has occurred and generally require you to recompile and relink your program.

```
filename, BAD FIXUP RECORD
filename, BAD RECORD SEQUENCE
filename, CHECKSUM ERROR
filename, ILLEGAL RECORD FORMAT
filename, NOT LIBRARY
filename, PHASE ERROR
filename, PREMATURE EOF
filename, RECORD TOO LONG
filename, RECORD TOO SHORT
filename, RELO FILE SEQUENCE ERROR
filename, SEGMENT TOO LARGE
filename, TOO MANY COMMON SEGMENTS
INSUFFICIENT MEMORY
```

The following messages indicate nonfatal errors. They do not prevent LINK from performing its assigned tasks, but the messages are reported as warnings.

```
MORE THAN ONE MAIN MODULE
modname-MODULE NOT FOUND IN LIBRARY
name, COMMON/PUBLIC/EXTERNAL NAME CLASH
name, HAS DIFFERING TYPES
name, MULTIPLE DEFINITION
name, UNEQUAL COMMON LENGTHS
```

For a detailed discussion of these error messages, see Chapter 4 of the *ISIS-II User's Guide.*

LOCATE Error Messages

The following LOCATE error messages indicate a fatal error has occurred. LOCATE terminates and returns control to ISIS-II.

```
filename, CHECKSUM ERROR
common name, COMMON NOT FOUND
filename, FIXUP BOUNDS ERROR
filename, ILLEGAL RELO RECORD
filename, ILLEGAL STACK CONTENT RECORD
filename, INSUFFICIENT MEMORY
token, INVALID SYNTAX
filename, NO MODULE HEADER RECORD
filename, PREMATURE EOF
filename, PROGRAM EXCEEDS 64K
filename, RECORD TOO LONG
```

The following messages indicate nonfatal errors. LOCATE completes processing before returning to ISIS-II.

```
INPAGE SEGMENT COERCED TO PAGE RELTYP
MEMORY CONFLICT FROM aaaaH THROUGH aaaaH
UNSATISFIED EXTERNAL REFERENCE AT aaaaH
```

See Chapter 4 of the *ISIS-II User's Guide* for a detailed discussion of these messages.

APPENDIX C
NUMBER FORMATS

This appendix provides definitions related to the internal representation of numbers in the 8080 or 8085 memory, along with the strategy used for rounding floating-point values and exponent wraparound.

Floating-Point Numbers

Floating-Point Standard

Floating-point number representations and floating-point arithmetic conform to the Intel floating-point standard, which is described in the following article:

> Palmer, John F., "The Intel Standard for Floating-Point Arithmetic," *Proceedings of the First International Computer Software and Applications Conference* (Chicago: IEEE Computer Society), November 1977, pp. 107-112.

Floating-Point Zero

The format with all bits equal to zero is defined as the unique floating-point zero. No other form for floating-point zero is supported.

Invalid Numbers

For floating-point numbers, all bit patterns are valid except those described here.

The first set of invalids are those whose exponent field is set to all ones. This set is used for infinities, indefinites, pointers, etc. Infinities are defined as:

+INF	's' bit = 0; all other bits = 1
−INF	all bits = 1

The indefinite form is:

IND	's' = 0; exponent bits all = 1; fraction bits = 0

A second set of bit patterns is currently defined as invalid. These are numbers whose exponent field is zero with at least one other bit set to one.

Floating-Point Number Format

The format of floating-point numbers in memory is as shown in Figure C-1.

HIGH ADDRESS	S	e_8	e_7	e_6	e_5	e_4	e_3	e_2
	e_1	f_{22}	f_{21}	f_{20}	f_{19}	f_{18}	f_{17}	f_{16}
	f_{15}	f_{14}	f_{13}	f_{12}	f_{11}	f_{10}	f_9	f_8
LOW ADDRESS (POINTER VALUE)	f_7	f_6	f_5	f_4	f_3	f_2	f_1	f_0

Figure C-1. Floating-Point Number Format

The address of the number (pointer value) is the low address. The three fields within the floating-point format are:

s Sign bit. Sign-magnitude representation where s=0 means positive and s=1 means negative.

e Exponent bits. The exponent is offset by 2^7-1. All zeros and all ones in the exponent field are currently reserved for the floating-point zero and the invalid numbers respectively described above.

f Fraction bits. When the exponent is nonzero, a one bit is assumed at the left of the fraction; the binary point is between the assumed bit and the explicit fraction bit.

The number base is binary. The value of a given binary representation (where 's' is the sign bit, 'e' is a binary exponent value, and 'f' is a binary fraction value) can be formulated as:

$$(-1)^{s}.2^{e-(2^7-1)}.(1.+.f) \text{ where } e \neq 0 \text{ and } e \neq FF$$

Example: 3F800000 is equivalent to 1.

Rounding

If rounding is required to produce the final result of a floating-point operation, 'unbiased' rounding is used. With this typeof rounding, the result is rounded up or down depending on whether the first bit beyond the last bit being retained is 1 or 0. In the ambiguous case where the true result es exactly midway between two floating-point numbers, the nearest 'even' number is returned (that is, the last bit retained is forced to a zero). Therefore, if no error occurs, the result is the floaing-point number closest to the true result.

Example: 40490FDB is the floating-point representation of π.

Exponent Wraparound

When overflow or underflow occurs during floating-point operations, the correct fraction results but the exponent is 'wrapped around.' This is consistent with the philosophy that no information should be lost and that you, the user, should be able to decide what you want to do when an overflow/underflow exception occurs.

A 'wrapped around' exponent is defined to be e_w where the true (offset) exponent e_t can be derived from e_w by considering an expanded range of exponents and

on overflow $e_t=e_w+(3*2^6-2)$

on underflow $e_t=e_w-(3*2^6-2)$

Example: 00800001−00800000 = 54000000.

Integers

Integers in FORTRAN-80 are signed two's-complement numbers; they may be 1, 2, or 4 bytes long. Unless the STORAGE compiler control is used to specify a different default, the default length for integer variables is 2.

The format of 1-, 2-, and 4-byte integers in memory is shown in Figure C-2. In this figure, s is the sign bit and i_l is the low-order bit. The address of the number (pointer value) is the low address.

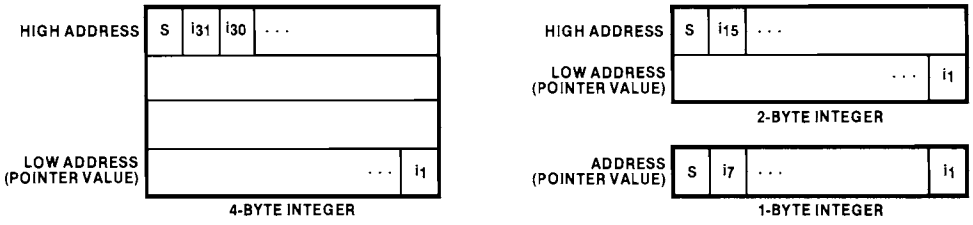


Figure C-2. Integer Formats

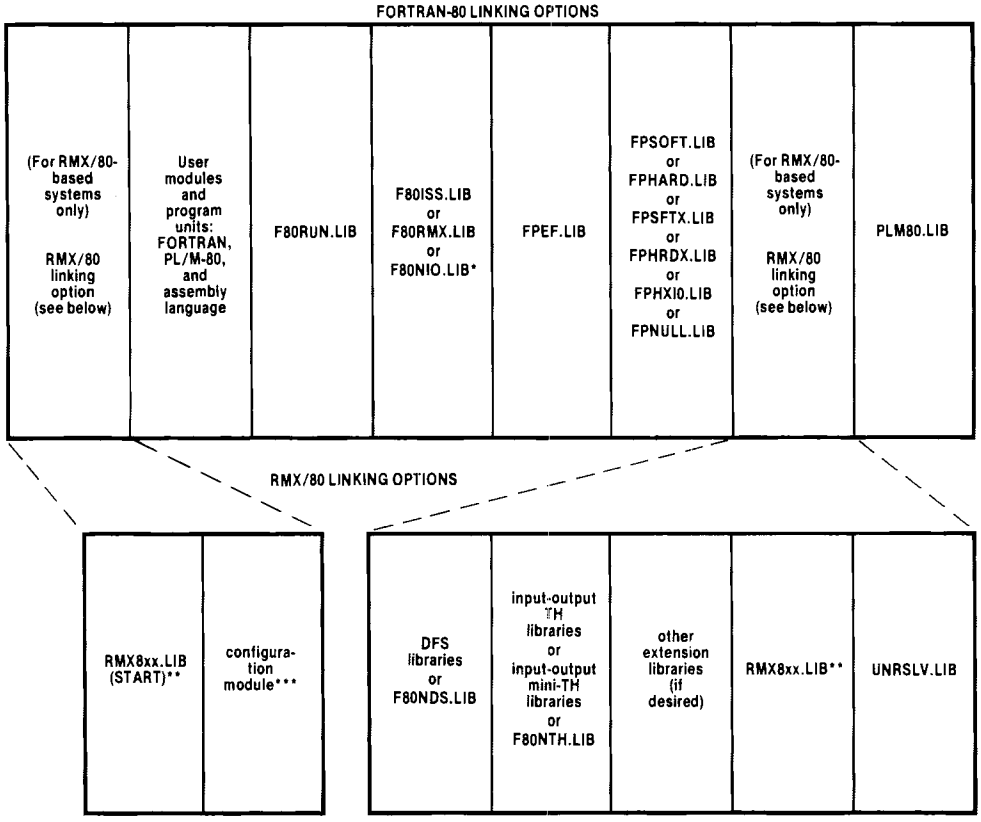
APPENDIX D
SUMMARY OF LINK OPTIONS

The syntax of the LINK command is:

LINK input-list TO link-file [link-controls]

The diagram below shows the FORTRAN-80 run-time libraries from which you may choose when linking your program(s) or RMX/80 system and the order (from left to right) in which they must be given in the 'input-list' to the LINK command. Except where otherwise specified, one and only one item *must* be selected from each block in the diagram.

Definitions of the libraries that appear in the diagram are given on the following page.



* Note that you must *not* link in F80NIO.LIB if your program includes any STOP or PAUSE statements; doing so may cause the 8080 processor to enter a halt state. Also note that in some cases when running in a stand-alone (non-ISIS, non-RMX) environment, both F80NIO.LIB and F80ISS.LIB should be linked in (in that order). Refer to 'Linking and Locating' in Chapter 7.

** 'xx' is 20, 30, or 10 for iSBC 80/20, 80/30, or 80/10 systems respectively.

*** Your configuration module may actually be included anywhere in the 'user modules and program units' section; it need not precede all other user-coded modules. It is separated out in this diagram merely to show that it is unique to programs running under RMX/80.

Library	Description
F80RUN.LIB	Integer arithmetic, array indexing, and miscellaneous routines
F80ISS.LIB	Input/output for the ISIS-II environment
F80RMX.LIB	Input/output for the RMX/80 environment
F80NIO.LIB	External reference library for programs using no FORTRAN I/O except port I/O
FPEF.LIB	Floating-point intrinsic functions
FPSOFT.LIB	Floating-point arithmetic library for the non-RMX (ISIS-II or stand-alone) environment
FPHARD.LIB	Floating-point arithmetic library using the iSBC 310 math unit for the non-RMX (ISIS-II or stand-alone) environment
FPSFTX.LIB	Floating-point arithmetic library for the RMX/80 environment
FPHRDX.LIB	Floating-point arithmetic library using the iSBC 310 math unit for RMX/80, iSBC 80/20 and 80/30 systems
FPHX10.LIB	Floating-point arithmetic library using the iSBC 310 math unit for RMX/80, iSBC 80/10 systems
FPNULL.LIB	External reference library for programs using no floating-point operations
PLM80.LIB	Support for library modules coded in PL/M
RMX8xx.LIB	Library containing the RMX/80 Nucleus (RMX820.LIB, RMX830.LIB or RMX810.LIB); the START module in this library must be linked explicitly at the beginning of 'input-list', as shown in the diagram
F80NDS.LIB	External reference library for programs running under RMX/80 without the Disk File System
F80NTH.LIB	External reference library for programs running under RMX/80 without the Terminal Handler
UNRSLV.LIB	External reference library for resolving unsatisfied references made by RMX8xx.LIB

The RMX/80 linking options—including Disk File System (DFS) libraries, Terminal Handler (TH) libraries, minimal Terminal Handler (mini-TH) libraries, and other extension libraries—are described in detail in the *RMX/80 User's Guide*.

Note that these libraries come from different software products. F80RUN.LIB, F80ISS.LIB, F80NIO.LIB, FPEF.LIB, FPSOFT.LIB, FPHARD.LIB, FPNULL.LIB, and PLM80.LIB are provided in the basic FORTRAN-80 package. F80RMX.LIB, FPSFTX.LIB, FPHRDX.LIB, FPHX10.LIB, F80NDS.LIB, and F80NTH.LIB are supplied in the FORTRAN-80 Run-Time Package for RMX/80 Systems (iSBC 801). RMX8xx.LIB, UNRSLV.LIB, the DFS libraries, the TH libraries, the mini-TH libraries, and the other extension libraries are part of the RMX/80 software package. (RMX/80 also includes PLM80.LIB.)

APPENDIX E
SPEEDS AND STACK REQUIREMENTS FOR FLOATING-POINT OPERATIONS

The following table gives execution speeds and minimum stack requirements for FORTRAN floating-point operations and intrinsic functions as implemented by the floating-point libraries for various run-time environments.

Execution times are given in milliseconds, and stack requirements (parenthesized) are given in bytes. Stack sizes do not take into account the extra stack required to process a floating-point error (20 bytes if the default arithmetic error handler is used.) Timings were determined on an Intellec Microcomputer Development System in which one clock cycle is 0.576 μsec. Execution times for software addition and subtraction are normalizing-dependent, and the times listed are "typical." Execution times for the RMX libraries assume that the previous floating-point operation was performed within the same task.

You can estimate the amount of stack space required for your floating-point operations in each program module by finding, in the table, the maximum stack requirement for any one floating-point operation you use, then adding 10-20% extra to that as a "safety factor."

Operation or Intrinsic Function	FPSOFT.LIB		FPSFTX.LIB		FPHARD.LIB		FPHRDX.LIB*	
	msec	(bytes)	msec	(bytes)	msec	(bytes)	msec	(bytes)
addition	1.0	(32)	1.1	(32)	0.18	(6)	0.35	(10)
subtraction	0.8	(32)	0.83	(32)	0.18	(6)	0.35	(10)
multiplication	1.7	(40)	1.8	(40)	0.18	(6)	0.35	(10)
division	3.8	(36)	3.9	(36)	0.18	(6)	0.35	(10)
negation	0.15	(10)	0.18	(10)	0.14	(4)	0.30	(10)
square root	14.	(26)	14.	(26)	0.11	(4)	0.26	(10)
square	2.2	(42)	2.3	(42)	0.09	(4)	0.24	(10)
comparison (two negative arguments)	0.36	(16)	0.38	(16)	0.83	(24)	1.8	(28)
comparison (two positive arguments)	0.35	(0)	0.38	(0)	0.26	(0)	0.62	(0)
comparison (positive or negative argument to zero)	0.15	(8)	0.18	(8)	0.09	(2)	0.26	(8)
load internal floating-point record	0.26	(16)	0.29	(16)	0.16	(6)	0.33	(10)
store internal floating-point record	0.23	(14)	0.25	(14)	0.14	(6)	0.30	(10)
converting 32-bit integer to float	0.71	(22)	0.73	(22)	0.15	(6)	0.32	(10)
converting 16-bit integer to float	1.0	(36)	1.1	(36)	0.17	(8)	0.32	(10)
converting float to 16-bit integer	1.1	(18)	1.2	(18)	0.67	(24)	1.1	(30)
converting float to 32-bit integer	0.55	(8)	0.56	(10)	0.85	(26)	1.4	(30)
converting binary to decimal	—	(93)	—	(93)	—	(77)	—	(91)
converting decimal to binary	—	(27)	—	(27)	—	(27)	—	(27)

*Stack requirements for FPHX10.LIB are the same as for FPHRDX.LIB. Execution times may be slightly slower.

Operation or Intrinsic Function	FPSOFT.LIB		FPSFTX.LIB		FPHARD.LIB		FPHRDX.LIB*	
	msec	(bytes)	msec	(bytes)	msec	(bytes)	msec	(bytes)
AINT (a)	3.1	(30)	3.2	(32)	1.3	(30)	2.3	(36)
ANINT (a)	4.7	(32)	4.9	(34)	1.8	(32)	3.3	(38)
NINT (a)	3.1	(64)	3.2	(62)	1.3	(66)	2.3	(78)
ABS (a)	0.14	(10)	0.17	(10)	0.14	(4)	0.30	(10)
MOD (a1,a2)	13.2	(54)	13.7	(54)	3.4	(38)	6.4	(44)
SIGN (a1,a2)	1.9	(40)	2.1	(50)	1.4	(30)	2.6	(34)
DIM (a1,a2)	2.8	(38)	3.1	(38)	1.4	(28)	2.8	(32)
EXP (a)	30.	(80)	31.	(82)	7.7	(76)	14.	(94)
ALOG (a)	28.	(82)	28.	(82)	4.6	(38)	8.6	(44)
ALOG10 (a)	28.	(82)	28.	(82)	4.6	(38)	8.7	(44)
SIN (a)	68.	(96)	71.	(96)	20.	(70)	37.	(76)
COS (a)	70.	(96)	73.	(96)	21.	(70)	38.	(76)
TAN (a)	57.	(90)	59.	(90)	15.	(66)	27.	(78)
ASIN (a)	55.	(84)	56.	(84)	9.4	(50)	17.	(54)
ACOS (a)	62.	(86)	63.	(86)	10.	(52)	18.	(56)
ATAN (a)	30.	(78)	30.	(80)	6.6	(66)	11.	(76)
ATAN2 (a)	38.	(88)	38.	(90)	10.	(80)	18.	(94)
SINH (a)	37.	(90)	39.	(90)	11.	(66)	20.	(72)
COSH (a)	34.	(82)	35.	(82)	9.	(58)	17.	(64)
TANH (a)	37.	(90)	39.	(90)	11.	(66)	21.	(72)

*Stack requirements for FPHX10.LIB are the same as for FPHRDX.LIB. Execution times may be slightly slower.

APPENDIX F
PROVIDING REENTRANCY FOR NON-RMX FLOATING-POINT LIBRARIES

The non-RMX floating-point libraries, FPSOFT.LIB and FPHARD.LIB, are not reentrant. However, two routines, FQFSAV and FQFRES, are included in these libraries to enable the user to effect reentrancy. If it is possible for a floating-point operation in a non-RMX system to be interrupted, FQFSAV and FQFRES should be called from any interrupt routine that may use floating-point operations or which may call another routine that uses floating-point operations.

FQFSAV should be called at the start of the interrupt handler, since it saves the floating-point status, and FQFRES should be called at the end, since it restores the previous status. There is no limitation on the number of levels in which these calls may be nested, other than the amount of stack space available.

INDEX

Primary references are italicized in this index.