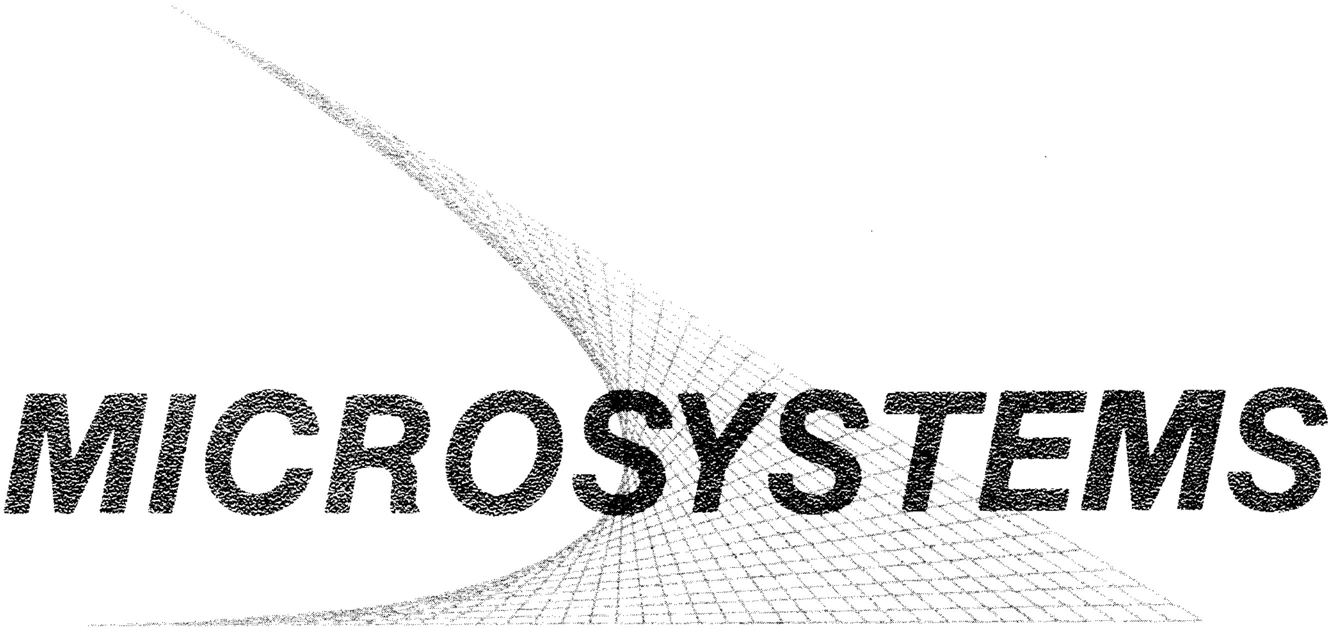
MOTOROLA

M68KFORTRN/D3

M68000 Family Resident FORTRAN Compiler User's Manual



QUALITY • PEOPLE • PERFORMANCE

M68KFORTRN/D3

NOVEMBER 1983

M68000 FAMILY

RESIDENT FORTRAN COMPILER

USER'S MANUAL

The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. Furthermore, Motorola reserves the right to make changes to any products herein to improve reliability, function, or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights or the rights of others.

EXORmacs, RMS68K, VERSAdos, VERSAmodule, VMC 68/2, VMEmodule, and VME/10 are trademarks of Motorola Inc.

LARK is a trademark of Control Data Corporation.

Third Edition

Second Edition September 1982

TABLE OF CONTENTS

Page

CHAPTER 1 GENERAL INFORMATION

CHAPTER 2 PREPARATION OF FORTRAN SOURCE PROGRAMS

CHAPTER 3 USING THE FORTRAN COMPILER

CHAPTER 4 OUTPUT LISTINGS

CHAPTER 5 CREATION OF AN EXECUTABLE LOAD MODULE

TABLE OF CONTENTS (cont'd)

Page

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

GENERAL INFORMATION

1.1 INTRODUCTION

This manual describes how to use the M68000 Family Resident FORTRAN Compiler. It also describes the language differences between the M68000 FORTRAN and the ANSI 77 subset standard (see Appendix E).

1.2 FUNCTION OF FORTRAN COMPILER

NOTE

> Unless otherwise specified, the designations "M68000" and "MC68000" refer to the entire M68000 family of microprocessors.

The FORTRAN Compiler translates source programs written in FORTRAN into MC68000 machine language, using one of the VERSAdos systems listed in paragraph 1.4. The MC68000 machine language relocatable programs produced by the Compiler will be referred to as object programs throughout the rest of this document.

```
                                                    MC68000
FORTRAN                   FORTRAN                   MACHINE
SOURCE       --->         COMPILER      --->        LANGUAGE
PROGRAM                                             PROGRAM
```

FIGURE 1-1. Function of the FORTRAN Compiler

1.3 FEATURES

The features of the Compiler are as follows:

- Language conformity to the ANSI FORTRAN 77 subset.
- Capability of performing bit operations.
- Capability of creating reentrant object programs.

1.4 OPERATING ENVIRONMENT FOR THE FORTRAN COMPILER

The following hardware and software facilities are required as a minimum to invoke the FORTRAN Compiler:

a. Hardware

- One of the following MC68000-based systems:

  EXORmacs Development System
  VMC 68/2 Microcomputer System
  VME/10 Microcomputer System
  VERSAmodule 01 or 02 Monoboard Microcomputer
  VMEmodule Monoboard Microcomputer

- 384K bytes of memory
- A keyboard/CRT terminal
- One of the following disk configurations:

  two floppy disk drives
  a LARK drive
  a Winchester drive

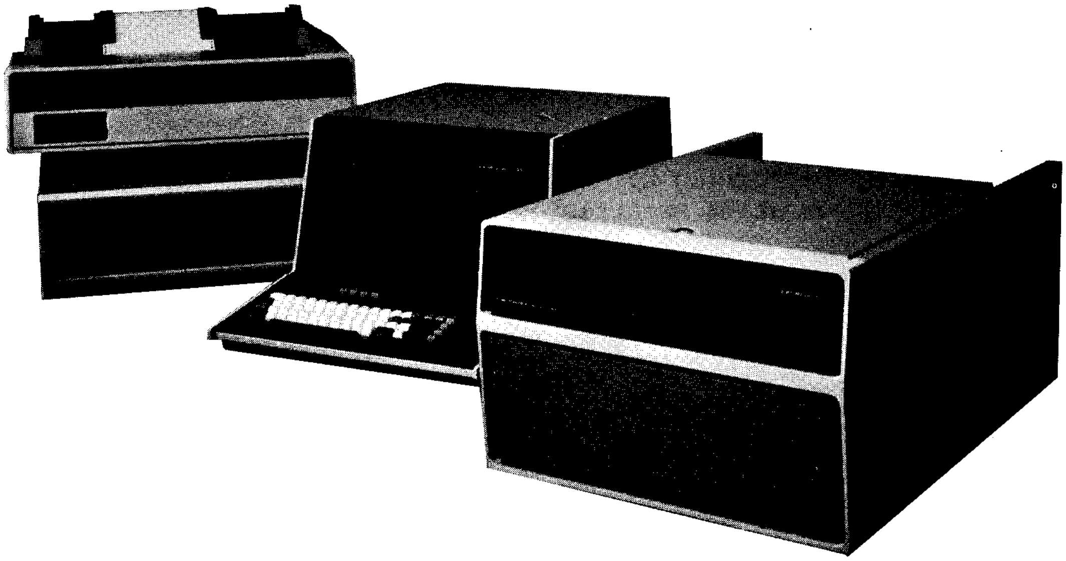
Figure 1-2 illustrates the standard configuration for an EXORmacs, which includes the above hardware elements along with a serial printer.

Figure 1-3 illustrates a VME/10 system. The standard VME/10 configuration allows the addition of a printer when an MVME410 dual parallel port module is used.

b. Software

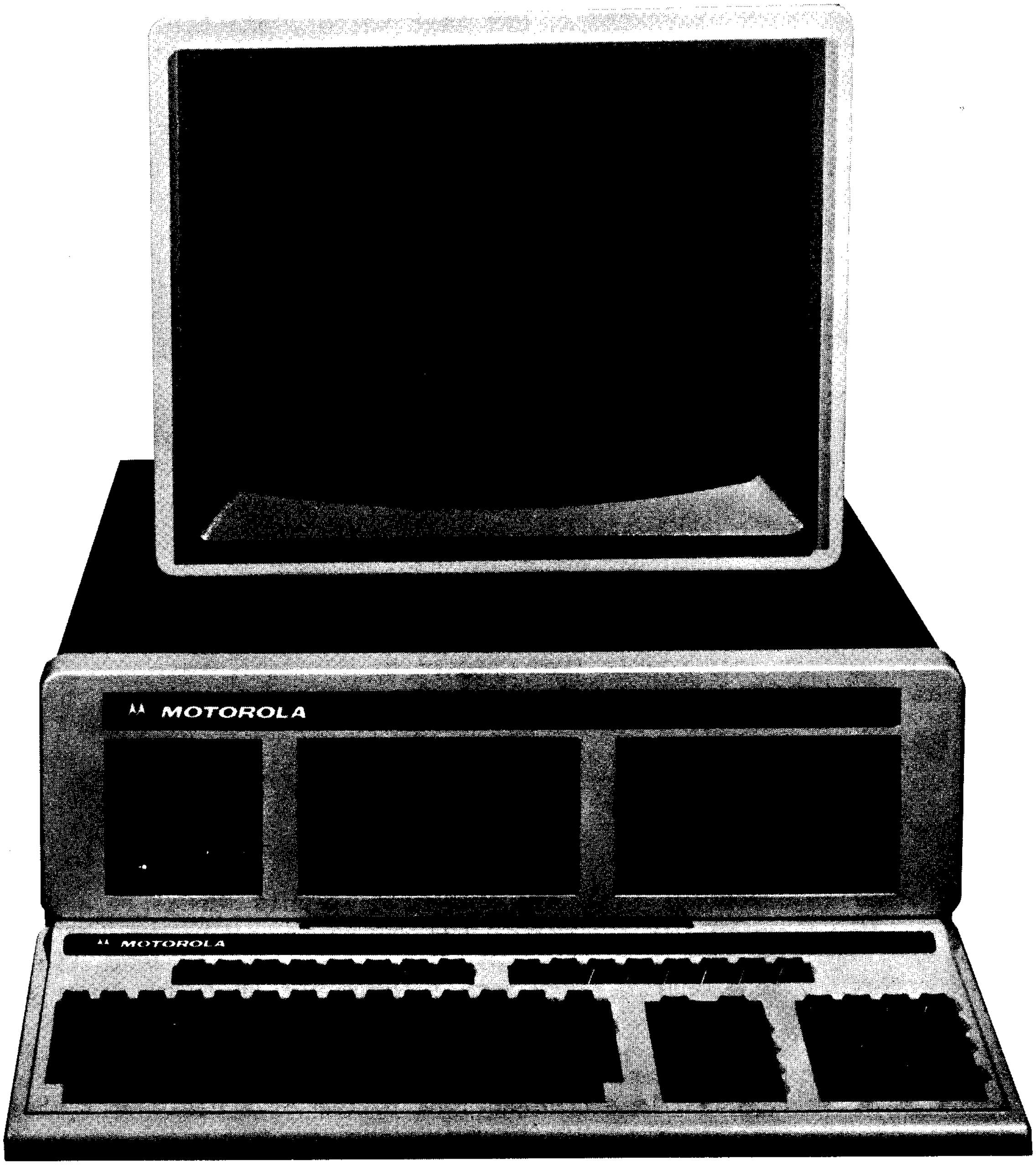
- VERSAdos (Disk Operating System)

  VERSAdos is a disk operating system available for the hardware systems listed in paragraph 1.4.a. It coordinates control of the Compiler, the data, and the disk.



01158/7361

FIGURE 1-2. EXORmacs Development System Standard Configuration



6-83-1537

FIGURE 1-3. VME/10 Microcomputer System

1.4.1 Form of FORTRAN Compiler

The FORTRAN Compiler and its runtime libraries are provided on LARK cartridge, VERSAdos cartridge, or VERSAdos floppy diskette. Other software necessary for program development (such as VERSAdos, CRT Text Editor, and Linkage Editor) is available on other disks called system disks. The disks that contain the user's programs are called user's disks.

1.4.2 Program Development

To develop a program for the MC68000 using the FORTRAN Compiler, the following four steps are required:

a. Preparation of the FORTRAN source program (see Chapter 2).
   Prepare FORTRAN source programs on a user's disk using the CRT Text Editor.

b. Compilation of the program using the FORTRAN Compiler (see Chapter 3).
   The FORTRAN program is compiled using the FORTRAN Compiler to produce the object program.

c. Preparation of the load module (see Chapter 5).
   The input object program that was created in item b. utilizes the Linkage Editor to prepare a load module combining any object programs required.

d. Execution of the load module (see Chapter 6).
   Execute the load module created in item c.

Figure 1-4 illustrates the process of program development.

1.5 NOTATION

Commands and other input/output (I/O) are presented in this manual in a modified Backus-Naur Form (BNF). Certain symbols in the syntax may be used, where noted, in the real I/O. Others are meta-symbols, which are used for definition only and are not entered by the user. These meta-symbols and their meanings are as follows:

< > Angular brackets enclose a symbol, known as a syntactic variable, that is replaced in a command line by one of a class of symbols it represents.

| This symbol indicates that a choice is to be made. One of several symbols, separated by this symbol, should be selected.

[ ] Square brackets enclose a symbol that is optional. The enclosed symbol may occur zero or one time.

[ ]... Square brackets followed by periods enclose a symbol that is optional/repetitive. The symbol may appear zero or more times.

In the examples given in the following chapters, operator entries are to be followed by a carriage return unless otherwise specified. The carriage return is not shown in examples except where it is the only entry, in which case it is shown as (CR).

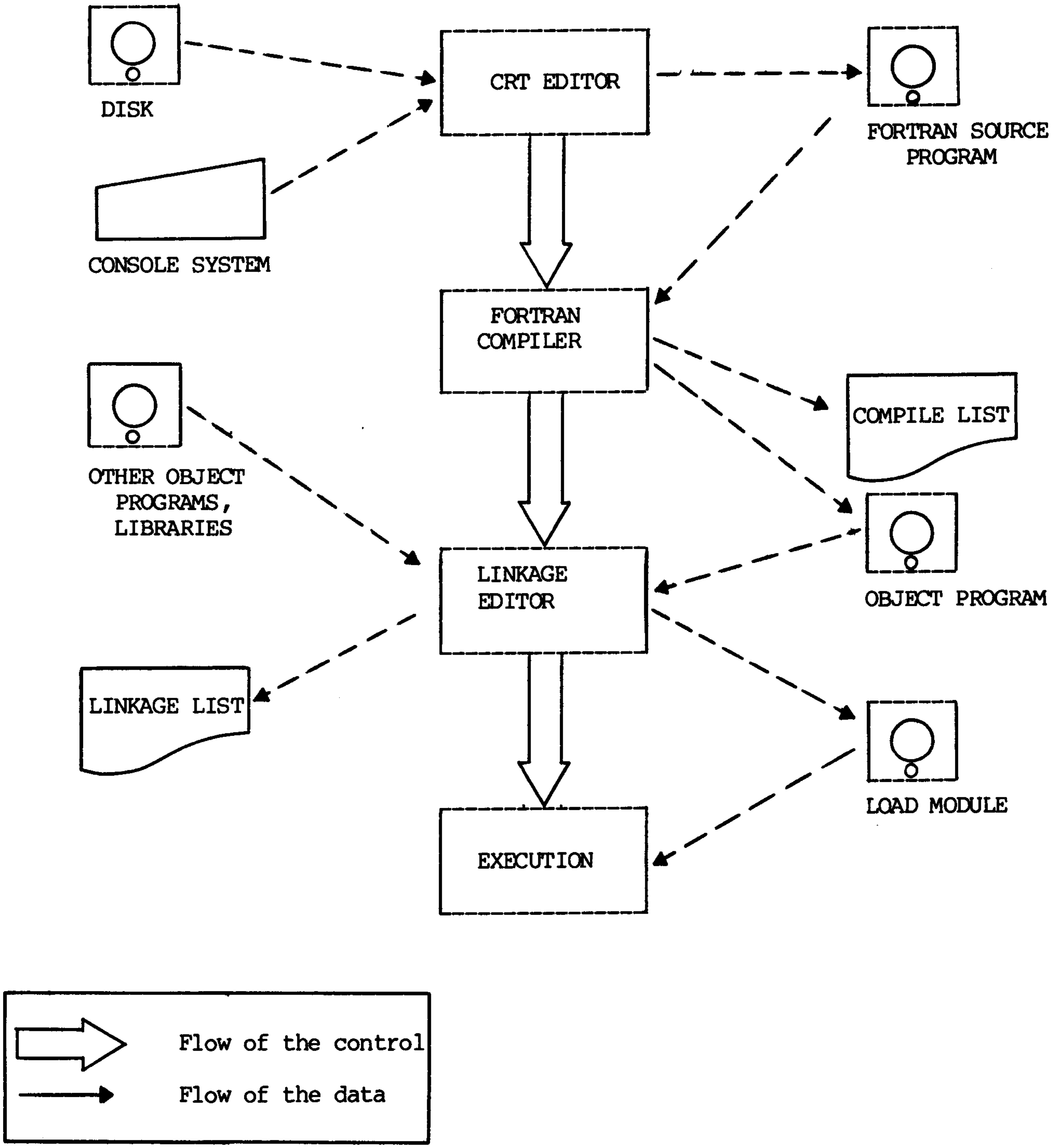


FIGURE 1-4. Process of Program Development

1.6 RELATED DOCUMENTS

Refer to the following documents for more information on the environments in which the M68000 Family FORTRAN Compiler is used.

M68000 Family VERSAdos System Facilities Reference Manual, M68KVSF

System Generation Facility User's Manual, M68KSYSGEN

VERSAdos Data Management Services and Program Loader User's Manual, RMS68KIO

M68000 Family Real-Time Multitasking Software User's Manual, M68KRMS68K

VERSAdos Overview, M68KVOVER

VERSAdos Reference Card, MVDOSCARD

M68000 Family Resident Structured Assembler Reference Manual, M68KMASM

M68000 Family Linkage Editor User's Manual, M68KLINK

M68000 CRT Text Editor User's Manual, M68KEDIT

VME/10 Text Editor User's Manual, M68KVSEDT

MVME110/-1 VMEmodule Monoboard Microcomputer User's Manual, MVME110

VERSAdos to VME Hardware and Software Configuration User's Manual, MVMEVDOS

Monoboard Microcomputer User's Guide, M68KVM01

VERSAmodule Monoboard Microcomputer User's Guide, M68KVM02

VME/10 Microcomputer System Overview Manual, M68KVSOM

VMC 68/2-Series Microcomputer System Manual, MVMCSM

EXORmacs Development System Operations Manual, M68KMACS

CHAPTER 2

PREPARATION OF FORTRAN SOURCE PROGRAMS

2.1 INTRODUCTION

FORTRAN source programs are prepared using the CRT Text Editor. This chapter describes the preparation of a simple program. Refer to the M68000 CRT Text Editor User's Manual or the VME/10 Text Editor User's Manual for further details concerning the CRT Text Editor.

2.1.1 Description of a Sample Program

The sample program used to describe the creation of a FORTRAN program takes five numerical values that are the input, searches for the greatest and smallest values, and then prints those.

2.1.2 Preparing a Source Program

a. The CRT Text Editor program resides on the system disk; the FORTRAN source program resides on the user's disk.

b. Enter the CRT Text Editor command (E) from the console - (in this example, VOL1 is the user disk volume name, and TEST is the source program file name). Use the Editor's option F for predefined tab sets.

c. Enter the source program starting at the beginning of each line, and perform a CR (carriage return) at the end of each line.

d. After keying in the source program, press the F1 key. The cursor will move to the prompt (>) in the lower portion of the screen. Enter QUIT to end the source program entry.

This process is illustrated in Figure 2-1.

```
      PROGRAM TEST
      INTEGER ARRAY(10)
      INTEGER MAX,MIN
C  INPUT VALUE SET
      DO 100 I=1,5
         READ(5,200) ARRAY(I)
200      FORMAT(I4)
100   CONTINUE
C  GET MAX, MIN
      MAX = ARRAY(1)
      MIN = ARRAY(1)
      DO 300 I = 2,5
         IF (MAX .LT.  ARRAY(I)) MAX = ARRAY(I)
         IF (MIN .GT.  ARRAY(I)) MIN = ARRAY(I)
300   CONTINUE
C  PRINT MAX, MIN
      WRITE (6,400) MAX, MIN
400   FORMAT(2I4)
      STOP
      END
```

Following program entry:

Type (F1) key

```
>QUIT(CR)
=
```

NOTES:

1. = means awaiting command.
2. > means awaiting program entry.
3. Each line is terminated with a carriage return (CR).

FIGURE 2-1. Preparation of a Source Program

CHAPTER 3

USING THE FORTRAN COMPILER

3.1 INVOKING THE FORTRAN COMPILER

The command line format for the FORTRAN Compiler is:

```
FORTRAN <source file>[,[<object file>][,<listing file>]][;<option>]
```

Only the <source file> is required. The default extension on the <source file> is SA. If the <object file> and/or <listing file> are not specified, they will default to the same file name as the <source file>, but with extensions of RO and LS, respectively.

Command line options are specified by placing the appropriate option letter(s), separated by commas, in the option field of the command line. To disable an option, a hyphen (-) must precede the option letter. Table 3-1 lists available Compiler options.

The following example will compile the source program created in Chapter 2, Figure 2-1.

```
FORTRAN VOL1:..TEST,,#PR
```

3.1.1 File Name Format

The general format of the file names that can be used by the FORTRAN command is as follows:

```
[[<volume name>]:[<user #>].[<catalog name>].]<file name>[.<extension name>]
```

where:

volume name — is a string which identifies the disk volume. It can be up to four characters in length, and the first character must be alphabetic (a-z). The volume name specified during logon is used as the default.

user # — is up to four digits in length. If this parameter is omitted, the user number specified during logon is used as the default.

catalog name — is a string of up to eight characters, and the first character must be alphabetic (a-z). The catalog name specified during logon is used as a default catalog name.

file name — is a string of up to eight characters, and the first character must be alphabetic (a-z).

extension name — is one or two characters or numbers which may be used to distinguish file names. When using an extension name, refer to paragraph 3.1.2.

Following are examples of the file name format. Refer to the VERSAdos System Facilities Reference Manual for further details concerning file names.

EXAMPLE 1: SYS1:1.CATLOG.FILEEX.KE

Volume Name	SYS1
User #	1
Catalog Name	CATLOG
File Name	FILEEX
Extension Name	KE

EXAMPLE 2: SYS5:..FILEEX2.NS

Volume Name	SYS5
User #	(default)
Catalog Name	(default)
File Name	FILEEX2
Extension Name	NS

EXAMPLE 3: FILEEX2.LO

Volume Name	(default)
User #	(default)
Catalog Name	(default)
File Name	FILEEX2
Extension Name	LO

EXAMPLE 4: FILEEX2 (source file)

Volume Name	(default)
User #	(default
Catalog Name	(default)
File Name	FILEEX2
Extension Name	SA (default)

EXAMPLE 5: FILEEX2 (object file)

Volume Name	(default)
User #	(default)
Catalog Name	(default)
File Name	FILEEX2
Extension Name	RO (default)

TABLE 3-1. Compiler Options

OPTION	ABBREV.	DEFAULT	DESCRIPTION
LIST	L	L	Prints the source listing.
-LIST	-L		Inhibits printing of source listing.
ASMCODE	A	-A	Prints the object-pseudo assembly listing.
-ASMCODE	-A		Inhibits printing of object-pseudo assembly listing.
SYMBOL	S	-S	Prints the symbol table.
-SYMBOL	-S		Inhibits printing of symbol table.
ERROR = 0	E=n	n=0	Prints all error messages.
ERROR = 1			Prints all error messages except warnings.
ERROR = 2			Prints only fatal error messages. A fatal error occurs when a table (e.g., symbol table) overflows.
PAGE	P	P	Prints page header.
-PAGE	-P		Inhibits printing of page header.
VERTICAL=n 5<=n<=999	V=n	n=60	When using PAGE, this option defines the number of lines (n) per page.
HORIZONTAL=n 40<=n<=132	H=n	n=132	Specifies number of characters per line.
TITLE = line	T=line	-	Specifies title for page header.
OBJECT	O	O	Outputs the object program.
-OBJECT	-O		Inhibits output of the object program.
BIG	B	-B	When the BIG option is specified, the code portion of the program unit is assumed to be larger than 32K bytes long, and forward branch instructions are generated accordingly.
	-B		In the default -B mode, the code portion of the program unit is assumed to be less than 32K bytes long, and more efficient branch instructions are generated accordingly.

TABLE 3-1. Compiler Options (cont'd)

OPTION	ABBREV.	DEFAULT	DESCRIPTION
MINI	M	-M	When the MINI option is specified, the code portion of the entire program (including the current program unit and all other programs units comprising the program) is assumed to occupy less than 32K bytes of memory. In this case, all subroutine and function references are handled more efficiently.
	-M		In the default -M mode, the code portion of the entire program is assumed to be larger than 32K bytes, and less efficient subroutine and function reference are generated accordingly.
SMALL COMMON	C	-C	When the SMALL COMMON option is specified, the common and SAVEd items for the entire program are assumed to occupy a total of less than 32K bytes. In this case, more efficient data will be generated in the object code.
	-C		In the default -C mode, the common and SAVEd items are assumed to occupy a total of more than 32K bytes, and less efficient data references are generated accordingly.

STORAGE	z=n[:s]	n=27
		s=8
			Specifies the amount of storage to be made available to the Compiler for its tables, storage areas, and stack. "n" specifies in K bytes the amount of space allocated for the Compiler's internal tables (default size is 27K). "s" specifies in K bytes the amount of space allocated for the Compiler stack (default size is 8K). Total Compiler size is 170K, for code, plus the sum of "n" and "s". Therefore, the default size is 205K. If "z=40:20" were specified, the size of the Compiler would be 230K (170K + 40K + 20K).
			If the stack size specified is not large enough, the Compiler aborts with a bus error. The user must increase the space allocated for the stack by assigning a larger value for "s".

TABLE 3-1. Compiler Options (cont'd)

OPTION	ABBREV.	DEFAULT	DESCRIPTION
			If the space allocated for the internal tables is too small, the Compiler aborts with an internal error message describing the problem. The user must increase the table size specified by assigning a larger value for "n". If too much space was specified, a smaller "n" value is recommended.

NOTES: 1. Options are separated on the command line by commas -- e.g.,

```
=FORTRAN FIX:77..ARRAY;A,S,H=80,T=SAMPLE HEADING
```

2. Use of the -LIST option also inhibits printing of the object-pseudo assembly listing -- i.e., -L, A is treated as -L,-A.

3.1.2 Examples of Invoking the FORTRAN Compiler

EXAMPLE 1: Source file name, object file name, and listing file name are all the same, using only the extension names SA, RO, and LS.

```
SOURCE --------> FORTRAN ------>  OBJECT
FILE                     |        FILE
SYS1:0.. TEXT.SA         |        SYS1:0..TEST.RO
                         |
                         +---->   LISTING
                                  FILE
                                  SYS1:0..TEST.LS
```

Enter the command:

```
FORTRAN SYS1:0..TEST
```

After execution of this command, the following files are created:

```
Relocatable object file name      SYS1:0..TEST.RO
Listing file name                 SYS1:0..TEST.LS
```

The above command would have the same result as the following command:

```
FORTRAN SYS1:0..TEST.SA,SYS1:0..TEST.RO,SYS1:0..TEST.LS
```

EXAMPLE 2: The listing is routed to the line printer.

```
SOURCE --------> FORTRAN ----->  OBJECT
FILE                     |       FILE
VOL1:0..SF1.SA           |       VOL1:0..SF1.RO
                         |
                         +--->   LISTING
                                 FILE
                                 Line Printer
```

Enter the command:

```
FORTRAN VOL1:0..SF1,,#PR
```

EXAMPLE 3: The listing is routed to the user's console.

```
SOURCE --------> FORTRAN -----> OBJECT
FILE                     |      FILE
VOL2:0..SF1.SA           |      VOL2:0..SF1.RO
                         |
                         +---> LISTING
                                FILE
                                Console System
```

Enter the command:

```
FORTRAN VOL2:0..SF1.SA,VOL2:1..OBJ,#
```

The # is used to specify that the listing should be directed to the user's console.

EXAMPLE 4: No listing is generated.

```
SOURCE --------> FORTRAN -----> OBJECT
FILE                            FILE
VOL3:100..NOLST.SA              VOL4:0..OBJ.RO
```

Enter the command:

```
FORTRAN VOL3:100..NOLST,VOL4:0..OBJ.RO,#NULL
```

When the #NULL is specified as the listing destination, no listing is provided.

NOTE

Each compilation can compile only one FORTRAN program unit (i.e., main routine, subroutine, or function). The linkage editor is responsible for combining the relocatable object modules for an entire program.

3.2 FILES FOR THE FORTRAN COMPILER

The FORTRAN Compiler uses the following files:

- Source file
- Object file
- List file

The FORTRAN command line specifies the source file, the object file, and the list file.

	File Name	Use	Volume Name	User Number
1.	User Defined	FORTRAN Source File	User Defined	User Defined
2.	User Defined	FORTRAN Object File	User Defined	User Defined
3.	User Defined	FORTRAN Listing File	User Defined	User Defined

CHAPTER 4

OUTPUT LISTINGS

4.1 FORTRAN COMPILER OUTPUT LISTINGS

The following listings are output to the listing file, depending upon which Compiler options are specified.

- Source listing
- Object-pseudo assembly language listing
- Symbol table listing
- Label table listing
- Statistical information listing
- Diagnostics listing

The header information shown at the top of each page appears below:

```
68000 FORTRAN <version #> <title> PAGE <nn>
```

version #	Version number
title	String specified by the TITLE option
nn	Page number

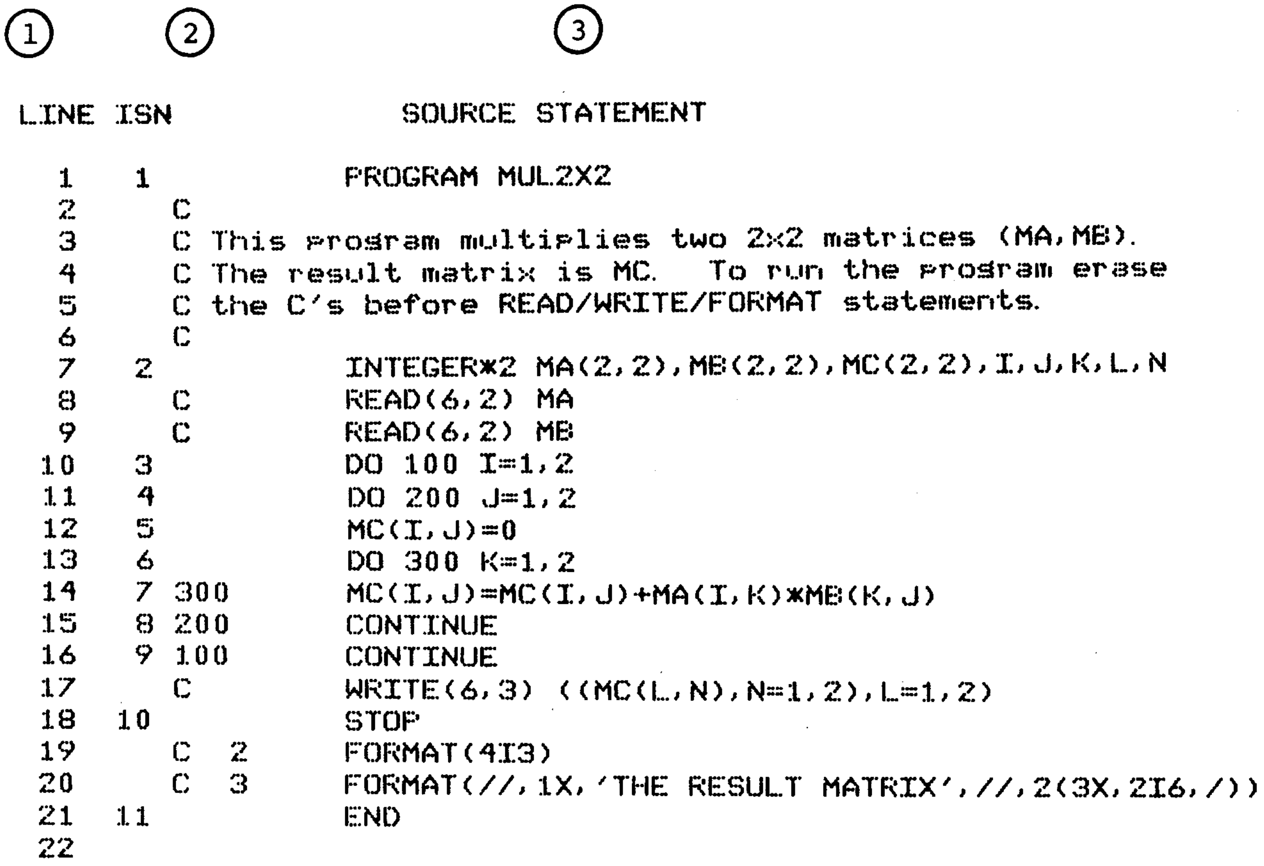
(if <title> is not specified, it is left blank)

4.2 SOURCE LISTING AND OBJECT-PSEUDO ASSEMBLY LISTING

This section describes the source and object-pseudo assembly listings produced under the control of compile time options.

4.2.1 Source Listing

The source listing lists the original source program, the source line numbers, and the internal statement numbers (ISN). Figure 4-1 illustrates the source listing with the L Compiler option.



```
LINE ISN            SOURCE STATEMENT

  1    1            PROGRAM MUL2X2
  2     C
  3     C This program multiplies two 2x2 matrices (MA,MB).
  4     C The result matrix is MC.  To run the program erase
  5     C the C's before READ/WRITE/FORMAT statements.
  6     C
  7    2            INTEGER*2 MA(2,2),MB(2,2),MC(2,2),I,J,K,L,N
  8     C           READ(6,2) MA
  9     C           READ(6,2) MB
 10    3            DO 100 I=1,2
 11    4            DO 200 J=1,2
 12    5            MC(I,J)=0
 13    6            DO 300 K=1,2
 14    7 300        MC(I,J)=MC(I,J)+MA(I,K)*MB(K,J)
 15    8 200        CONTINUE
 16    9 100        CONTINUE
 17     C           WRITE(6,3) ((MC(L,N),N=1,2),L=1,2)
 18   10            STOP
 19     C  2        FORMAT(4I3)
 20     C  3        FORMAT(//,1X,'THE RESULT MATRIX',//,2(3X,2I6,/))
 21   11            END
 22
```

NOTES

(1) Line number.
(2) Internal statement number.
(3) Source program.

FIGURE 4-1. Source Code Listing (L Option)

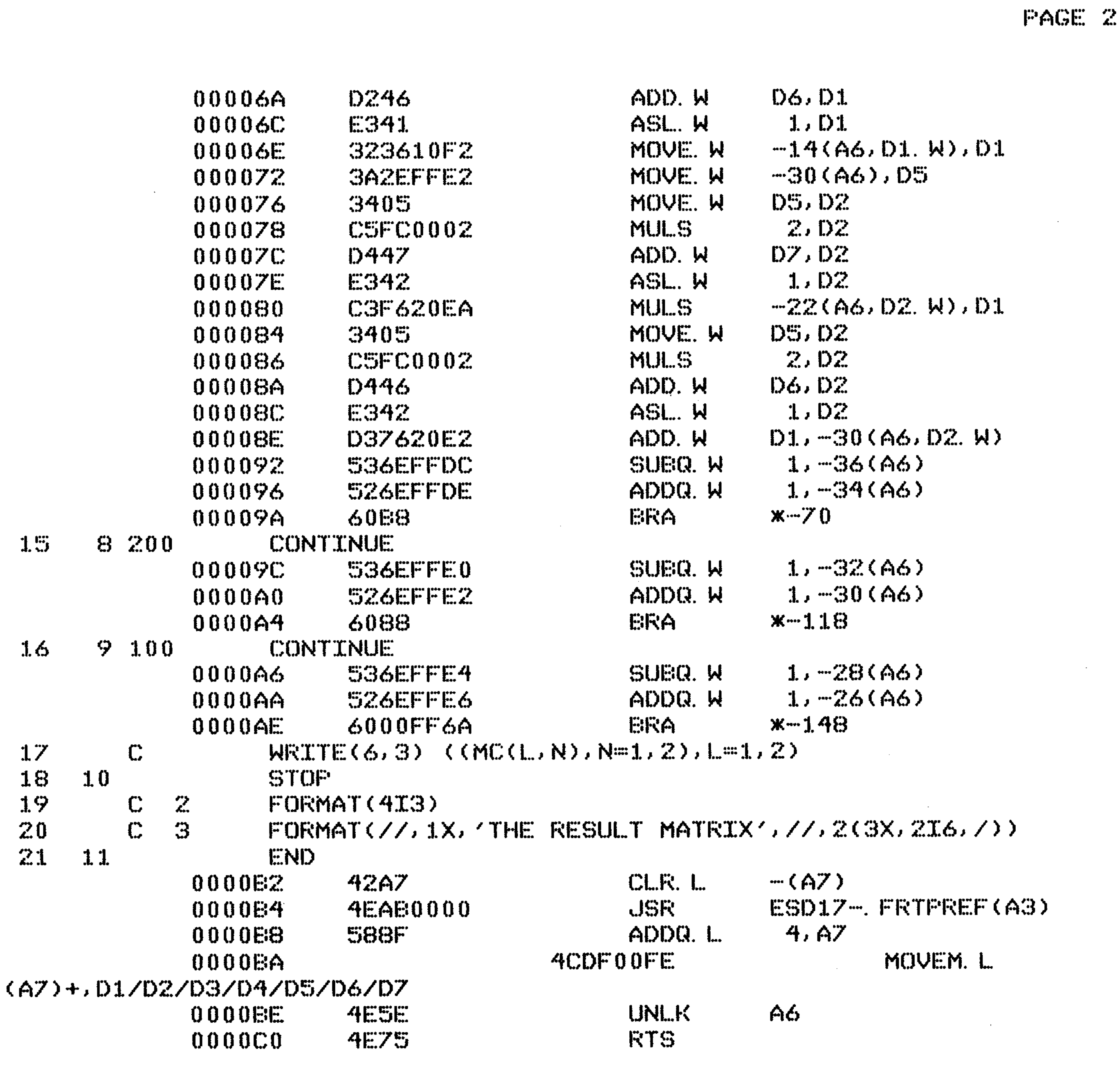
4.2.2 Object-Pseudo Assembly Listing

When the L and A Compiler options are selected, the Compiler prints the object-pseudo assembly listing corresponding to each group of source statements. Figure 4-2 illustrates an example of this.

```
LINE ISN             SOURCE STATEMENT

  1    1           PROGRAM MUL2X2
  2      C
  3      C This program multiplies two 2x2 matrices (MA,MB).
  4      C The result matrix is MC.  To run the program erase
  5      C the C's before READ/WRITE/FORMAT statements.
  6      C
  7    2           INTEGER*2 MA(2,2),MB(2,2),MC(2,2),I,J,K,L,N
  8      C         READ(6,2) MA
  9      C         READ(6,2) MB
 10    3           DO 100 I=1,2
            000000     2F0E                MOVE.L    A6,-(A7)
            000002     2C4F                MOVE.L    A7,A6
            000004     9FFC00000000        SUB.L      ****,A7
            00000A               48E77F00                     MOVEM.L
D1/D2/D3/D4/D5/D6/D7,-(A7)
            00000E     3D7C0001FFE6        MOVE.W     1,-26(A6)
            000014     3D7C0001FFE4        MOVE.W     1,-28(A6)
            00001A     4A6EFFE4            TST.W     -28(A6)
            00001E     6D000000            BLT       ***
 11    4           DO 200 J=1,2
            000022     3D7C0001FFE2        MOVE.W     1,-30(A6)
            000028     3D7C0001FFE0        MOVE.W     1,-32(A6)
            00002E     4A6EFFE0            TST.W     -32(A6)
            000032     6D000000            BLT       ***
 12    5           MC(I,J)=0
 13    6           DO 300 K=1,2
            000036     322EFFE2            MOVE.W    -30(A6),D1
            00003A     C3FC0002            MULS       2,D1
            00003E     D26EFFE6            ADD.W     -26(A6),D1
            000042     E341                ASL.W      1,D1
            000044     427610E2            CLR.W     -30(A6,D1.W)
            000048     3D7C0001FFDE        MOVE.W     1,-34(A6)
            00004E     3D7C0001FFDC        MOVE.W     1,-36(A6)
            000054     4A6EFFDC            TST.W     -36(A6)
            000058     6D000000            BLT       ***
 14    7 300       MC(I,J)=MC(I,J)+MA(I,K)*MB(K,J)
            00005C     3E2EFFDE            MOVE.W    -34(A6),D7
            000060     3207                MOVE.W    D7,D1
            000062     C3FC0002            MULS       2,D1
            000066     3C2EFFE6            MOVE.W    -26(A6),D6
```

FIGURE 4-2. Example of Output with Options L and A Specified (Sheet 1 of 2)



```
                                                                        PAGE 2

               00006A     D246                 ADD.W     D6,D1
               00006C     E341                 ASL.W      1,D1
               00006E     323610F2             MOVE.W    -14(A6,D1.W),D1
               000072     3A2EFFE2             MOVE.W    -30(A6),D5
               000076     3405                 MOVE.W    D5,D2
               000078     C5FC0002             MULS       2,D2
               00007C     D447                 ADD.W     D7,D2
               00007E     E342                 ASL.W      1,D2
               000080     C3F620EA             MULS      -22(A6,D2.W),D1
               000084     3405                 MOVE.W    D5,D2
               000086     C5FC0002             MULS       2,D2
               00008A     D446                 ADD.W     D6,D2
               00008C     E342                 ASL.W      1,D2
               00008E     D37620E2             ADD.W     D1,-30(A6,D2.W)
               000092     536EFFDC             SUBQ.W     1,-36(A6)
               000096     526EFFDE             ADDQ.W     1,-34(A6)
               00009A     60B8                 BRA       *-70
 15    8 200        CONTINUE
               00009C     536EFFE0             SUBQ.W     1,-32(A6)
               0000A0     526EFFE2             ADDQ.W     1,-30(A6)
               0000A4     6088                 BRA       *-118
 16    9 100        CONTINUE
               0000A6     536EFFE4             SUBQ.W     1,-28(A6)
               0000AA     526EFFE6             ADDQ.W     1,-26(A6)
               0000AE     6000FF6A             BRA       *-148
 17      C          WRITE(6,3) ((MC(L,N),N=1,2),L=1,2)
 18   10            STOP
 19      C  2       FORMAT(4I3)
 20      C  3       FORMAT(//,1X,'THE RESULT MATRIX',//,2(3X,2I6,/))
 21   11            END
               0000B2     42A7                 CLR.L     -(A7)
               0000B4     4EAB0000             JSR       ESD17-.FRTPREF(A3)
               0000B8     588F                 ADDQ.L     4,A7
               0000BA                 4CDF00FE                   MOVEM.L
(A7)+,D1/D2/D3/D4/D5/D6/D7
               0000BE     4E5E                 UNLK      A6
               0000C0     4E75                 RTS
```

FIGURE 4-2. Example of Output with Options L and A Specified (Sheet 2 of 2)

4.3 SYMBOL TABLE LISTING

The symbol table is a list of the symbolic names that exist in the source program. Figure 4-3 shows an example of the symbol table listing.

```
                                   SYMBOL TABLE

   ①          ②          ③      ④          ⑤         ⑥

 NAME       ATTR       ADDR   SIZE       TYPE     COMMON
C1     UNDEFINED     ********             C1
L1     SAVE.V              16             L1
```

NOTES

(1) Symbol names as they exist in the source program.

(2) Type of attribute (further details in Table 4-1, ATTR column).

(3) Address assigned to the symbol; if the space is blank, it is not applicable.

(4) Number of elements.

(5) Refer to Table 4-1 for this column.

(6) Name of common block to which symbol belongs.

FIGURE 4-3. Symbol Table Listing (Option S)

TABLE 4-1. Symbol Table Contents

COLUMN	INDICATION	MEANING
ATTR (attribute)	UNDEFINED. x	Attributes not determined
	LOCAL. x	Local variable
	COMMON. x	Common
	PROG	Program
	SUB	Subroutine
	AFDS	Function name
	INTFUNC	Intrinsic functions
	EXT	Externally declared subroutine
	FUNC	Function declared externally
	BLOCD	Block data name
	SAVE. x	SAVEd variable
	PARAMETER. x	Parameter variable
x indicates the following:		
	B	Common Block name
	V	Variable
	A	Array
SIZE	_	NUMBER OF ELEMENTS
TYPE	I2	2-byte Integer
	I4	4-byte Integer
	R4	4-byte Real
	R8	8-byte Real
	L4	4-byte Logical
	Cn	n-byte Character String n = 1 to 255

4.4 LABEL TABLE LISTING

Figure 4-4 provides an example of the label table listing.

```
 ①        ②        ③

LABEL    ATTR     ADDR
10       EXEC     003E
20       FRMT
```

NOTES

(1) The label.

(2) The type of statement specifying the label:

FRMT: Format label

EXEC: Execution statement label

(3) The relative address, from the beginning of the object module, for the executable statement labels.

FIGURE 4-4. Example of the Label Table Listing

4.5 MODULE INFORMATION LISTING

This listing displays the detected error numbers, the memory capacity that the object program requires, and the number of errors detected by the Compiler. Figure 4-5 is an example of one such listing.

```
             ①               ②            ③               ④

        CODE SIZE 293e,  SAVE SIZE 4, STACK SIZE 28, CONSTANT SIZE 220

                                Z=28 IS SUFFICIENT ⑥a
     ⑤ CURRENT Z=70

                               A LARGER VALUE IS RECOMMENDED ⑥b

        *****  TOTAL ERRORS 0   TOTAL WARNINGS 2

                      ⑦                  ⑧
```

NOTES

(1) The size of the object program (ROM).

(2) The number of bytes required for local static storage (SAVEd and initialized variables).

(3) The number of bytes required for local dynamic storage.

(4) The number of bytes required for format and other string constants.

(5) The value of Z used for this compilation -- i.e., Z=70:n.

(6) Either:

(a) The recommended size for Z.

(b) A larger value for Z is recommended; choose a large value for the next recompilation and then reduce it to the recommended value for future recompilations.

(7) The number of Level 1 or 2 diagnostic messages the Compiler detected. (Refer to Table 4-2 for the diagnostic message error level.)

(8) The number of Level 0 diagnostic messages the Compiler detected. (Refer to Table 4-2 for the diagnostic message error level.)

FIGURE 4-5. Module Information Listing

4.6 DIAGNOSTIC MESSAGES

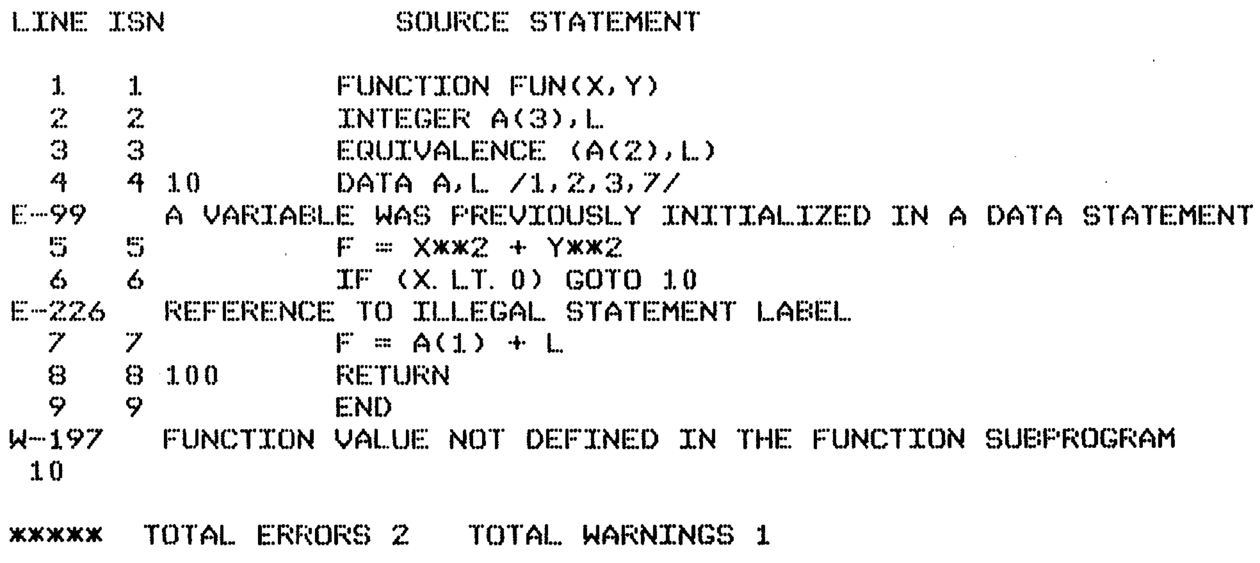
The Compiler outputs a diagnostic message when an error is detected in the source program. It outputs the diagnostic message as a possible form of warning to which it assigns an error level to distinguish severity. Table 4-2 displays the various levels of error messages and their implications.

TABLE 4-2. Diagnostic Message Error Levels

LEVEL	CATEGORY	MEANING
0 or W	Warning	It is possible that there is an error, but the program is acceptable.
1 or E	Normal Warning	Syntax or other error. The program is unacceptable and no object code will be generated, but the remainder of the program will be checked.
2 or F	Fatal Error	The error detected cannot be resolved by the Compiler, and the Compiler will abort without checking the remainder of the program.

The diagnostic message is printed right after the error is detected in the source listing.

Refer to Appendix A for further details concerning the diagnostic messages. Figure 4-6 provides an example of diagnostic messages.



```
LINE ISN              SOURCE STATEMENT

  1    1              FUNCTION FUN(X,Y)
  2    2              INTEGER A(3),L
  3    3              EQUIVALENCE (A(2),L)
  4    4 10           DATA A,L /1,2,3,7/
E-99     A VARIABLE WAS PREVIOUSLY INITIALIZED IN A DATA STATEMENT
  5    5              F = X**2 + Y**2
  6    6              IF (X.LT.0) GOTO 10
E-226    REFERENCE TO ILLEGAL STATEMENT LABEL
  7    7              F = A(1) + L
  8    8 100          RETURN
  9    9              END
W-197    FUNCTION VALUE NOT DEFINED IN THE FUNCTION SUBPROGRAM
 10

*****  TOTAL ERRORS 2   TOTAL WARNINGS 1
```

FIGURE 4-6. Diagnostic Message Example

CHAPTER 5

CREATION OF AN EXECUTABLE LOAD MODULE

5.1 INTRODUCTION

Relocatable object modules, generated by the FORTRAN Compiler, are processed by the M68000 Family Linkage Editor (referred to as the "linker") to produce an absolute load module. A FORTRAN program requires the linker because:

a. every FORTRAN program refers to runtime routines which reside in the System Library,

b. if a program consists of one or more subprograms which were compiled separately, the linkage between modules must be constructed, and

c. if a FORTRAN program calls a procedure or function written in assembly language, the load module must include object modules produced by the M68000 Assembler.

In all these cases, the linker is required to assign memory space to each required object module, enable intermodule communication, and create a load module that is ready to run.

FORTRAN programs are linked by the program LINK. LINK expects to find the FORTRAN runtime library FORTLIB.RO on the system volume under user number 0. By default, FORTRAN programs are linked to execute on a system hosting a Memory Management Unit (MMU). If the target system does not have an MMU (e.g., the VMC 68/2 or MVME110), then file FINITVM2.RO must be linked before the library is linked. An example of this activity may be seen in paragraph 5.4.

5.2 INVOKING THE LINKAGE EDITOR

Enter the following command from the system console to invoke the Linkage Editor:

```
LINK <f1>[/<f1>]...,[<f2>],[<f3>];[<options>]
```

f1 These are the object files produced by the FORTRAN Compiler. Up to 16 different object files can be specified by separating file names with a slash (/).

f2 This specifies the load module file name. If this is omitted, the same name as the first f1 is used with extension LO.

f3 This file is used for outputting linkage information that is produced by the Linkage Editor. #PR or # is usually specified. #PR indicates that the linkage information is routed to the line printer, and # indicates that the system console is the destination. If omitted, # will be used.

options This specifies the options for the Linkage Editor. Refer to the Linkage Editor User's Manual for further details on the options.

5.3 EXAMPLES

Following are some examples of load module generation.

EXAMPLE 1: Preparation of the load module when compiling with one source program.

```
SOURCE --------> FORTRAN ----->  LISTING
FILE                .     |      FILE
SYS1:O..TEST.SA     .     |      Line Printer
                    .     |
                    .     +--->  OBJECT          ------+
                    .            FILE                  |
                    .            SYS1:0..TEST.RO       |
                    .                                  |
                                                       |
                 LINKAGE  <----------------------------+
                 EDITOR
                    |
                    |---------> LOAD MODULE
                    |           SYS1:0..TEST.LO
                    |
                    |
                    +---------> LINKAGE LIST
```

Enter the FORTRAN command:

```
FORTRAN SYS1:..TEST,,#PR
```

which invokes the FORTRAN Compiler, inputs the source program from SYS1:0..TEST.SA, and prepares the object program SYS1:0..TEST.RO

then enter the LINK command:

```
LINK SYS1:..TEST,,#PR;L=SYS0:0..FORTLIB.RO
```

which invokes the Linkage Editor, inputs the object program from SYS1:0..TEST.RO, and creates the load module in SYS1:0..TEST.LO. The FORTRAN runtime library, FORTLIB.RO, is on volume SYS0:0. It need not be specified if SYS0 is the logon volume.

EXAMPLE 2: Preparation of the load module after compiling a source program and then linking together several relocatable object modules.

```
SOURCE --------> FORTRAN -----> LISTING
FILE                .     |      FILE
VOL2:0..MAIN.SA     .     |      Line Printer
                    .     |
                    .     +--->  OBJECT          ------+
                    .            FILE                  |
PREPARED            .            VOL2:0..MAIN.RO       |
OBJECT FILE         .                                  |
VOL2:0..SUB1.RO     .                                  |
    |               .                                  |
    |                                                  |
    +-------->   LINKAGE  <----------------------------+
                 EDITOR
    +-------->
    |               |---------> LOAD MODULE
    |               |           VOL2:0..MAIN.LO
PREPARED            |
OBJECT FILE         |
VOL2:0..SUB2.RO     +---------> LINKAGE LIST
                                Line Printer
```

Enter the FORTRAN command:

```
FORTRAN VOL2:0..MAIN,,#PR
```

which invokes the FORTRAN Compiler, inputs the source program VOL2:0..MAIN.SA, and prepares the object program VOL2:0..MAIN.RO

then enter the LINK command:

```
LINK VOL2:0..MAIN/VOL2:0..SUB1/VOL2:0..SUB2,,#PR;L=FORTLIB
```

which invokes the Linkage Editor, combines the object program compiled by the FORTRAN command and the two relocatable object programs, and creates the load module in VOL2:0..MAIN.LO

5.4 RUNTIME LIBRARIES FOR VERSAmodule SYSTEMS

The code produced by the FORTRAN Compiler is position-independent. The following commands create a load module from TEST.RO.

```
LINK ,TEST.LO,TEST,LL
SEG SEG0:7,15
SEG SEG1:8-10
IN  0..FINITVM2
IN  TEST.RO
```

5.5 FREEING A SEGMENT FOR A FORTRAN PROGRAM

5.5.1 Default Situation

By default, the FORTRAN Compiler uses the following segment allocation scheme (refer to paragraph 6.3.1 for definition of segments and sections):

```
SEG0 - Section 7  - Common blocks and SAVE parameters

SEG1 - Section 8  - Runtime routines
       Section 9  - FORTRAN program and subroutines, FORMAT statements
       Section 10 - Constant strings

SEG2 - Section 15 - Command line, stack area, and RMA block
```

At runtime, another segment is requested which is contiguous to the stack. This area is used for the stack and the parameter block areas associated with each file (logical unit). Therefore, all four segments are allocated.

5.5.2 Freeing A Segment

To free a segment, the Linkage Editor user commands must be used. The following example illustrates how program TEST would do this.

```
=LINK ,TEST.LO,TEST.LL
SEG SEG0(R):8-10
SEG SEG1:7,15
IN TEST.RO
END
```

To free more than one segment, all of the sections could be linked into one segment.

```
 SEG SEG0:7-10,15
```

This, however, does not prevent the code from being overwritten by an illegal array reference.

5.6 SHARING A SEGMENT BETWEEN TWO FORTRAN TASKS

5.6.1 Intertask Communication Through a Global Common

For several tasks to have access to a global FORTRAN common, the following steps must be taken.

a. All RO (relocatable object) modules referencing this global common must be patched. For instance, if there is a global common named GLOBAL in a FORTRAN program TEST, then TEST.RO must be patched. Using utility DUMP, dump TEST.RO and look at the first several sectors of this file. Within these sectors will be found the external symbol definition (ESD) for GLOBAL. Preceding GLOBAL will be $17 which specifies GLOBAL as a common in section 7. $17 must be changed to $1x, where x could be any section other than 7, 8, 9, 10, or 15. For this example, assume GLOBAL is to be in section 5. Therefore, $17 must be patched to $15.

b. Now a segment must be freed in the load module. See paragraph 5.5.2 for more information. The following example frees one segment, associates it with section 5 (GLOBAL), and declares this segment to be globally shareable.

```
=LINK ,TEST.LO,TEST.LL;B
SEG SHAR(G):5
SEG SEG1:8-10
SEG SEG2:7-15
IN TEST.RO
END
```

Now any other program which has been linked in a similar fashion will share common GLOBAL found in segment SHAR.

5.6.2 Sharing Program Segments

FORTRAN tasks can use a shared program segment. The shared routines must be assembly language routines, which also includes the FORTRAN runtime library.

The following examples explain how this shared segment can be created. The first example illustrates how two FORTRAN tasks can share runtime library routines. Note that if one runtime routine is shared, they must all be shared. This is because all runtime routines are located in section 8. In general, the routines located in those sections contained by the shared segment are also shared.

EXAMPLE 1: Sharing the FORTRAN runtime library.

The following Linkage Editor commands must be used to share the runtime library:

```
=LINK ,<load module file>,<listing file>;<options>
SEG SEG0:7-10
SEG SHAR(G):8
SEG SEG2:15
IN <RO modules>
<other linker commands>
END
```

No special options are required. Notice that a segment was freed by placing the FORTRAN code (section 9) in segment SEG0.

EXAMPLE 2: Sharing assembly language subroutines while not sharing the runtime library.

A call to a subroutine produces an XREF (external reference) for section 9. If the subroutine is an assembly language routine, it may be incorporated in a shared program segment. To do this, the XREF must be changed from section 9 to any other section except 7, 8, 10, or 15, as section 9 contains the code produced for the FORTRAN routine. To change this XREF, the RO module containing the XREF must be patched. For example:

```
PROGRAM MAIN
...
CALL SHARE(I)

XREF in section 9 for SHARE is produced.
```

In MAIN.RO, the XREF for SHARE will look like $69 (XREF in section 9), followed by SHARE. This can be found in the first few sectors of MAIN.RO. If SHARE is to be in section 11, $69 must be changed to $6B. In the assembly language source for SHARE, a SECTION 11 command is required. After this patching, the following linkage commands can occur:

```
=LINK ,<load module name>,<listing file name>;<options>
SEG SEG0:7-10
SEG SHAR(G):11
SEG SEG2:15
IN MAIN.RO
IN SHARE.RO
<other linker commands>
END
```

5.7 USEFUL EXTERNAL DEFINITIONS - XDEF

The registers passed by the SCT can be found at a 4-byte offset from .FZWRK. .FZWRK is an XDEF which is at the beginning of section 15.

CHAPTER 6

EXECUTION

6.1 EXECUTION OF THE PROGRAM

The FORTRAN load module can be executed by entering the following command:

```
    <command> [<f1>][[,<f2>][,...[,<fn>]]][,O=<device name>][;Z=n[:s]]
```

where:

command — is the load module file name.

f1...fn — is the file or device name(s) associated with logical unit number by position within list. Files are referenced by unit number within the program.

NOTE

> This is the only time at which an external file can be assigned to a logical unit. Within the FORTRAN code, the ANSI subset standard does not provide this feature. Thus, the first file is associated with logical unit 1, the second file with logical unit 2, and the nth file with logical unit n.

device name — is the file or device name specified as the recipient of program output.

Z=n[:s] — is the stack size and I/O buffer size.

6.1.1 Program Not Requiring Other Files

If a program does not require any other files, simply enter the program load module name and execute the program. Following is an example:

```
    =WORK:..TEST
```

Volume Name	WORK
User #	default (number specified during logon)
Catalog Name	default
File Name	TEST
Extension Name	LO (default)

By default, logical units 5 and 6 are assigned to the user's terminal.

6.1.2 Program Requiring Other Files

If a program does require another file, the program can be entered as in the following example:

```
=<command> WORK:..WORKFILE
```

Command	Load module file name
Volume Name	WORK
User #	default (number specified during logon)
Catalog Name	default
File Name	WORKFILE
Extension Name	FT (default)

When this command is entered, it executes using WORKFILE assigned to logical unit 1. The following options may be associated with each file.

(1) <file name> [(W)|(F(l[:m]))|(D(l[:m]))]

W — Overwrites the file that already exists.

Be cautious with this option because it will destroy existing records.

F(l[:m]) — Creates a new sequential file with the specified record length. Be sure not to specify an already existing file.

l — Specifies the record length.

m — Specifies the number of records (optional).

D(l[:m]) — Creates a new indexed sequential file with the specified record length. Be sure not to specify an already existing file.

l — Specifies the record length. The key length is always four bytes, leaving a data length of l-4.

m — Specifies the number of records (optional).

When an option is not specified, it defaults to the files as they already exist (if a file already exists). If a file is not allocated, it allocates a sequential file with variable-length records.

(2) O=<file name>|<device name>

Specifies the output file (logical unit 6) that the FORTRAN program uses -- in this case, the line printer.

(3) Z=n[:s] Specifies the stack size and the I/O buffer size.

s Specifies the size of the I/O buffer in n (the stack). If s is omitted, the I/O buffer size is n/2. If the Z option is omitted, it is assumed that the sizes are n = 32 (K) and s = 16 (K). When only n is specified, the stack size is n/2. If both n and s are specified, the stack size is n - s (K), with the area that is not included on the stack used as I/O buffer or the parameter area. The expression for the evaluation of s is: s >= L + 0.09J + 0.5 (K), where L is the largest record length in the file program and J is the number of units used.

Following are some examples of program execution.

EXAMPLE 1:

(a) Description of the example

This program adds two numerical values that are read from the console and writes the result to the console.

(b) Command line for the compilation of the program, its linkage, and its execution.

(i) Compilation

```
=FORTRAN WORK:..ADD,,#PR

    Volume Name          WORK
    User #               default
    Catalog Name         default
    File Name            ADD
    Extension Name       SA (default)
```

```
LINE     ISN     SOURCE STATEMENT

  1     00001          PROGRAM ADD
  2     00002          WRITE(6,50)
  3     00003     50   FORMAT(1X, 'INPUT DATA')                 ①
  4     00004          READ(5,100)  I,J                         ②
  5     00005    100   FORMAT (I4,1X,I4)
  6     00006          K=I+J
  7     00007          WRITE(6,200)K                            ③
  8     00008    200   FORMAT(1X ,'ADD RESULT = ',I6)
  9     00009          STOP
 10     00010          END
```

FIGURE 6-1. Result of Compilation for Example 1

(ii) Linkage

```
=LINK WORK:..ADD,,#PR;L=FORTLIB

    Volume Name            WORK
    User #                 default
    Catalog Name           default
    File Name              ADD
    Extension Name         RO (default)
```

```
Options in Effect:  -A, -B, -D, -H, -I, L, -M, O, -P, -Q, -R, -S, -U, -X


Unresolved References: None


Multiply Defined Symbols: None


Lengths (in bytes):

     Segment       Hex            Decimal

      SEG0      00000100              256
      SEG1      00004600            17920
      SEG2      00000400             1024
Total Length    00004B00            19200


   No Errors
   No Warnings


Load module has been created.
```

FIGURE 6-2. Linkage Result for Example 1

(iii) Execution

```
=WORK:..ADD                                        ④
```

Volume Name	WORK
User #	default
Catalog Name	default
File Name	ADD
Extension Name	LO (default)

Execute this module. After the input command is entered, wait for the system to respond.

Input Data:

```
    1000 2000                                       ⑤
```

Execution Result:

```
    ADD RESULT = 3000                               ⑥
    **FORTRAN STOP                                  ⑦
```

Explanation of numbered items:

This program uses two logical units (5 and 6), but since these are the units automatically assigned, the execution command requires only the load module (4) to execute.

(1) Write INPUT DATA to logical unit 6. This defaults to the console because the command line (4) does not specify an alternative.

(2) Read two integers from logical unit 5. This defaults to the console because the command line (4) does not specify an alternative.

NOTE

The FORTRAN subset does not support list-directed format statements. Therefore, the FORMAT statement must be adhered to.

(3) Write the result of the addition to console.

(4) Invoke program WORK:..ADD.

(5) Example of an input to READ statement (2).

(6) Result of adding 1000 to 2000 (3).

(7) This message is sent to the system console whenever the STOP instruction is executed.

EXAMPLE 2:

(a) Description of the example

This program outputs to a direct access file five numerical values that are read from the console. The odd-numbered record entries are added, and their sum is sent to the printer.

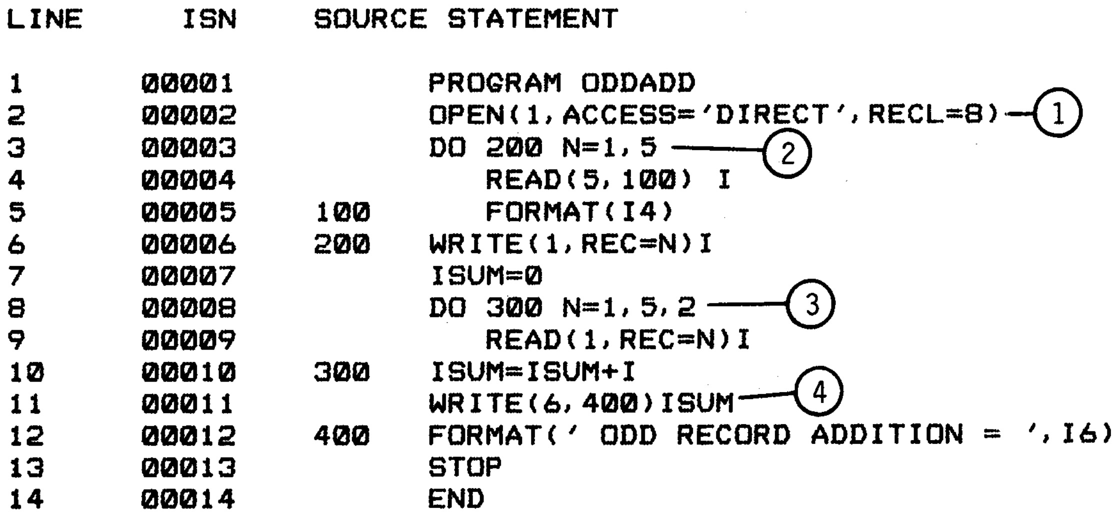
(b) Command line for the compilation of the program, its linkage, and its execution.

(i) Compilation

```
=FORTRAN WORK:..ODDADD,,#
```

Volume Name	WORK
User #	default
Catalog Name	default
File Name	ODDADD
Extension Name	SA (default)

The listing file in this example is sent to the user's terminal. The relocatable object module defaults to WORK:..ODDADD.RO.



```
LINE       ISN     SOURCE STATEMENT

1          00001            PROGRAM ODDADD
2          00002            OPEN(1,ACCESS='DIRECT',RECL=8)
3          00003            DO 200 N=1,5
4          00004               READ(5,100) I
5          00005      100      FORMAT(I4)
6          00006      200   WRITE(1,REC=N)I
7          00007            ISUM=0
8          00008            DO 300 N=1,5,2
9          00009               READ(1,REC=N)I
10         00010      300   ISUM=ISUM+I
11         00011            WRITE(6,400)ISUM
12         00012      400   FORMAT(' ODD RECORD ADDITION = ',I6)
13         00013            STOP
14         00014            END
```

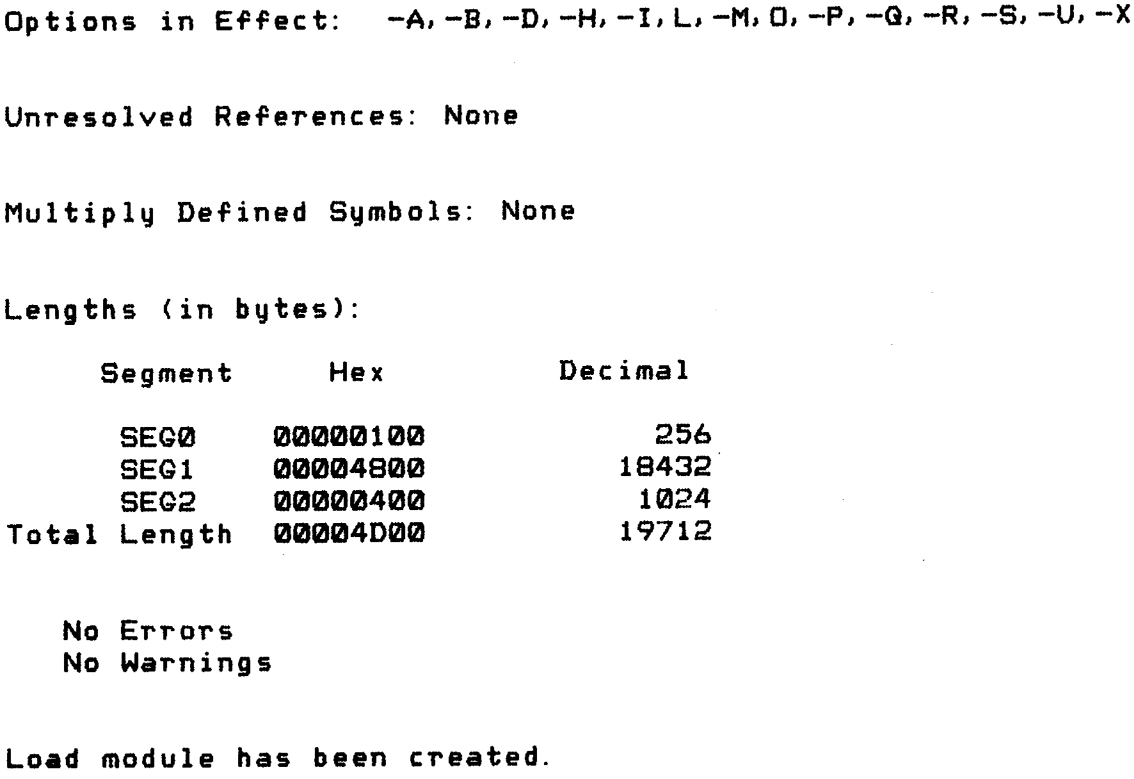
FIGURE 6-3. Result of Compilation for Example 2

(ii) Linkage

```
=LINK WORK:..ODDADD,,#;L=FORTLIB
```

Volume Name	WORK
User #	default
Catalog Name	default
File Name	ODDADD
Extension Name	RO (default)

It links the object file given above with the Library during execution. It also outputs the results to the user's console and creates the load module file with the extension name LO.



```
Options in Effect:  -A, -B, -D, -H, -I, L, -M, O, -P, -Q, -R, -S, -U, -X


Unresolved References: None


Multiply Defined Symbols: None


Lengths (in bytes):

     Segment      Hex          Decimal

      SEG0     00000100             256
      SEG1     00004800           18432
      SEG2     00000400            1024
Total Length   00004D00           19712


   No Errors
   No Warnings


Load module has been created.
```

FIGURE 6-4. Linkage Result for Example 2

(iii) Execution

```
=WORK:..ODDADD WORK:..F1(D(8)),O=#PR          ⑤
```

Volume Name	WORK
User #	default
Catalog Name	default
File Name	ODDADD
Extension Name	LO (default)

Execute this module. After the input command is entered, wait for the system to respond with a prompt for input data to be entered.

Input Data:

```
1000                                     ⑥
2000
3000
4000
5000
```

Execution Result:

```
ODD RECORD ADDITION = 9000               ⑦
**FORTRAN STOP
```

Explanation of the numbered items:

When invoking the program (5), logical units 1 and 6 are assigned to direct access file WORK:..F1 and the printer, respectively. Logical unit 5 defaults to the user's console.

(1) Open logical unit 1 for direct access.

(2) Read five data elements from the user's console and write the data to logical unit 1.

(3) Read odd data records.

(4) Output the addition result to logical unit 6.

(5) The execution command. Option D(8) means that the indexed sequential file has a key length of four bytes and a data length of four bytes for a total record length of eight bytes.

(6) The input data.

(7) The resulting output.

NOTE

> Because the file already exists, it must be deleted before the above command line is entered again. An alternative, when the file exists, is to invoke the program as:
>
> ```
> =WORK:..ODDADD WORK:..F1
> ```
>
> This results in use of the existing file rather than an attempt to allocate another.

EXAMPLE 3:

(a) Description of the example

This program copies the first five records in a direct access file to a sequential access file with unformatted records.

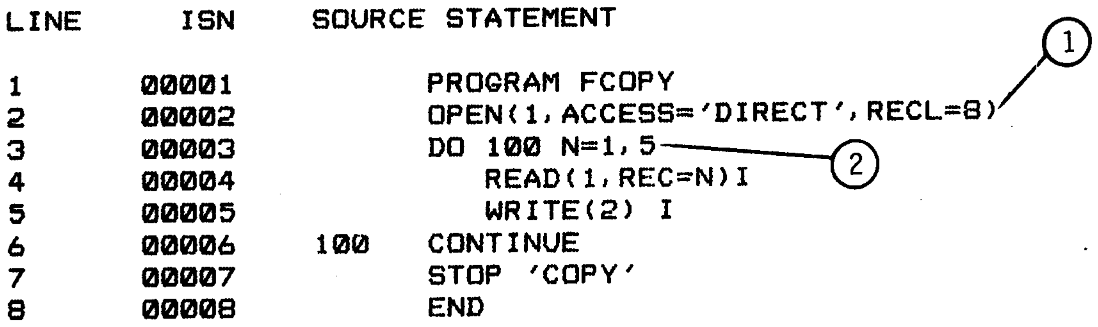
(b) Command line for the compilation of the program, its linkage, and its execution.

(i) Compilation

```
=FORTRAN WORK:..FCOPY,,#PR
```

Volume Name	WORK
User #	default
Catalog Name	unused
File Name	FCOPY
Extension Name	SA (default)

The listing file in this example is sent to the printer. The relocatable object module defaults to FCOPY.RO.



```
LINE      ISN     SOURCE STATEMENT

1        00001          PROGRAM FCOPY
2        00002          OPEN(1,ACCESS='DIRECT',RECL=8)
3        00003          DO 100 N=1,5
4        00004             READ(1,REC=N)I
5        00005             WRITE(2) I
6        00006    100   CONTINUE
7        00007          STOP 'COPY'
8        00008          END
```

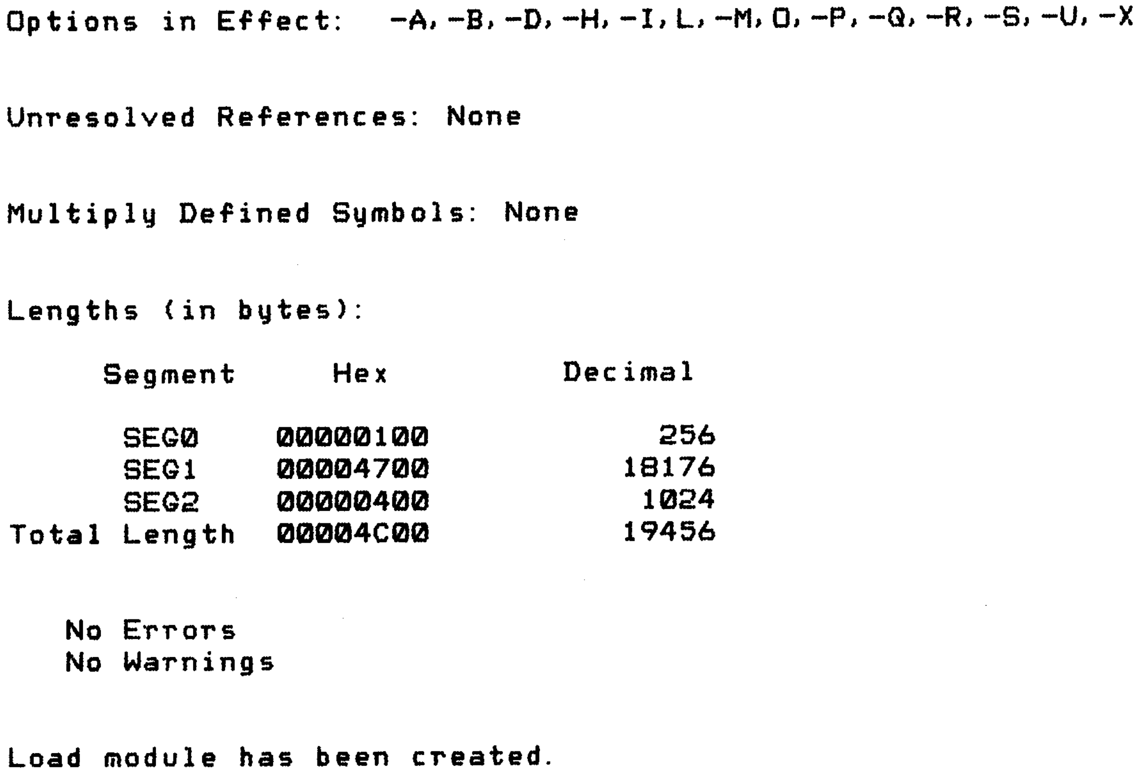
FIGURE 6-5. Result of Compilation for Example 3

(ii) Linkage

```
=LINK WORK:..FCOPY,,#PR;L=FORTLIB

    Volume Name            WORK
    User #                 default
    Catalog Name           default
    File Name              FCOPY
    Extension Name         RO (default)
```

It edits and combines the object file given above with the Library during execution. It also outputs the results to the printer and creates the load module file with the extension name LO.



```
Options in Effect:  -A, -B, -D, -H, -I, L, -M, O, -P, -Q, -R, -S, -U, -X


Unresolved References: None


Multiply Defined Symbols: None


Lengths (in bytes):

     Segment      Hex           Decimal

      SEG0     00000100            256
      SEG1     00004700          18176
      SEG2     00000400           1024
Total Length   00004C00          19456


   No Errors
   No Warnings


Load module has been created.
```

FIGURE 6-6. Linkage Result for Example 3

(iii) Execution

```
=WORK:..FCOPY WORK:..F1,WORK:..F2
```

Volume Name	WORK
User #	default
Catalog Name	unused
File Name	FCOPY
Extension Name	LO (default)

Execute this module.

Execution result:

```
**FORTRAN STOP  COPY                         ③
```

Explanation of the numbered items:

The command line for this program (3) assigns file F1 to logical unit 1, and file F2 to logical unit 2.

File F1 does not require a D(8) specification because it was created by another program. In fact, it would be illegal to specify D(8) in this case. File F2 is a sequential file with variable-length records.

(1) Open logical unit 1 for direct access.

(2) Read a record from logical unit 1 and write data to unit 2. This loop is repeated five times.

(3) Execute FORTRAN STOP statement and display the word COPY on the user's console upon completion.

6.2 FILES

6.2.1 File Formats

There are four kinds of VERSAdos file formats: a sequential file; a contiguous file; and indexed sequential files, with and without duplicate keys.

A sequential file has an optional record length and does not require contiguous sectors. A contiguous file has a record length of 256 and does require contiguous sectors. The indexed sequential file is a sequential file with keys for each record.

The sequential file contains ASCII data and can be read and written with FORTRAN input/output statements. FORTRAN direct access files contain ASCII or binary data and use indexed sequential files.

6.2.2 Record Formats

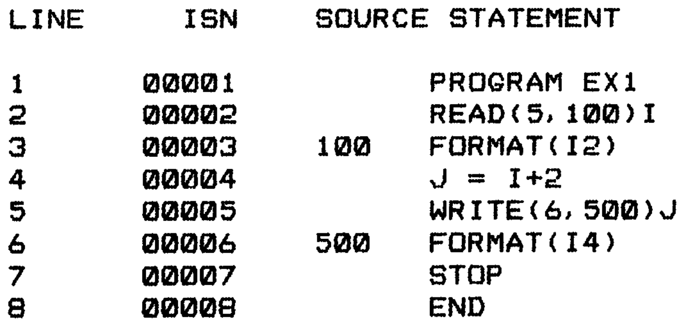
There are two kinds of record formats in FORTRAN: variable and fixed length records.

A variable length record has a length of 1 to 65,535 bytes. To read a record, the specified length of the record in the FORMAT statement must be equal to or smaller than the actual record length of the file. To write a record, these lengths must be equivalent. To rewrite a record with a WRITE statement, the fixed length record file is useful. A fixed length record has the same record length throughout a file. To create a new fixed length file, either F(l[:m])) or D(l[:m]) (refer to paragraph 6.1.2) must be specified as a file option on the command line.

6.2.3 File Access Methods

There are two methods for accessing files in FORTRAN: sequential and direct access.

Sequential access is the orderly access of one record at a time. The sequential access is able to use the READ, WRITE, BACKSPACE, REWIND, and ENDFILE statements. Logical units 5 and 6 are sequentially accessed.



```
LINE      ISN     SOURCE STATEMENT

1         00001          PROGRAM EX1
2         00002          READ(5,100)I
3         00003   100    FORMAT(I2)
4         00004          J = I+2
5         00005          WRITE(6,500)J
6         00006   500    FORMAT(I4)
7         00007          STOP
8         00008          END
```

FIGURE 6-7. Sequential Access; Input/Output

A direct access file is accessed with a specified record number. The READ and WRITE statements can be used with direct access files. The direct access file uses a fixed length record file that contains the record. The record number is specified by REC= in the READ and WRITE statements. The record length is specified for the data size that the WRITE statement outputs plus the key size of four bytes. The four bytes are used as an information area for direct access files. An example is listed below.

```
LINE      ISN     SOURCE STATEMENT

1         00001            PROGRAM DIRECT
2         00002            DIMENSION A(10),B(10)
3         00003            OPEN(1,ACCESS='DIRECT',RECL=44)
4         00004            WRITE(1,REC=1)A
5         00005            WRITE(1,REC=2)B
6         00006            STOP
7         00007            END
```

FIGURE 6-8. Direct Access I/O

In this example, the file associated with logical unit 1 has been specified as a direct access file with a record length of 44 bytes. Arrays A and B occupy 40 bytes each (four bytes per each real element). Therefore, it is possible to write the entire array A in record number 1 (line 4) and to write array B in record number 2 (line 5). If the OPEN statement were changed to OPEN (1, ACCESS = 'DIRECT', RECL = 40), then array B would be written to record number 3, because array A would occupy records 1 and 2.

6.2.4 Formatted and Unformatted I/O

The format of an input/output statement can be specified in FORTRAN in the following manner. Two examples are given to illustrate the two cases.

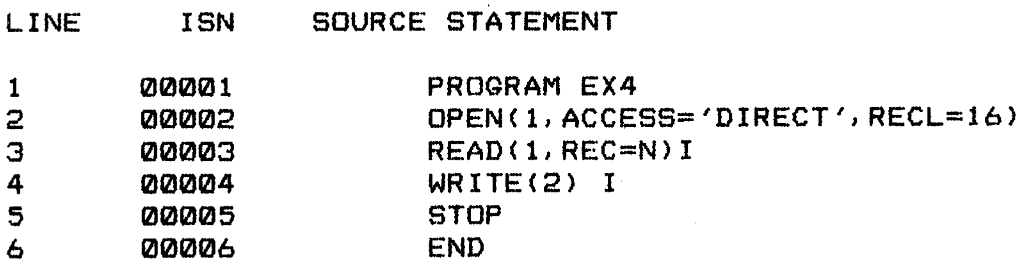
In the case of input/output with the FORMAT statement specified in a READ or WRITE statement, the record unit length is from a left parenthesis "(" to a right parenthesis ")" or from a slash "/" to another slash "/". In Figure 6-9, READ and WRITE statements (line 2 and line 5) have their format specified by FORMAT statements (line 3 and line 6, respectively) and execute their I/O accordingly.

```
LINE      ISN     SOURCE STATEMENT

1         00001            PROGRAM EX3
2         00002            READ(5,100)J
3         00003     100    FORMAT (I2)
4         00004            I=I+1000
5         00005            WRITE(6,500)I
6         00006     500    FORMAT (10H RESULT = ,I4,5X,3H***)
7         00007            STOP
8         00008            END
```

FIGURE 6-9. Input/Output with FORMAT

If the READ or WRITE statement does not specify a FORMAT statement, it executes in the default manner. The record size for the I/O is the data size plus four bytes (for a control area). If the record length of a file is not large enough, it inputs/outputs a multiple number of records.



```
LINE      ISN     SOURCE STATEMENT

1         00001          PROGRAM EX4
2         00002          OPEN(1,ACCESS='DIRECT',RECL=16)
3         00003          READ(1,REC=N)I
4         00004          WRITE(2) I
5         00005          STOP
6         00006          END
```

FIGURE 6-10. Input/Output without a FORMAT Statement

In this case, since the format is not specified, the WRITE statement (line 4) outputs the data read by the READ statement (line 3) without any modifications.

6.3 LOAD MODULE

6.3.1 Memory Organization

The FORTRAN Compiler produces relocatable object modules. These relocatable object modules consist of sections (the logical units into which code/data are placed), which are linked with other relocatable object modules to produce the load module. The basic unit of a load module is the segment.

TABLE 6-1. Memory Organization

SEGMENT NUMBER	SECTION NUMBER	CONTENTS
0	7	Common Block Variables (SAVE *) and SAVEd Variables
1	8	Runtime Routines
1	9	FORTRAN Program and Subroutines, FORMAT Statements
1	10	String Constants
2	15	Command Line, Stack Area, and RMA Block

(SAVE *): All common blocks are SAVEd. (Refer to ANSI Standard.)

6.4 FORTRAN STATEMENTS THAT CONTROL EXECUTION

This section explains the relationship between FORTRAN statements and execution.

6.4.1 PAUSE Statement

The PAUSE statement is used to stop execution momentarily. When this statement executes, it outputs a message to the user's console and stops the execution. It then waits for the return key before continuing with the execution of the next statement. If there is no user's console, this statement does nothing, and execution continues with the next statement.

```
**FORTRAN PAUSE   [<text>]
```

<text>, which can be any string enclosed in single quotes, is printed when the PAUSE statement is executed.

6.4.2 STOP Statement

The STOP statement is used to stop execution of the program. When this statement executes, it outputs a message to the user's console and stops the execution.

```
**FORTRAN STOP   [<text>]
```

<text>, which can be any string enclosed in single quotes, is printed when the STOP statement is executed.

Refer to the three execution examples in paragraph 6.1.

6.4.3 ENDFILE Statement

The ENDFILE statement writes an END OF RECORD to the file, but it does nothing in the operating system. Thus, it cannot delete a created record.

```
    .
    .
WRITE (1) A
WRITE (1) B
BACKSPACE 1
ENDFILE 1
    .
    .
```

FIGURE 6-12. ENDFILE Statement

The record remains on the file with the ENDFILE statement execution.

6.5 DEFAULT LOGICAL UNITS

Logical units 5 and 6 are always the read and write default logical units on VERSAdos systems. The remaining logical units are assigned by their position within the command line as was described in paragraph 6.1. If there is no command line, as in a SYSGENed environment, then the logical units are assigned as described in Chapter 8.

CHAPTER 7

INCLUSION OF ASSEMBLY ROUTINES

7.1 INTRODUCTION

A call to an assembly language routine from a FORTRAN program is handled like a call to a FORTRAN subroutine or function in a FORTRAN program, and execution continues with the next statement.

7.2 INTERFACE WITH EXTERNAL PROCEDURE (WITHOUT ARGUMENTS)

When calling an external procedure, the return address is placed on the stack.

```
                Stack

      (A7) -> Return Address
              ---------------------
              Pre-Call Top of
                  Stack
```

FIGURE 7-1. Stack Contents when Control is Passed to the Procedure

7.3 INTERFACE WITH EXTERNAL PROCEDURE (WITH ARGUMENTS)

When calling an external function or procedure into which it is necessary to transfer arguments, the address of the arguments is put onto the stack followed by the return address. If the called routine is a function, the result is returned to the calling routine in D0. If the format of the function is to return a double-length result (eight bytes), it is left on top of the stack when returning from the function call.

```
A = FUN(B,C,D)

         Stack

(A7) -> Return Address
        ---------------------
        Function Result
         (2, 4, or 8 bytes)
        ---------------------
        D Address (4 bytes)
        ---------------------
        C Address (4 bytes)
        ---------------------
        B Address (4 bytes)
        ---------------------
        Pre-Call Top of
            Stack            <--- highest address
```

FIGURE 7-2. Stack Contents when Control is Passed to a Procedure Requiring Arguments

Figure 7-2 shows the contents of the stack when control is transferred to function FUN. The stack pointer (A7) points to the return address. Immediately under the return address are two, four, or eight bytes for the function result. Next are the addresses of the actual parameters. Note that the address of the last parameter (D in the example) is immediately under the function result.

When control is passed back from the function, the stack pointer (A7) points below the return address. If it is a REAL*8 function, the result is on top of the stack; otherwise, the result is also put into D0. Adjusting the SP to point to the place it pointed to before executing the calling sequence is the responsibility of the caller.

For functions returning a LOGICAL result, written in assembly language, it is essential that the zero bit (Z) of the status register be properly set (or cleared) on return. This is achieved by loading D0 with the function result (0 or 1) immediately before return. Thus, the control returns to the instruction following the one which called the function, and data register D0 contains the result of the function.

7.4 EXAMPLE OF COMBINING ASSEMBLY ROUTINES

To combine FORTRAN and assembly language routines, the interface should be as illustrated in Figure 7-1 and Figure 7-2. Figures 7-3 and 7-4 give further examples.

For the code:

```
=FORTRAN WORK:..ASMLNK,,#PR;S,T= ASM LINK TEST
```

```
LINE     ISN     SOURCE STATEMENT

1        00001          PROGRAM ASMEX
2
3                C      Computes the time necessary to do 10,000 double precision multiplies.
4
5        00002          REAL*8 D1,D2,D3
6        00003          REAL AVE,TOTAL
7        00004          INTEGER START,STOP
8
9        00005          D2 = 1.765D0
10       00006          D3 = 3.45765D2
11       00007          CALL TIME(START) ———————— (1)
12       00008          DO 10 I=1,10000
13       00009    10       D1=D2*D3
14       00010          CALL TIME(STOP)
15       00011          TOTAL = (STOP-START)/1000.0
16       00012          AVE = TOTAL / 10000.0
17       00013          WRITE(6,20) TOTAL,AVE
18       00014    20    FORMAT(1X,'TIME IN SECONDS',/,
19                     1        ' TOTAL = ',F10.6 ,'  AVERAGE TIME = ',F10.8)
20       00015          STOP
21       00016          END
```

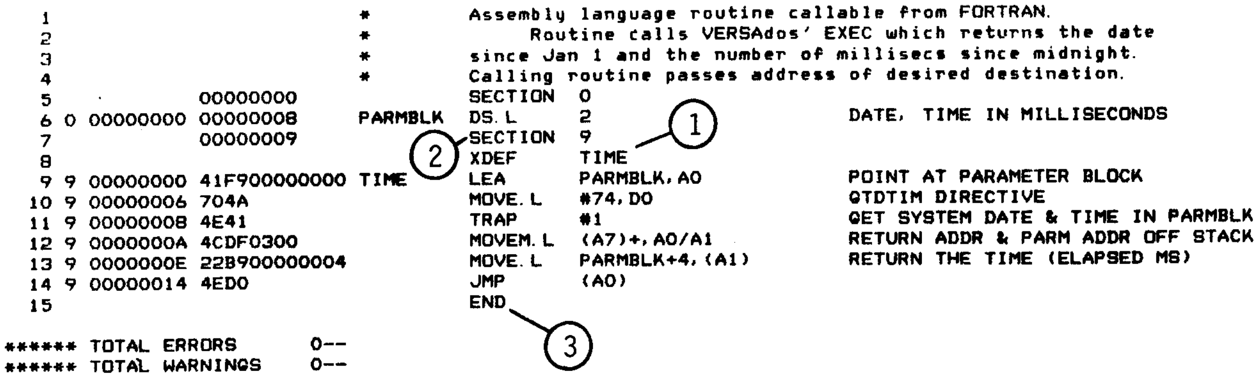
NOTE

(1) The call to an assembly language routine from a FORTRAN program is the same as a FORTRAN subroutine call.

FIGURE 7-3. FORTRAN Program Calling an Assembly Language Routine

For the code:

```
=ASM WORK:..TIME,,#PR;L
```



```
  1                               *       Assembly language routine callable from FORTRAN.
  2                               *            Routine calls VERSAdos' EXEC which returns the date
  3                               *       since Jan 1 and the number of millisecs since midnight.
  4                               *       Calling routine passes address of desired destination.
  5                00000000               SECTION   0
  6 0 00000000 00000008       PARMBLK     DS.L      2                     DATE, TIME IN MILLISECONDS
  7                00000009               SECTION   9
  8                                       XDEF      TIME
  9 9 00000000 41F900000000 TIME          LEA       PARMBLK,A0            POINT AT PARAMETER BLOCK
 10 9 00000006 704A                       MOVE.L    #74,D0                QTDTIM DIRECTIVE
 11 9 00000008 4E41                       TRAP      #1                    GET SYSTEM DATE & TIME IN PARMBLK
 12 9 0000000A 4CDF0300                   MOVEM.L   (A7)+,A0/A1           RETURN ADDR & PARM ADDR OFF STACK
 13 9 0000000E 22B900000004               MOVE.L    PARMBLK+4,(A1)        RETURN THE TIME (ELAPSED MS)
 14 9 00000014 4ED0                       JMP       (A0)
 15                                       END

****** TOTAL ERRORS       0--
****** TOTAL WARNINGS     0--
```

NOTES

(1) XDEF TIME defines to the outside world the name of this routine.

(2) FORTRAN routines are placed in section 9; this declaration allocates the assembly language routine to the same section. (See paragraph 6.3 for outline.)

(3) The subroutine/function entry point must not be included as a parameter on the END statement. Doing so will cause the resultant load module, after linking, to begin execution in the subroutine rather than in the main program.

FIGURE 7-4. Assembly Language Routine Callable from FORTRAN

The following LINK command is used to combine the two code segments into a load module:

```
=LINK WORK:..ASMLNK/WORK:..TIME,,#PR;L=FORTLIB
```

In this example, the LINK is the same as in other FORTRAN programs.

7.5 REGISTER USAGE IN FORTRAN PROGRAMS

The following registers are used in FORTRAN programs in the indicated manner. FORTRAN assumes that all registers are saved upon entry to a subroutine. Therefore, user-created assembly language subroutines must preserve the value of the following registers:

A3 - Base address of RTL routines.

A4 - Base address for dummy arguments of statement functions.

A5 - Base address of the (.FCBREF) SAVE and common variables.

A6 - Base address for local variables and parameters.

CHAPTER 8

FORTRAN'S RMA (RUNTIME MAINTENANCE AREA)

8.1 RUNTIME MAINTENANCE AREA

The Runtime Maintenance Area (RMA) is a data block which contains global information for the runtime routines. Each logical unit which is in use has a Unit Control Block (UCB) which contains a File Handling Services (FHS) block and an Input/Output Services (IOS) parameter block. The UCB's are in the RMA. Other information contained in the RMA would include format flags, and the beginning and the end of the free memory space.

In the default situation the RMA is located in the middle of the runtime allocated segment. Its address is contained in A5 throughout execution.

8.2 RMA LAYOUT

The following equates represent the layout of the RMA.

```
*
CMRGB     EQU     0          (LENGTH)    start of com-res.
*
OBJSP     EQU     CMRGB+0    (4)      STACK PTR. OF OBJECT REGS STORED
IOFLAGI   EQU     CMRGB+4    (2)      PARAM OF READ/WRITE
IOFLAG    EQU     CMRGB+5

QWRITE    EQU      0  <BIT>
QCOREF    EQU      1  <BIT>
QDFLTU    EQU      2  <BIT>           UNIT=*
QDIREC    EQU      3  <BIT>           REC=
QFORMT    EQU      4  <BIT>
QENDAD    EQU      5  <BIT>           END=
QERRAD    EQU      6  <BIT>           ERR=
QIOST     EQU      7  <BIT>           IOSTAT=

IOFLAG1   EQU     CMRGB+4
QHIOST    EQU      0  <BIT>           IOSTAT=INTEGER*2
UNITN     EQU     CMRGB+6    (2)      UNIT NUMBER
CFLINF    EQU     CMRGB+8    (6)      INTERNAL FILE
CFLINFS   EQU     CFLINF+0    (2)      SIZE
CFLINFA   EQU     CFLINF+2    (4)      ADDRESS
RECN      EQU     CMRGB+24   (4)      REC=               *D. A. *
ADREOF    EQU     CMRGB+16   (4)      END=
ADRERR    EQU     CMRGB+20   (4)      ERR=
ADIOST    EQU     CMRGB+24   (4)      IOSTAT=
*
*
ADCUCB    EQU     CMRGB+28   (4)      ADDR.  OF CURRENT UCB
ADTUCB    EQU     CMRGB+32   (4)      ADDR.  OF CRT UCB (6)
ADSUCB    EQU     CMRGB+36   (4)      ADDR.  OF LISTING UCB (6)
ADBBUF    EQU     CMRGB+40   (4)      BEGINNING OF BUFFER  whoes buffer???
ADEBUF    EQU     CMRGB+44   (4)      END OF BUFFER
ADCBUF    EQU     CMRGB+48   (4)      CURRENT OF BUFFER    of current ucb???
ADESPC    EQU     CMRGB+52   (4)      END OF SPACE    top of i/o-buffer.
ADBFSP    EQU     CMRGB+56   (4)      BEGINNING OF FREE SPACE in i/o-buff.
*
*
ADBFMT    EQU     CMRGB+60   (4)      BEGINNING OF FORMAT
ADCFMT    EQU     CMRGB+64   (4)      CURRENT OF FORMAT
ADLPN1    EQU     CMRGB+68   (4)      ADDR. OF FIRST LEVEL LEFT PAREN.
ADLPN2    EQU     CMRGB+72   (4)      ADDR  OF SECOND LEVEL LEFT PAREN.
REPN1     EQU     CMRGB+76   (2)      REPEAT SPECIFICATION OF OUTER LOOP
REPN2     EQU     CMRGB+78   (2)      REPEAT SPECIFICATION OF INNER LOOP
REPN3     EQU     CMRGB+80   (2)      DUPLICATION COUNTER
SCALF     EQU     CMRGB+82   (2)      SCALE FACTOR
BNIND     EQU     CMRGB+84   (1)      FORMAT BN INDICATER
EFORMT    EQU     CMRGB+85   (1)      END OF FORMAT IND.
```

```
SGNSCN   EQU      CMRGB+86   (1)      SIGN SCANED
QMINUS   EQU       $FF                MINUS
QPLUS    EQU       $F0                PLUS
FMTDLM   EQU      CMRGB+87   (1)      DELIMITER SCANED
PRDSCN   EQU      CMRGB+88   (1)      PRIOD SCANED
EXPSCN   EQU      CMRGB+89   (1)      FORMAT SCANED
*
*
FMTINF   EQU      CMRGB+91   (7)      FORMAT INFORMATION
FMTCOD   EQU      FMTINF+0    (1)      FORMAT CODE
QFI      EQU      4                   I
QFD      EQU      8                   D
QFE      EQU      12                  E
QFF      EQU      16                  F
QFG      EQU      20                  G
QFL      EQU      24                  L
QFA      EQU      28                  A
QFZ      EQU      32                  Z
QFAA     EQU      36                  A (NO WID)
FMTWID   EQU      FMTINF+1    (2)      FORMAT WIDTH
FMTDIG   EQU      FMTINF+3    (2)
FMTEXP   EQU      FMTINF+5    (2)     EXP. PART DIGITS
*
DATINF   EQU      CMRGB+98   (6)      INFORMATION OF I/O LIST
DATADR   EQU      DATINF+0    (4)      ADDR. OF I/O LIST
DATLEN   EQU      DATINF+4    (1)      LENGTH OF ELEMENT
DATTYP   EQU      DATINF+5    (1)      TYPE
QTI      EQU       0  <BIT>             INTEGER
QTR      EQU       1  <BIT>             REAL
QTL      EQU       2  <BIT>             LOGICAL
QTC      EQU       3  <BIT>             CHAR
*
*
ADRTRN   EQU      CMRGB+104 (4)       ADDR.  OF . FIFMT/. FINFT
ADRACC   EQU      CMRGB+108 (4)       ADDR.  OF . FISEQ/. FIDIR/. FICFL
ADRCNV   EQU      CMRGB+112 (4)       ADDR.  OF . FICVI/. FICVO
*
RCONT    EQU      CMRGB+116 (2)       RECORD COUNTER OF UNFORMATTED I/O
DATRLEN  EQU      CMRGB+118 (2)       REMAINING BYTES OF UNFORMATTED I/O
*                                        SET TO 0 BY FIINT
*                                        SET TO DATLEN BY FILST
*                                        RESET TO REMAINING BY FINFT
*
*
ERRNUM   EQU      CMRGB+120 (2)       ERROR NUMBER
ERRINF   EQU      CMRGB+122 (12)      ERROR INFORMATION
ERRINF1  EQU      ERRINF+1    (1)      LENGTH OF CHAR.
ERRINF3  EQU      ERRINF+2    (1)      LENGTH OF DATA (HEX)
ERRINF2  EQU      ERRINF+4    (4)      ADDR.  OF CHAR.
ERRINF4  EQU      ERRINF+8    (4)      ADDR.  OF DATA (HEX)
*
*
UCBLEN   EQU      CMRGB+136 (4)     LENGTH OF UCB  contant.
RECFLG   EQU      CMRGB+134 (2)
*
IOKIND   EQU      CMRGB+140       (1)
QCKI     EQU       1
QCKL     EQU       2
QCKF     EQU       3
* FIXED BUG: SEE LAST PARAGRAPH OF  STANDARD 13.3
CONREP   EQU        CMRGB+141    (1)      FLAG: RESET AT INITIALIZATION AND WHEN
*                                         AN OUTER '(' IS ENCOUNTERED; SET WHEN
*                                         A REPEATABLE EDIT DESCRIPTOR IS
*                                         ENCOUNTERED. MEANS THAT CURRENT PORTION
*                                         OF FORMAT SPEC MAY BE REUSED
REPEAT   EQU        CMRGB+142    (2)      REPEAT SPECIFICATION OF CURRENT
*                                         POTENTIAL REUSABLE PORTION OF FORMAT
*                                         SPEC. INITIALLY SET TO 1(ALL FORMAT IS
*                                         REUSED); AFTER OUTER '(', IS ASSIGNED
*                                         THE SAME VALUE AS REPN1(BUT IS NOT
*                                         DECREMENTED LIKE IT).
*FREE 144-159
CMRGE    EQU      CMRGB+160
*
```

```
*
ZERO      EQU       0
ONE       EQU       1
TWO       EQU       2
THREE     EQU       3
FOUR      EQU       4
FIVE      EQU       5
SIX       EQU       6
SEVEN     EQU       7
EIGHT     EQU       8
NINE      EQU       9
TEN       EQU       10
C99       EQU       99
C132      EQU       132
C255      EQU       255
*
*    PARAMETER TO .FISEQ/.FIDIR/.FICFL
*              D1
QINIT     EQU       0       INITIAL.I/O CALL
QNEXT     EQU       4       NEXT RECORD I/O CALL
QFINL     EQU       8       FINAL I/O CALL
*
*    RETURN CODE FROM .FISEQ/.FIDIR/.FICFL
*              D0
QNRM      EQU       0       NORMAL
QEOF      EQU       1       END OF FILE
QERR      EQU       2       ERROR OCCURED
*
*    RETURN CODE FROM .FIUBA
*              D0
QREADY    EQU       0       ALREADY OPENED
QFOUND    EQU       1       NOT OPENED, BUT UCB FOUND
QCREAT    EQU       2       UCB NOT FOUND, CREATE.
*
*    RETURN CODE FROM .FIFMT/.FINFT
*
QCMPLT    EQU       0       PROCESSING RECORD COMPLETED
QUNCPT    EQU       1       PROCESSING RECORD NOT COMPLETED
*
*
*    PARAMETER OF GET SEGMENT AND RECEIVE SEGMENT ATTRIBUTE
*
SEGMPB    EQU      0
TASKN     EQU      0  (4)     TASK NAME
SESSN     EQU      4  (4)     SESSION NAME
DIROPT    EQU      8  (2)     DIRECTIVE OPTION
SEGATT    EQU      10 (2)     SEGMENT ATTRIBUTE
SEGNAM    EQU      12 (4)     SEGMENT NAME
LOGADR    EQU      16 (4)     LOGICAL ADDRESS
SEGLEN    EQU      20 (4)     SGMENT LENGTH                      #140
RETADR    EQU      24 (4)     RECEIVE AREA ADDR.  OF RECEIVE SEG. ATT. #140
*                                                                #140
*    RECEIVE AREA OF RET.  SEG.  ATT.                            #140
*                                                                #140
RSASN     EQU      0  (4)     SEGMENT NAME                       #140
RSASA     EQU      4  (2)     SEGMENT ATTRIBUTE                  #140
RSABA     EQU      6  (4)     BEGINNING ADDR.                    #140
RSAEA     EQU      10 (4)     ENDING ADDR.                       #140
RSAFA     EQU      12 (4)     PHISICAL ADDR.                     #140
*                                                                #140

*
*******    RMA EQUS    **************************************    #140
*                                                                #140
RMA       EQU      0          BASED(A5)      start of i/o-buff   #140
RMASC7    EQU      40         ADDR(.FESC7)  main-program LOCALS.   #140
RMAFMA    EQU      44         ADDR(.FMAIN)  main-program's addr    #140
RMASC6    EQU      48         ADDR(.FESC6)  main-program SAVE+COMMON#140
RMACML    EQU      52         ADDR(.FCOML)  command-line (from mainprog) #140
RMAEFS    EQU      56         ADDR(END OF FREE SPACE) top of i/o-buff.   #140
RMAORG    EQU      60         ADDR(SAVE REGS OF OS)  in .FZWORK  #140
RMAEND    EQU      64     END=ADDR(CMRG)  followed by COM-REGION.    #140
*
*    ERROR NUMBER
*
```

```
E101     EQU      101    RECURSIVE CALL
E102     EQU      102    UNIT NO.  OUT OF RANGE
E103     EQU      103    END OF RECORD
E104     EQU      104    FORMAT CODE MISSING
E105     EQU      105    INVALID CHARACTER IN FORMAT
E106     EQU      106    NEST OUT OF RANGE IN FORMAT
E107     EQU      107    NUMBER OUT OF RANGE IN FORMAT
E108     EQU      108    ILLEGAL DISCRIPTORS IN FORMAT
E109     EQU      109    ILLEGAL SIGN WITHOUT SCAL FACTOR
E110     EQU      110    INVALID DECIMAL CHARACTER
E111     EQU      111    INVALID CHARACTER
E112     EQU      112    INVALID HEXADECIMAL CHARACTER
E113     EQU      113    FIXED OVERFLOW
E114     EQU      114    FLOATING OVERFLOW
E115     EQU      115    FLOATING UNDERFLOW
E116     EQU      116    NOT ENOUGH RECORDS UNFORMATED READ
E117     EQU      117    TOO MANY RECORDS UNFORMATED WRITE
E118     EQU      118    ASSIGN MISSING
E119     EQU      119    INSUFFICIENT MEMORY FOR BUFFER
E120     EQU      120    ERROR RETURN ON FHS
E121     EQU      121    READ NOT SUPORTED DEVICE
E122     EQU      122    WRITE NOT SUPORTED DEVICE
E123     EQU      123    BACKSPACE NOT SUPORTED DEVICE
E124     EQU      124    REWIND NOT SUPORTED DEVICE
E125     EQU      125    ENDFILE NOT SUPORTED DEVICE
E126     EQU      126    DIRECT ACCESS NOT SUPORTED
E127     EQU      127    SEQUENTIAL ACCESS NOT SUPORTED
E128     EQU      128    UNFORMATED NOT SUPORTED
E129     EQU      129    ILLEGAL DIRECT WITHOUT OPEN STMT.
E130     EQU      130    ALREADY ACCESSED DIRECT
E131     EQU      131    ALREADY ACCESSED SEQUENTIAL
E132     EQU      132    I/O ERROR AT SEQUENTIAL ACCESS
E133     EQU      133    I/O ERROR AT DIRECT ACCESS
E134     EQU      134    END OF FILE
E135     EQU      135    ALREADY OPENED
E136     EQU      136    ILLEGAL RECORD FORMAT
E137     EQU      137    RECORD LENGTH OF OPEN STMT GT OF FILE
E138     EQU      138    RECORD NUMBER LE 0
E139     EQU      139    NO FORMATTING FILE
E140     EQU      140    I/O ERROR AT PAUSE OR STOP
E141     EQU      141    ERROR OCCURED AT CLOSE
E142     EQU      142    OUT OF RANGE OF ARRAY ELEMENT
E143     EQU      143    INSUFFICIENT MEMORY
E144     EQU      144    ZERO DIVIDE
E145     EQU      145
E146     EQU      146
E147     EQU      147
E148     EQU      148
E149     EQU      149
E150     EQU      150
*
         LIST
```

8.3 UCB LAYOUT

The following equates represent the memory layout for each UCB. The basic format is an overhead data block followed by the IOS parameter block and then the FHS parameter block. Detailed information of the IOS and the FHS parameter blocks are contained in the VERSAdos Data Management Services and Program Loader User's Manual.

```
*...UCB

UCB       EQU     0
UCBNEXT   EQU     UCB+0   (4)    ADDR. OF NEXT UCB
UCBLUN    EQU     UCB+4   (1)    LOGICAL UNIT NO.
OPNFLG    EQU     UCB+5   (1)    0 -closed, QOPEN - opened.
QOPEN     EQU      1             OPENED
ACSFLG    EQU     UCB+6   (1)
QSEQ      EQU      1             SEQUENTIAL ACCESS
QDIR      EQU      2             DIRECT ACCESS
UCBFDCD   EQU     UCB+7   (1)    FILE/DEVICE CODE (FROM FHS)
QCONTIG   EQU      0             CONTIGUOS FILE
QSEQUEN   EQU      1             SEQUENTIAL FILE
QISEQND   EQU      2             INDEXED SEQUENTIAL FILE (NO DUPLICATE KEY
QISEQDK   EQU      3             INDEXED SEQUENTIAL FILE (DUP. KEYS ALLOWD
QTRMNLI   EQU      30            INTERACTIVE TERMINAL ON IPC INTERFACE
QTRMNLL   EQU      35            INTERACTIVE TERMINAL ON LOCAL DRIVER
QDISCFP   EQU      40            5/10 MB DISC, FIXED PLATTER
QDISCRP   EQU      41            5/10 MB DISC, REMOVABLE PLATTER
QFLPYSS   EQU      50            FLOPPY, SINGLE DENSITY SINGLE SIDED
QFLPYSD   EQU      51            FLOPPY, SINGLE DENSITY DOUBLE SIDED
QFLPYDD   EQU      52            FLOPPY, DOUBLE DENSITY DOUBLE SIDED
QMGTAPE   EQU      60            MAGNETIC TAPE
QLLPI     EQU      90            LOW SPEED LP ON IPC
QHLPI     EQU      91            HIGH SPEED LP ON IPC
QLLPL     EQU      95            LOW SPEED LP ON LOCAL DRIVER
QASYCOM   EQU      100           ASYNCHRONOUS COMM. LINE
QNULLD    EQU      255           NULL DEVICE
UCBDATTW  EQU     UCB+8   (2)  DEVICE ATTRIBUTES WORD (FROM FHS)
UCBDATT   EQU     UCB+9
QREADAT   EQU      0  <BIT>      SUPPORTS READ
QWRITAT   EQU      1  <BIT>      SUPPORTS WRITE
QBINRAT   EQU      2  <BIT>      SUPPORTS BINARY
QRANDAT   EQU      3  <BIT>      SUPPORTS RANDOM
QIMAGAT   EQU      4  <BIT>      SUPPORTS IMAGE
QHALTAT   EQU      5  <BIT>      SUPPORTS HALT I/O
QPOSTAT   EQU      6  <BIT>      SUPPORTS POSITION RECORD
QFILMAT   EQU      7  <BIT>      SUPPORTS FILEMARK
UCBDATT1  EQU     UCB+8
QINTRAC   EQU      0 . BIT>          INTERACTIVE DEVICE
UCBRECL   EQU     UCB+10 (2)     RECORD LENGTH (FROM FHS)
PREIO     EQU     UCB+12  (1)
QPREAD    EQU      0  <BIT>      READ
QPWRITE   EQU      1  <BIT>      WRITE
QPBCSP    EQU      2  <BIT>      BACKSPACE
QPREWID   EQU      3  <BIT>      REWIND
QPENDFL   EQU      4  <BIT>      ENDFILE
QPOPEN    EQU      5  <BIT>      OPEN
QPFIRST   EQU      6  <BIT>      FIRST I/O
PREIO1    EQU     UCB+13  (1)
QPUNFT    EQU      0  <BIT>      UNFORMATTED I/O
QPCUNB    EQU      1  <BIT>      PROCESS IN BACKSP OF UNFORMAT REC.
RECLEN    EQU     UCB+14 (2)     RECORD LENGTH OF OPEN STMT.

*
IOSHD     EQU     UCB+16         IOS PARAMETER BLOCK (in UCB)
IOSCODE   EQU     IOSHD+0 (1)
QRDTRAN   EQU      $00           DATA TRANSFER REQUESTS
QRCFUNC   EQU      $01           COMMAND FUNCTIONS
IOSFUNC   EQU     IOSHD+1 (1)
QDREAD    EQU      1             READ REQUEST
QDWRIT    EQU      2             WRITE REQUEST
QDOUIN    EQU      4             OUTPUT WITH INPUT
QDUPDT    EQU      8             UPDATE REQUEST
QDDELT    EQU      $10           DELETE RECORD
```

```
QDFMTD    EQU      $20          FORMAT DISK
QCPSTN    EQU      1            POSITION
QCRWND    EQU      2            REWIND
QCTEST    EQU      4            TEST I/O
QCWAIT    EQU      8            WAIT ONLY
QCHALT    EQU      $10          HALT I/O
QCBRAK    EQU      $20          BREAK SERVICE
IOSOPTW   EQU     IOSHD+2  (2)
IOSOPT    EQU     IOSHD+3
QOBNRY    EQU      0  <BIT>     ASCII/BINARY BIT
QOPRCD    EQU      1  <BIT>     WAIT/PROCEED BIT
QOIMAG    EQU      3  <BIT>     FORMAT/IMAGE BIT
QOBRAK    EQU      4  <BIT>     BREAK NOTIFICATION BIT
QOSECH    EQU      5  <BIT>     SUPRESS ECHO BIT
QOBLCK    EQU      6  <BIT>     RECORD/BLOCK ACCESS BIT
QORKEY    EQU      7  <BIT>     LOGICAL RECORD/RANDOM KEY ACCESS BIT
IOSOPT1   EQU     IOSHD+2  (1)
**********
QORTKY    EQU      0 <BIT>  RETURN KEY WITH RECORD BIT
QOCMPA    EQU      1 <BIT>  COMPLETION ADDRESS BIT
QOIIMG    EQU      2 <BIT>  INPUT FORMAT/IMAGE BIT
QOSCMM    EQU      3 <BIT>  PRIMARY/SECONDARY MEMORY MAP BIT
QOFRMT    EQU      4 <BIT>  FORMAT OPTION BIT
QOLNEXT   EQU      $00          NEXT RECORD
QOLCRNT   EQU      $20          CURRENT RECORD
QOLPRIR   EQU      $40          PRIOR RECORD
QOLRECN   EQU      $60          RECORD ASSOCIATED WITH IOSRECN
*   IOSOPT OF SEQ.
QIOPAFRN EQU       $0000    FORMATTED WITHOUT CONTIGUOUS FILE
QIOPBIRN EQU       $0409    UNFORMATTED WITHOUT CONTIGUOS FILE
QIOPBIBN EQU       $0449    UNFORMATTED CONTIGUOUS FILE
*   IOSOPT OF DIR.
QIOPAFRR EQU       $6000    FORMATTED WITHOUT CONTIGUOUS FILE
QIOPBIRR EQU       $6409    UNFORMATTED WITHOUT CONTIGUOUS FILE
QIOPBIBR EQU       $6449    CONTIGUOUS FILE OR DISC
*   IOSOPT OF PSN.
QIOPBCS   EQU      $4000    BACKSPACE
QIOPRWD   EQU      $0000    REWIND
*   IOSOPT OF PST.
QIOPSTP   EQU      $0000    STOP , PAUSE
IOSSTUS   EQU     IOSHD+4 (1)
QERILF    EQU      $82              ILLEGAL FUNC.  (SEE ONLY BACKSPACE)
QEREOF    EQU      $C2              END OF RECORD
QERCFND   EQU      $CA               RECORD FOUND
IOSLUN    EQU     IOSHD+5 (1)
IOSRECN   EQU     IOSHD+8 (4)
IOSSTRT   EQU     IOSHD+12 (4)
IOSEND    EQU     IOSHD+16 (4)
IOSTRNL   EQU     IOSHD+20 (4)

*
FHSHD     EQU     UCB+44        FHS PARAMETER BLOCK (in UCB)
FHSCODE   EQU     FHSHD+0 (1)
QCDEVF    EQU      $00          DEVICE/FILE COMMANDS
QCUTLY    EQU      $01          UTILITY COMMANDS
FHSCMND   EQU     FHSHD+1 (1)
QFCHKPT   EQU      1            CHECK POINT
QFDELET   EQU      2            DELETE
QFCLOSE   EQU      4            CLOSE
QFPRTCT   EQU      8            PROTECT
QFRENAM   EQU      $10          RENAME
QFCHGAP   EQU      $20          CHANGE ACCESS PERMISSION
QFASSGN   EQU      $40          ASSIGN
QFALLOC   EQU      $80          ALLOCATE
QUCHGLU   EQU      $10          CHANGE LU ASSIGNMENT
QUFTDMN   EQU      $20          FETCH DEVICE MNEMONICS
QUFTDIR   EQU      $40          FETCH DIRECTORY ENTRY
QURETAT   EQU      $80          RETRIEVE ATTRIBUTES
*FHSOPT
QFHOAP    EQU      $0004            ACCESS PERMISSION = PUBLIC READ/WRITE
QFHOAPW   EQU      $0002            A. P.  = PUBLIC WRITE
QFHOAPR   EQU      $0000    A. P.  = PUBLIC READ
QFHOOW    EQU      $0008            OVERWRITE OPTION
```

```
QFHOPE   EQU      $0040           OPEN POSITION IS END OF FILE
QFHOSF   EQU      $0100           SEQUENTIAL FILE
QFHOIF   EQU      $0200           INDEXED SEQ. FILE (NO DUP. KEY)
QFHOIFD  EQU      $0300           INDEXED SEQ. FILE (DUP.  KEY ALLOWED)
FHSOPT   EQU     FHSHD+2 (2)  UPPER ONE BYTE DEVICE CODE (TO UCBFDCD)
FHSSTUS  EQU     FHSHD+4 (1)  RETURN STATUS
QERFUN   EQU      $02          INVALID FUNCTION
QERAAS   EQU      $0D          ALREADY ASSIGNED
QEREFL   EQU      $17          FILE NOT EXIST
FHSLUN   EQU     FHSHD+5 (1)  LOGICAL UNIT NO.
FHSFDMP  EQU     FHSHD+6  (4)     POINTER OF FETCH DEVICE MNEMONIC
FHSUSN   EQU     FHSHD+10 (2)     USER NUMBER
FHSFDML  EQU     FHSHD+10 (4)     LENGTH OF FETCH DEVICE MNEMONIC
FHSVOLN  EQU     FHSHD+6 (4)  VOLUME NAME
FHSCATN  EQU     FHSHD+12      (8)  CATALOG NAME
FHSEXT   EQU     FHSHD+28 (2)     EXTENSION
FHSFILN  EQU     FHSHD+20 (8) FILE NAME
FHSDATT  EQU     FHSHD+32 (2) DEVICE ATTRIBUTE WORD
FHSRECL  EQU     FHSHD+34 (2) RECORD LENGTH
FHSSIZE  EQU     FHSHD+36 (4)
*
UCBEND   EQU     FHSHD+40
*
IOS      EQU     2       TRAP NO.
FHS      EQU     3       TRAP NO.
         LIST
```

CHAPTER 9

RUNTIME INTERFACE FOR NON-VERSAdos SYSTEMS

1.1 INTRODUCTION

The runtime routines supplied with M68000 FORTRAN depend upon the presence of VERSAdos for proper operation. This chapter explains how to create a FORTRAN load module which is dependent upon RMS68K and BIOS, a basic I/O system. The information in this chapter applies to those users who have purchased the RMS68K package. Source is provided in the RMS68K package to allow customizing.

9.2 ADDING FILE HANDLING SERVICES TO BIOS

BIOS may be SYSGENed with RMS68K to provide basic I/O functions for user tasks. In the RMS68K and BIOS environment provided by Motorola, the File Handling Services (FHS) are not provided. In support of this environment, the FORTRAN runtime library FORTBIOS is provided. This runtime library does not contain any FHS calls, which implies that no file I/O can occur.

If a user wants to provide file support, then the file handling services may be added to BIOS. In this case the library, FORTVMC, would be needed.

The following table identifies the serial and parallel port configuration for VERSAmodules 1 and 2.

SYSTEM	SERIAL	PARALLEL	LU	
VM01	1		5	READ LOGICAL UNIT
			6	WRITE LOGICAL UNIT
	2		4	READ, WRITE LOGICAL UNIT
		1	3	WRITE LOGICAL UNIT
VM02	1		5	READ LOGICAL UNIT
			6	WRITE LOGICAL UNIT
	2		4	READ, WRITE LOGICAL UNIT
		No Parallel Port		

9.3 EXAMPLE

The following example illustrates a SYSGEN command file which allows the user to generate an operating system with a FORTRAN task. For more information about the SYSGEN facility, refer to the System Generation Facility User's Manual.

For more information about BIOS, refer to M68000 Family Real-Time Multitasking Software User's Manual, Appendix H.

```
*
*    This file builds up the operating system for a VM02
*    board system.  The operating system includes the
*    EXEC, BIOS, and INITialization tasks, and FORTRAN
*    task FORT.
*
*    SYSTEM PARAMETERS
*
GST=4                   Global Segment Table - number of pages
UST=2                   User Semaphore Table - number of pages
TRACE=5                 Trace Table          - number of pages
IOV=1                   I/O Vector Table     - number of pages
MMU=$0                  Address of MMU
TIMER=$F70000           Address of timer
CLOCKFRQ=800            Number of clock ticks per millisecond
TIMINTV=10              Number of milliseconds between timer
*                       interrupts
TIMSLIC=2               Number of timer interrupts before task
*                       forced to relinquish processor
PANEL=$0                Front panel address
MEMEND1=$20000          Maximum memory address
MEMEND2=$20000
MEMEND3=$40000
UDR=0                   User-defined directive tables not existent
TRCFLAG=$C000           Trace flag
WHERLOAD=$0             Memory address where boot file loaded
PAT=2                   Pages in the Periodic Activation Table
BUGTRAC=$F000BC         Address of VERSAbug trace routine
PC=$E00                 Initialize Program Counter
STACK=$C00              Stack location
KILVECT=142             Killer vector number
SERPTS=140              Serial port vector number
PTMVECT=28              Timer vector number
FAIL=141                AC fail vector number
SWABRT=31               Software abort vector number
NRAD1=0                 Number of RAD1 boards on system
DPRVAO=0                Dual-ported RAM VERSAbus address offset
NUSRRAD=0               Number of RAD1 users/boards
IOBINT4=$74             I/O channel interrupt vector number
IOBINT3=$73             I/O channel interrupt vector number
IOBINT2=$72             I/O channel interrupt vector number
IOBINT1=$71             I/O channel interrupt vector number
BCLRV=147               Bus clear interrupt vector number
*
*     Build EXEC
*
STARTRMS=$F00
PROCESS VM2.RMSV2.LO
END EXEC
MSG EXEC BUILT
*
*     Build BIOS
*
MEMBEG=*
TASK VM2.BIOS.LO
BIOSSTRT=*
SUBS VM2.LBIOS.CF
```

```
LINK VM2.LBIOS.CF
SESSION=1
PRIORITY=200
END BIOS
MSG BIOS BUILT
*
*   Build FORTRAN program
*
TASK VM2.FORT.LO
FORTSTRT=*
SUBS VM2.FORT.CF
LINK VM2.FORT.CF
SESSION=2
PRIORITY=100
ATTRIB='USER'
END FORT
MSG FORT BUILT
*
*   Build INITializer
*
PROCESS VM2.INIT.LO
SUBS VM2.INTIOV2
ASM VM2.EQUTIMER.SA/VM2.INTIOV2.SA,VM2.INTIOV2.RO,VM2.INTIOV2.LS
SUBS VM2.INDV.SA
ASM FIX:77.VM2.INDV,FIX:77.VM2.INDV,FIX:77.VM2.INDV
INTSTR=*
SUBS VM2.LNKINT2.CF
LINK VM2.LNKINT2.CF
END INIT
MSG INIT BUILT
END
```

The following are listings of the chain files mentioned above.

a. VM2.LBIOS.CF - link BIOS

```
=LINK ,VM2.BIOS.LO,#PR;MIX
SEG SEG0:8 \BIOSSTRT
IN FIX:77.VM2.BIOS
END
```

b. VM2.FORT.CF - link FORTRAN program

The LINK command must have the S and the -P options.

```
=LINK ,FIX:77.VM2.FORT,#PR;SMIX-P
SEG SEG0:6,7 \FORTSTRT
SEG SEG1:8,9
SEG SEG2:15
IN FIX:77.VM2.TEST1
LIB FIX:0.&.FORTBIOS
LIB FIX:0.&.FORTMATH
END
=END
```

c. VM2.LNKINT2.CF - link initializer

```
=LINK ,VM2.INIT.LO,#PR;IXHM
SEGMENT .INT:8 \INTSTR
INPUT VM2.INIT.RO,VM2.INTIOV2.RO,VM2.INDV.RO,VM2.SYSPARV.RO
END
=END
```

The following command line was used to execute the above SYSGEN file:

```
=SYSGEN SYSFORT,/VMCSYS.TEST1.SY,#PR;R
```

There are two ways to test VMCSYS.TEST1.SY:

a. Use the utility BUILDS to transform the binary load module into a file of ASCII-encoded information. Then use VERSAbug commands to load and execute the S-record file. Refer to the VERSAdos System Facilities Reference Manual for more information on BUILDS and S-records.

b. To test a VMC 68/2 system, use the following steps.

   1) Patch the following addresses on sector 0 of the floppy diskette containing the SYSGENed program.

```
$16 - starting sector number from DIR + 1  (1 word)
$18 - length of program - 1 from DIR       (1 word)
$1E - beginning address of EXEC from SYSGEN (2 words)
```

   2) Reset the VMC 68/2 and do the following:

```
- BH 0,1     (Boot and Halt from channel 0 device 1)
- .A7 C00    (set PC)
- G          (execute)
```

APPENDIX A

COMPILER MESSAGES

This appendix describes the messages output by the Compiler. There are three types:

a. a diagnostic message output to the listing file when the Compiler encounters a source program error (Table 1),

b. a message output to the user's console to describe the condition of the compile (Table 2), and

c. an abnormal ending message which is output to the user's console when an extraordinary termination occurs (Table 3).

TABLE 1. Error Messages

ERROR NUMBER	ERROR LEVEL	DESCRIPTION
002	E	INVALID CHARACTER APPEARS IN COLUMNS 1-5 OF LINE
003	E	THE STATEMENT NUMBER HAS ALREADY BEEN DEFINED
004	E	THE FIRST CHARACTER OF THE STATEMENT IS NOT ALPHABETIC
005	E	CONTINUATION LINE ENCOUNTERED WHEN COMMENT OR INITIAL LINE EXPECTED
006	E	LIMIT OF 9 CONTINUATION LINES EXCEEDED
007	W	COLUMNS 1-5 OF A CONTINUATION LINE ARE NOT BLANK
009	W	MISSING 'END' STATEMENT
010	W	THE NAME \P IS TOO LONG. IT HAS BEEN TRUNCATED TO SIX CHARACTERS
011	E	SYMBOL TABLE OVERFLOW
014	E	REAL CONSTANT OVERFLOW
015	F	ILLEGAL COMMAND LINE
016	E	INVALID CONSTANT FORMAT
017	E	INTEGER CONSTANT OVERFLOW
018	F	INTERNAL ERROR: ILLEGAL NODE TYPE FOUND IN "CODE_GEN".
022	E	EXPECTING RIGHT PARENTHESIS

TABLE 1. Error Messages (cont'd)

ERROR NUMBER	ERROR LEVEL	DESCRIPTION
023	E	EXPECTING SINGLE QUOTE
025	E	UNDECODABLE STATEMENT
026	E	INVALID CHARACTER \P
033	W	INVALID STATEMENT AFTER END STATEMENT. IT WAS IGNORED
035	F	ILLEGAL OPTION(S) IN COMMAND LINE
036	F	ILLEGAL INPUT FILE NAME
037	F	ILLEGAL OUTPUT FILE NAME
038	F	ILLEGAL LISTING FILE NAME
040	E	MISSING PROGRAM NAME
041	E	MISSING SUBROUTINE NAME
042	E	MISSING FUNCTION NAME
044	E	NON-SYMBOLIC NAME IS SPECIFIED IN TYPE SPECIFICATION STATEMENT
045	E	INVALID ARRAY DECLARATOR \P
047	E	EXPECTED COMMON BLOCK NAME
048	E	MISSING COMMA
049	E	NON-SYMBOLIC NAME IN AN EQUIVALENCE LIST
051	E	INCORRECT LENGTH SPECIFICATION IN TYPE SPECIFICATION STATEMENT
052	E	MISSING LIST OF NAMES IN INTRINSIC STATEMENT
053	E	INVALID TYPE OR LENGTH SPECIFICATION IN IMPLICIT STATEMENT
054	E	INVALID LETTER IN IMPLICIT STATEMENT \P
055	E	INVALID DIMENSION DECLARATOR IN \P
056	E	THE LENGTH OF A LITERAL IS LONGER THAN THE VARIABLE OR ARRAY ELEMENT
064	E	ILLEGAL STATEMENT IN BLOCKDATA SUBPROGRAM
065	E	ATTEMPT TO DEFINE A PREVIOUSLY DEFINED NAME IN EXTERNAL STATEMENT \P

TABLE 1. Error Messages (cont'd)

ERROR NUMBER	ERROR LEVEL	DESCRIPTION
067	E	NAME IN AN INTRINSIC STATEMENT MUST BE AN INTRINSIC FUNCTION NAME \P
069	E	ATTEMPT TO DEFINE A PREVIOUSLY DEFINED NAME IN INTRINSIC STATEMENT \P
070	E	ATTEMPT TO DEFINE A PREVIOUSLY DEFINED NAME IN SAVE STATEMENT
072	E	ATTEMPT TO ESTABLISH THE TYPE OF A CHARACTER MORE THAN ONCE
073	E	THE RANGE OF LETTERS IN AN IMPLICIT STATEMENT LIST IS NOT ALPHABETIC
079	E	ATTEMPT TO DEFINE A PREVIOUSLY DEFINED NAME AS A COMMON VARIABLE \P
083	E	WRONG NUMBER OF SUBSCRIPTS IN AN EQUIVALENCE LIST
085	E	A VARIABLE'S DIMENSION IS NOT A SIMPLE INTEGER VARIABLE \P
086	E	ATTEMPTING TO USE A PREVIOUSLY DEFINED NAME AS AN ARRAY \P
087	E	AN ADJUSTABLE ARRAY OR ASSUMED SIZE ARRAY MUST BE A DUMMY ARGUMENT \P
088	E	ATTEMPTING TO REDIMENSION A VARIABLE \P
090	E	INVALID FORMAT OF AN ASSUMED SIZE ARRAY DECLARATION
092	E	A VARIABLE DIMENSION \P IS NOT A DUMMY ARGUMENT OR COMMON VARIABLE
093	E	MORE THAN 3 DIMENSIONS FOR THE ARRAY \P
098	E	INVALID SYMBOLIC NAME APPEARS IN DATA STATEMENT \P
099	E	A VARIABLE WAS PREVIOUSLY INITIALIZED IN A DATA STATEMENT
100	E	ATTEMPT TO INITIALIZE NAMED COMMON ENTITY \P NOT IN BLOCK DATA SUB
101	E	ATTEMPTING TO INITIALIZE A BLANK COMMON VARIABLE \P
102	E	TYPE OF DATA AND VARIABLE DO NOT MATCH
108	E	A FUNCTION MUST NOT BE OF TYPE CHARACTER
112	E	ADJUSTABLE ARRAYS ARE VALID ONLY IN PROCEDURE SUBPROGRAMS
123	E	INVALID REFERENCE TO SUBROUTINE NAME

TABLE 1. Error Messages (cont'd)

ERROR NUMBER	ERROR LEVEL	DESCRIPTION
126	E	A DO LOOP PARAMETER IS NOT AN INTEGER EXPRESSION OR IS MISSING
131	E	THE DO INDEX IS NOT A SIMPLE INTEGER VARIABLE
133	E	MISSING INPUT/OUTPUT LIST IN IMPLIED DO LIST
135	E	TYPE DISAGREEMENT BETWEEN LEFT AND RIGHT SIDE OF EQUAL SIGN
139	E	ILLEGAL SEQUENCE OF OPERATORS/OPERANDS IN EXPRESSION
143	E	TYPE DISAGREEMENT BETWEEN ACTUAL AND DUMMY ARGUMENT
150	E	DIVIDE BY ZERO
152	E	UNDEFINED STATEMENT FUNCTION, OR STATEMENT FUNCTION REFERENCE ERROR
154	E	STATEMENT FUNCTION STATEMENT NAME CONFLICTS WITH PRIOR DEFINITIONS \P
155	E	DISAGREEMENT BETWEEN TYPE OR NUMBER OF ACTUAL AND DUMMY ARGUMENTS
156	E	MISMATCH IN NUMBER OF ACTUAL AND DUMMY ARGUMENTS IN AN INTRINSIC FUNCTION
157	E	\P IS DOUBLY DEFINED
158	E	PROCEDURE \P APPEARS AS ARGUMENT WITHOUT EXTERNAL DECLARATION
159	E	THERE IS AN ASSUMED SIZE ARRAY IN INPUT/OUTPUT LIST \P
160	E	STATEMENT FUNCTION STATEMENT NAME \P PASSED AS PARAMETER OR IN COMMON
163	E	THERE IS AN ERROR ON THE LEFT SIDE OF AN ASSIGNMENT STATEMENT
166	E	UNDECODABLE TYPE OF GOTO STATEMENT
172	E	LOGICAL IF CONTAINS ILLEGAL STATEMENT(S)
174	E	DO CONTROL VARIABLE USED PREVIOUSLY IN THE NEST
175	E	ILLEGAL TERMINAL STATEMENT OF DO
179	E	RECORD AND EOF SPECIFIER CONFLICT
180	E	FORMAT AND RECORD SPECIFIER CONFLICT
181	E	MISSING FORMAT IDENTIFIER WHERE AN INTERNAL FILE IS SPECIFIED

TABLE 1. Error Messages (cont'd)

ERROR NUMBER	ERROR LEVEL	DESCRIPTION
182	E	INTERNAL FILE AND RECORD SPECIFIER CONFLICT
183	E	WRITE STATEMENT MUST NOT CONTAIN AN EOF SPECIFIER
189	W	RETURN STATEMENT APPEARS IN THE MAIN PROGRAM
195	W	MAIN PROGRAM HAS NO STOP STATEMENT
197	W	FUNCTION VALUE NOT DEFINED IN THE FUNCTION SUBPROGRAM
199	E	ANYTHING AFTER A STATEMENT IS ILLEGAL
200	E	EXPECTING STATEMENT LABEL
201	E	EXPECTING COMMA OR RIGHT PARENTHESIS
203	E	EXPECTING SYMBOLIC NAME
204	E	EXPECTING COMMA OR RIGHT PARENTHESIS
205	E	EXPECTING LEFT PARENTHESIS
206	E	EXPECTING COMMA
207	E	EXPECTING EQUAL SIGN
208	E	EXPECTING LABEL, SYMBOLIC NAME, CHARACTER CONSTANT, 'REC' OR 'END'
211	E	EXPECTING 'DIRECT'
215	E	EXPECTING 'THEN'
218	E	EXPECTING 'TO'
220	E	MULTIPLE 'END' OR 'REC' SPECIFIED
224	W	NO STATEMENT LABEL AFTER ARITHMETIC IF, 'GOTO', 'STOP', OR 'RETURN'
226	E	REFERENCE TO ILLEGAL STATEMENT LABEL
227	E	ILLEGAL TRANSFER INTO DO LOOP, IF BLOCK, ELSE IF BLOCK OR ELSE BLOCK
230	E	INCREMENTATION PARAMETER IS ZERO
235	E	THE DO CONTROL VARIABLE IS REDEFINED WITHIN THE DO LOOP
237	E	THE VARIABLE MUST BE OF TYPE INTEGER

TABLE 1. Error Messages (cont'd)

ERROR NUMBER	ERROR LEVEL	DESCRIPTION
251	E	MORE THAN THREE LEVELS OF PARENTHESES IN FORMAT SPECIFICATION
254	E	NUMERIC SPECIFICATION GREATER THAN 255 IN FORMAT SPECIFICATION
255	E	NUMERIC SPECIFICATION IS ZERO IN FORMAT SPECIFICATION
257	E	DIGITS OF FRACTIONAL PART EXCEED TOTAL DIGITS OF NUMBER
260	E	CHARACTER CONSTANT LENGTH GREATER THAN 255 IN FORMAT SPECIFICATION
261	E	MISSING 'N' OR 'Z' AFTER 'B'
265	E	THE FIRST CHARACTER OF A CHARACTER FORMAT SPECIFICATION IS NOT '('
267	E	NO STATEMENT LABEL ON FORMAT STATEMENT
270	E	FORMAT INDEX VARIABLE MUST BE INTEGER*4
271	W	USELESS DATA TYPE - EXPECTED VARIABLE, ARRAY OR FUNCTION \P
272	E	OVERFLOW IN HEXADECIMAL NUMBER (MORE THAN 8 DIGITS)
273	E	ILLEGAL CHARACTER IN HEXADECIMAL NUMBER
274	E	MISSING ENDING 'H' IN HEXADECIMAL NUMBER
275	E	UNRECOGNIZED NAME OF LOGICAL/RELATIONAL OPERATOR
276	E	DOUBLE-REAL CONSTANT OVERFLOW
277	E	MORE THAN ONE PERIOD DETECTED IN REAL CONSTANT
278	E	MORE THAN ONE EXPONENT DETECTED IN REAL CONSTANT
279	E	UNDERFLOW IN REAL CONSTANT
281	E	TWO DIFFERENT VARIABLE TYPES ARE BOUND BY EQUIVALENCE STATEMENT \P
282	E	TWO EQUIVALENCED CHARACTER ENTITIES DO NOT HAVE THE SAME LENGTH \P
283	E	TWO DIFFERENT ARRAY ELEMENTS ARE ASSIGNED TO THE SAME ADDRESS \P
285	E	CHARACTER DATA AND NONCHARACTER DATA CANNOT BE IN THE SAME COMMON \P

TABLE 1. Error Messages (cont'd)

ERROR NUMBER	ERROR LEVEL	DESCRIPTION
287	E	COMMON BLOCK STORAGE CANNOT BE EXTENDED UPWARD BY EQUIVALENCE \P
289	E	A COMMON VARIABLE AND A SAVE VARIABLE ARE EQUIVALENCED \P
290	E	ELEMENTS OF DIFFERENT COMMON BLOCKS ARE EQUIVALENCED \P
293	E	THE SUBSCRIPT OF \P IN AN EQUIVALENCE STATEMENT IS INVALID
297	F	CAN'T OPEN INPUT FILE
298	F	CAN'T OPEN LISTING FILE
300	E	INTERNAL ERROR
301	E	INTEGER EXPRESSION IS EXPECTED
302	E	NUMBER SHOULD BE GREATER THAN ZERO
303	E	EOF MUST NOT BE SPECIFIED FOR AN INTERNAL FILE
304	E	FORMAT IDENTIFIER, IF ANY, MUST BE SECOND ITEM IN CIOLIST
305	E	UNEXPECTED EQUAL SIGN
306	E	ILLEGAL FORMAT SPECIFICATION
307	E	IMPLIED-DO CONTROL VARIABLE IS NOT A SIMPLE INTEGER VARIABLE
308	E	IMPLIED-DO LOOP HAS TOO MANY SIMPLE IOLIST ITEMS
309	E	DO CONTROL VARIABLE \P IS REDEFINED IN AN IOLIST
310	E	IMPLIED-DO CONTROL VARIABLE IS REDEFINED IN AN IOLIST \P
311	E	UNEXPECTED LEFT PARENTHESIS
312	E	UNEXPECTED RIGHT PARENTHESIS
313	E	UNEXPECTED COMMA
314	E	UNEXPECTED SLASH
315	E	UNEXPECTED NUMBER
316	E	UNEXPECTED MINUS SIGN
317	E	UNEXPECTED APOSTROPHE
318	E	UNEXPECTED B FORMAT SPECIFICATION

TABLE 1. Error Messages (cont'd)

ERROR NUMBER	ERROR LEVEL	DESCRIPTION
319	E	ILLEGAL X FORMAT SPECIFICATION
320	E	UNEXPECTED I FORMAT SPECIFICATION
321	E	UNEXPECTED L FORMAT SPECIFICATION
322	E	UNEXPECTED A FORMAT SPECIFICATION
323	E	UNEXPECTED D FORMAT SPECIFICATION
324	E	UNEXPECTED E FORMAT SPECIFICATION
325	E	UNEXPECTED F FORMAT SPECIFICATION
326	E	NUMBER IS MISSING BEFORE P FORMAT SPECIFICATION
327	E	NUMBER IS MISSING BEFORE H FORMAT SPECIFICATION
328	E	A NON-LOGICAL OPERAND \P APPEARS IN A LOGICAL EXPRESSION
329	E	UNEXPECTED END-OF-STATEMENT
330	E	UNEXPECTED CHARACTER IN FORMAT STATEMENT
331	E	MISSING FIELD WIDTH
332	F	INTERNAL ERROR -- NAME NOT FOUND
333	E	ILLEGAL USE OF MODULE NAME \P
334	E	ATTEMPTED TO PASS STATEMENT-FUNCTION-STATEMENT NAME AS ADDRESS
335	E	CONFLICT WITH PRIOR DEFINITIONS: \P
336	E	ILLEGAL ATTEMPT TO PASS \P AS ADDRESS
337	E	NO INTRINSIC STATEMENT FOR \P BUT IT IS PASSED AS ARGUMENT
338	E	AN ATTEMPT WAS MADE TO ASSIGN THE PROCEDURE \P
339	E	PROCEDURE NAME \P APPEARS IN DATA STATEMENT
340	E	UNBALANCED PARENTHESES IN IF STATEMENT
341	F	INTERNAL - NAME IN ATTRIBUTE TABLE DOES NOT START WITH ALPHA CHARACTER
342	F	INTERNAL - HASH TABLE FULL
343	E	EXPECTED VARIABLE NAME OR ARRAY NAME INSTEAD OF \P

TABLE 1. Error Messages (cont'd)

ERROR NUMBER	ERROR LEVEL	DESCRIPTION
344	E	DUMMY ARGUMENT \P APPEARS MORE THAN ONCE IN DUMMY ARGUMENT LIST
345	E	DUMMY ARGUMENT LIST MISSING -- PARENTHESES MUST APPEAR EVEN IF EMPTY
346	E	'RECL' EXPECTED
347	E	'ACCESS' EXPECTED
348	E	MISSING ELEMENTS IN EQUIVALENCE LIST
349	E	SLASH EXPECTED
350	E	DUMMY ARGUMENT OR SAVED ENTITY \P NOT ALLOWED IN COMMON
351	E	ARRAY ELEMENTS NOT ALLOWED IN SAVE \P
352	E	DUMMY ARGUMENT OR COMMON ENTITY \P NOT ALLOWED IN SAVE
353	E	SAVE ENTITIES MUST BE SIMPLE VARIABLES, ARRAY NAMES OR COMMON BLOCKS
354	E	NUMERIC INTEGER CONSTANT EXPECTED
355	E	DUMMY ARGUMENT \P NOT ALLOWED IN EQUIVALENCE LIST
356	W	RETURN MISSING IN FUNCTION OR SUBROUTINE
357	E	MORE THAN ONE HEADER (PROGRAM, FUNCTION, SUBROUTINE OR BLOCKDATA)
358	E	ILLEGAL ORDER OF STATEMENTS
359	E	MISSING DATA STATEMENTS IN BLOCKDATA SUBPROGRAM
360	E	MISSING EXECUTABLE STATEMENTS
361	E	UNCLOSED BLOCKS
362	E	ILLEGAL STATEMENTS IN BLOCKDATA SUBPROGRAM
363	E	ILLEGAL CHARACTERS IN STOP OR PAUSE STATEMENT
364	F	MISSING DIMENSION NUMBER IN ATTRIBUTE OF \P
365	F	"BIN_CODE" - ILLEGAL OPERATION CODE PASSED: \N
366	F	INTERNAL: "NEXTWORK" - TOO MANY WORK-REGISTERS NEEDED BY "BIN_CODE"

TABLE 1. Error Messages (cont'd)

ERROR NUMBER	ERROR LEVEL	DESCRIPTION
367	F	INTERNAL: "BIN_SPECIAL" - ILLEGAL COMMAND-CODE PASSED \N
368	F	INTERNAL: "BNFLSMPL" - ILLEGAL VAR-KIND IN SIMPLE-VAR NODE: \N
369	F	INTERNAL: "BNFLARRAY" - ILLEGAL VAR-KIND IN ARRAY-VAR NODE: \N
370	F	INTERNAL: "BNFLOPND/BNFLRUTN" - ILLEGAL OPERAND NODE-TYPE PASSED: \N
371	E	CODE SIZE IS GREATER THAN 32KB WHICH CONFLICTS WITH '-B' OPTION
372	F	INTERNAL: "BINCRE" - ILLEGAL OPERAND ADDRESS-MODE (MDxxx) FOUND: \N
373	F	INTERNAL: "BINPSEUD" - ILLEGAL OPERAND ADDRESS-MODE (MDxxx) FOUND: \N
374	F	INTERNAL: "BINOPEN" - CAN'T OPEN RO-FILE: \P
375	F	INTERNAL: "BNTMOPEN" - CAN'T RE-OPEN TEMPORARY RO-FILE: \P
376	F	INTERNAL: "BNTMREAD" - READ OF TEMPORARY RO-FILE FAILED: \P
377	F	INTERNAL: "BINWRT" - ILLEGAL READ-CODE (WR_xxx) PASSED: \N
378	E	SAVE+COMMON CODE IS GREATER THAN 32KB WHICH CONFLICTS WITH 'C' OPTION
379	F	INTERNAL: "BNWRBYTE" - WRITE ON RO-FILE FAILED: \P
380	F	INTERNAL: "BINCLOSE" - WRITE ON RO-FILE FAILED: \P
381	F	NESTING ERROR
382	E	STRING-CONSTANTS AND FORMATS SECTION SIZE IS GREATER THAN 32KB
383	E	ILLEGAL INTEGER NUMBER
384	E	CHARACTER LENGTH OF BOTH OPERANDS SHOULD BE THE SAME
385	E	ILLEGAL SYNTAX IN DATA STATEMENT
386	E	INCONSISTENT SUBSCRIPT REFERENCE
387	E	UNEQUAL NUMBER OF NAMES AND VALUES
388	E	ATTEMPT TO INITIALIZE NONCOMMON VARIABLE \P in BLOCK DATA SUBPROGRAM
389	E	SUBSCRIPT OF \P IS NOT AN INTEGER CONSTANT

TABLE 1. Error Messages (cont'd)

ERROR NUMBER	ERROR LEVEL	DESCRIPTION
390	F	FATAL ERROR IN DATA - DATA TABLE IS FULL
391	F	FATAL ERROR IN DATA/KEEP_GEN - KEEP_GEN GOT ODD OFFSET
393	E	MISMATCH BETWEEN OPERAND AND OPERATOR DATA TYPES
394	E	UNDEFINED LABEL \N
395	E	FIXED-POINT OVERFLOW
396	E	FIXED-POINT ZERO RAISED TO POWER OF NEGATIVE OR ZERO NUMBER
397	E	UNEXPECTED Z FORMAT SPECIFICATION
398	E	UNEXPECTED G FORMAT SPECIFICATION
399	F	PROGRAM IS EMPTY
400	E	INTERNAL FATAL ERROR IN: "MATCH_CONVERT"
401	F	INTERNAL FATAL ERROR IN: "SUBST_OP"
402	F	INTERNAL FATAL ERROR IN: "EXECUTE_OP"
403	E	INTERNAL FATAL ERROR IN: "CONVERT"
404	F	INTERNAL FATAL ERROR IN: "EXPARS"
405	F	INTERNAL FATAL ERROR IN: "ELESIZE" (VARIABLE NAME \P)
406	E	EXPRESSION NESTED TOO DEEP
407	F	INTERNAL FATAL ERROR IN: "SINTOF"
408	F	INTERNAL: NO MORE BUFFER ROOM AVAILABLE
409	F	INTERNAL: ATTRIBUTES TABLE FULL
410	F	INTERNAL: AN I/O ERROR OCCURRED

TABLE 2. Console Messages

MESSAGE	MEANING	NEXT STEP
FORTRAN (Vxx-xx).	Version number xx-xx of the FORTRAN Compiler is executing.	
COMPILATION CONCLUDED.	The FORTRAN Compiler has completed successfully.	
SOURCE FILE INVALID	The Compiler was unable to open the source file.	Ensure that the source file exists on the disk.
LOADING FAILED--- WLF1xxxx PHASE	The FORTRAN Compiler failed to load phase WLF1xxxx.	Check Table 3 for the meaning of abort code.
COMPILER FAILED xxxxxxxxxxxxxx.	The FORTRAN Compiler failed internally.	Check Table 3 for the meaning of abort code.

ABORT Codes

TABLE 3. Abnormal Termination

nnnn VALUE	MEANING	NEXT STEP
0000-00FF	FHS/IOS error in VERSAdos.	Refer to VERSAdos Data Management Services and Program Loader User's Manual.
0100-1999	Internal Compiler error.	1. Fix the errors and recompile. 2. If Appendix D applies, fix and recompile again. 3. Please contact local Motorola office if error is not solved by the above.
2001	Invalid file name in the FORTRAN command.	Check the file name and recompile.
2002	Invalid compile options in the FORTRAN command.	Check compile options and recompile.

APPENDIX B

RUNTIME ERROR MESSAGES

When an error occurs during execution, the program either continues or aborts. Error numbers 201 and 144 allow execution to continue; the rest cause an abort. The format of the error message is as follows:

```
** ERROR nnn message      (nnn is error number)
```

Table 1 shows these diagnostic messages.

TABLE 1. Diagnostic Messages

ERROR NUMBER	ERROR LEVEL	MESSAGE
101	C	RECURSIVE CALL
102	C	LOGICAL UNIT NUMBER OUT OF RANGE
103	C	END OF RECORD
104	C	FORMAT CODE MISSING
105	C	INVALID CHARACTER IN FORMAT
106	C	NEST OUT OF RANGE IN FORMAT
107	C	NUMBER OUT OF RANGE IN FORMAT
108	C	ILLEGAL DESCRIPTOR IN FORMAT
109	C	ILLEGAL SIGN WITHOUT SCALE FACTOR
110	C	INVALID DECIMAL CHARACTER
111	C	INVALID CHARACTER
112	C	INVALID HEXADECIMAL CHARACTER
113	C	FIXED POINT OVERFLOW
114	C	FLOATING POINT OVERFLOW
115	C	FLOATING POINT UNDERFLOW
116	C	NOT ENOUGH RECORDS IN UNFORMATTED READ
117	C	TOO MANY RECORDS IN UNFORMATTED READ

TABLE 1. Diagnostic Messages (cont'd)

ERROR NUMBER	ERROR LEVEL	MESSAGE
118	C	ASSIGN MISSING
119	C	INSUFFICIENT MEMORY FOR BUFFER
120	C	ERROR RETURN ON FHS CALL
121	C	DEVICE IS NOT READABLE
122	C	DEVICE IS WRITE PROTECTED
123	C	DEVICE DOES NOT SUPPORT BACKSPACE
124	C	DEVICE DOES NOT SUPPORT REWIND
125	C	DEVICE DOES NOT SUPPORT ENDFILE
126	C	DIRECT ACCESS NOT SUPPORTED
127	C	SEQUENTIAL ACCESS NOT SUPPORTED
128	C	UNFORMATTED I/O NOT SUPPORTED
129	C	DIRECT ACCESS IS ILLEGAL WITHOUT OPEN STATEMENT
130	C	UNIT WAS PREVIOUSLY ACCESSED DIRECTLY
131	C	UNIT WAS PREVIOUSLY ACCESSED SEQUENTIALLY
132	C	I/O ERROR DURING SEQUENTIAL ACCESS
133	C	I/O ERROR DURING DIRECT ACCESS
134	C	END OF FILE
135	C	UNIT ALREADY OPENED
136	C	ILLEGAL RECORD FORMAT
137	C	RECORD LENGTH OF OPEN STATEMENT GREATER THAN RECORD LENGTH OF FILE
138	C	RECORD NUMBER LESS THAN OR EQUAL TO 0
139	C	FILE IS NOT FORMATTED
140	C	I/O ERROR AT PAUSE OR STOP
141	C	ERROR OCCURRED DURING CLOSE
142	C	INDEX OUT OF RANGE

TABLE 1. Diagnostic Messages (cont'd)

ERROR NUMBER	ERROR LEVEL	MESSAGE
143	C	INSUFFICIENT MEMORY
144	C	DIVIDE BY ZERO
145	C	SOURCE ERROR
201	S	DIVIDE BY REAL ZERO
202	S	DIVIDE BY DOUBLE PRECISION ZERO
203	S	DIVIDE BY INTEGER ZERO
204	S	REAL POWER BASE = 0, EXP <= 0
205	S	DOUBLE PRECISION POWER BASE = 0, EXP <= 0
206	S	INTEGER POWER BASE = 0, EXP <= 0
207	S	SQRT ARG. <0
208	S	DSQRT ARG <0
209	S	EXP ARG > 127 LOG(2)
210	S	DEXP ARG > 1023 LOG(2)
211	S	ALOG ARG <= 0
212	S	DLOG ARG <= 0
213	S	ALOG10 ARG <= 10
214	S	DLOG10 ARG <= 10
215	S	SIN ARG >= 10**6
216	S	DSIN ARG >= 10**14
217	S	COS ARG >= 10**6
218	S	DCOS ARG >= 10**14
219	S	TAN ARG TOO LARGE
220	S	DTAN ARG TOO LARGE
221	S	ASIN ABS ARG > 1
222	S	DASIN ABS ARG > 1

TABLE 1. Diagnostic Messages (cont'd)

ERROR NUMBER	ERROR LEVEL	MESSAGE
223	S	ATAN ABS ARG TOO LARGE
224	S	DATAN ABS ARG TOO LARGE
225	S	ATAN2(X/Y) AREG X=Y=0
226	S	DATAN2(X/Y) ARG X=Y=0
227	S	ATAN2(X/Y) ARG Y TOO LARGE
228	S	DATAN2(X/Y) ARG Y TOO LARGE

APPENDIX C

AN EXAMPLE FROM COMPILATION TO EXECUTION

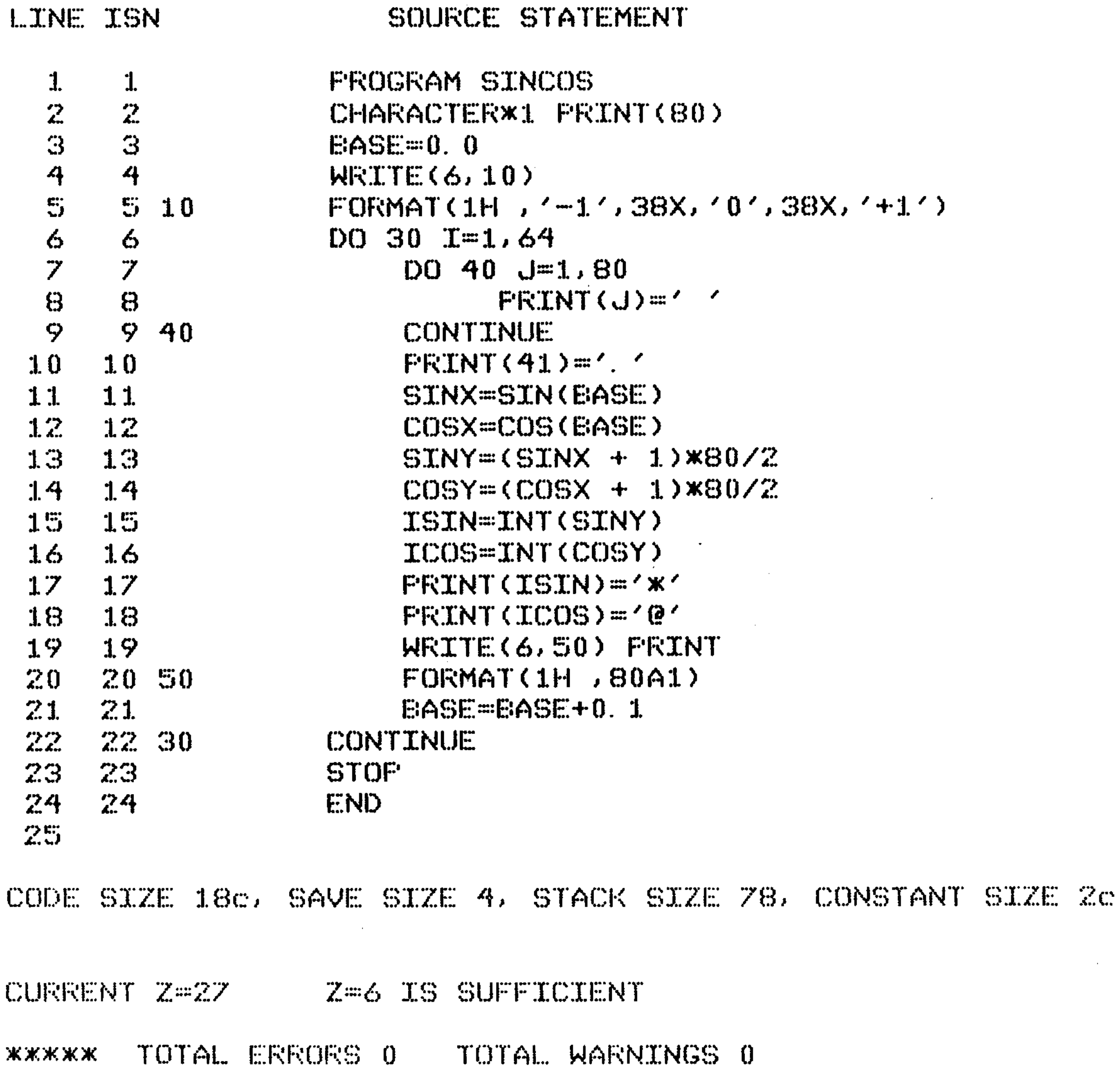
This appendix uses a simple program to illustrate the complete path of a FORTRAN program from compilation to execution. In this example, TESTPROG must print out the sine and cosine for values of X and also plot them on an X,Y grid.

Example

Using the VERSAdos FORTRAN command,

```
FORTRAN WORK:..TESTPROG,,#PR;A,S
```

the source program is compiled into a relocatable object module. The object file has the same name as the source file, with an extension name of RO for distinction. The compilation listings are output to the line printer. Figure 1 displays these listings.



```
LINE ISN             SOURCE STATEMENT

  1    1          PROGRAM SINCOS
  2    2          CHARACTER*1 PRINT(80)
  3    3          BASE=0.0
  4    4          WRITE(6,10)
  5    5 10       FORMAT(1H ,'-1',38X,'0',38X,'+1')
  6    6          DO 30 I=1,64
  7    7               DO 40 J=1,80
  8    8                    PRINT(J)=' '
  9    9 40            CONTINUE
 10   10               PRINT(41)='.'
 11   11               SINX=SIN(BASE)
 12   12               COSX=COS(BASE)
 13   13               SINY=(SINX + 1)*80/2
 14   14               COSY=(COSX + 1)*80/2
 15   15               ISIN=INT(SINY)
 16   16               ICOS=INT(COSY)
 17   17               PRINT(ISIN)='*'
 18   18               PRINT(ICOS)='@'
 19   19               WRITE(6,50) PRINT
 20   20 50            FORMAT(1H ,80A1)
 21   21               BASE=BASE+0.1
 22   22 30       CONTINUE
 23   23          STOP
 24   24          END
 25

CODE SIZE 18c, SAVE SIZE 4, STACK SIZE 78, CONSTANT SIZE 2c


CURRENT Z=27       Z=6 IS SUFFICIENT

*****   TOTAL ERRORS 0    TOTAL WARNINGS 0
```

FIGURE 1. Compilation Listing of Program TESTPROG (Sheet 1 of 4)

```
LINE ISN            SOURCE STATEMENT

  1    1          PROGRAM SINCOS
  2    2          CHARACTER*1 PRINT(80)
  3    3          BASE=0. 0
            000000     2F0E                MOVE. L    A6,-(A7)
            000002     2C4F                MOVE. L    A7,A6
            000004     9FFC00000000        SUB. L     #****,A7
            00000A     48E77F00            MOVEM. L   D1/D2/D3/D4/D5/D6/D7,-(A7)
  4    4          WRITE(6,10)
            00000E     42AEFFAC            CLR. L     -84(A6)
            000012     42A7                CLR. L     -(A7)
            000014     41F900000000        LEA        STR_ESDID-*+****,A0
            00001A     487B8800            PEA        0(PC,A0. L)
            00001E     42A7                CLR. L     -(A7)
            000020     42A7                CLR. L     -(A7)
            000022     7206                MOVEQ      #6,D1
            000024     48C1                EXT. L     D1
            000026     2F01                MOVE. L    D1,-(A7)
            000028     48780011            PEA        17. W
            00002C     4EAB0000            JSR        ESD17-. FRTPREF(A3)
            000030     4FEF0018            LEA        24(A7),A7
            000034     4EAB0000            JSR        ESD18-. FRTPREF(A3)
  5    5 10       FORMAT(1H ,'-1',38X,'0',38X,'+1')
  6    6          DO 30 I=1,64
            000038     7001                MOVEQ      #1,D0
            00003A     2D40FFA8            MOVE. L    D0,-88(A6)
            00003E     3D7C003FFFA6        MOVE. W    #63,-90(A6)
            000044     4A6EFFA6            TST. W     -90(A6)
            000048     6D000000            BLT        ***
  7    7              DO 40 J=1,80
            00004C     7001                MOVEQ      #1,D0
            00004E     2D40FFA2            MOVE. L    D0,-94(A6)
            000052     3D7C004FFFA0        MOVE. W    #79,-96(A6)
            000058     4A6EFFA0            TST. W     -96(A6)
            00005C     6D000000            BLT        ***
  8    8                  PRINT(J)=' '
  9    9 40           CONTINUE
            000060     222EFFA2            MOVE. L    -94(A6),D1
            000064     41F90000001C        LEA        STR_ESDID-*+28,A0
            00006A     1DBB880010AF        MOVE. B    0(PC,A0. L),-81(A6,D1. W)
            000070     536EFFA0            SUBQ. W    #1,-96(A6)
            000074     52AEFFA2            ADDQ. L    #1,-94(A6)
            000078     60DE                BRA        *-32
 10   10              PRINT(41)='. '
 11   11              SINX=SIN(BASE)
 12   12              COSX=COS(BASE)
 13   13              SINY=(SINX + 1)*80/2
 14   14              COSY=(COSX + 1)*80/2
 15   15              ISIN=INT(SINY)
 16   16              ICOS=INT(COSY)
 17   17              PRINT(ISIN)='*'
 18   18              PRINT(ICOS)='@'
 19   19              WRITE(6,50) PRINT
            00007A     41F90000001E        LEA        STR_ESDID-*+30,A0
            000080     1D7B8800FFD8        MOVE. B    0(PC,A0. L),-40(A6)
            000086     2C2EFFAC            MOVE. L    -84(A6),D6
            00008A     2006                MOVE. L    D6,D0
```

FIGURE 1. Compilation Listing of Program TESTPROG (Sheet 2 of 4)

```
LINE ISN            SOURCE STATEMENT

            00008C     4EAB0000          JSR        ESD19-. FRTPREF(A3)
            000090     2E00              MOVE. L    D0,D7
            000092     2D47FF9C          MOVE. L    D7,-100(A6)
            000096     2006              MOVE. L    D6,D0
            000098     4EAB0000          JSR        ESD20-. FRTPREF(A3)
            00009C     2A00              MOVE. L    D0,D5
            00009E     2D45FF98          MOVE. L    D5,-104(A6)
            0000A2     2007              MOVE. L    D7,D0
            0000A4     247C3F800000      MOVE. L    #1065353216,A2
            0000AA     4EAB0000          JSR        ESD21-. FRTPREF(A3)
            0000AE     247C42A00000      MOVE. L    #1117782016,A2
            0000B4     4EAB0000          JSR        ESD22-. FRTPREF(A3)
            0000B8     247C40000000      MOVE. L    #1073741824,A2
            0000BE     4EAB0000          JSR        ESD23-. FRTPREF(A3)
            0000C2     2C00              MOVE. L    D0,D6
            0000C4     2D46FF94          MOVE. L    D6,-108(A6)
            0000C8     2005              MOVE. L    D5,D0
            0000CA     247C3F800000      MOVE. L    #1065353216,A2
            0000D0     4EAB0000          JSR        ESD21-. FRTPREF(A3)
            0000D4     247C42A00000      MOVE. L    #1117782016,A2
            0000DA     4EAB0000          JSR        ESD22-. FRTPREF(A3)
            0000DE     247C40000000      MOVE. L    #1073741824,A2
            0000E4     4EAB0000          JSR        ESD23-. FRTPREF(A3)
            0000E8     2E00              MOVE. L    D0,D7
            0000EA     2D47FF90          MOVE. L    D7,-112(A6)
            0000EE     2006              MOVE. L    D6,D0
            0000F0     4EAB0000          JSR        ESD24-. FRTPREF(A3)
            0000F4     2A00              MOVE. L    D0,D5
            0000F6     2D45FF8C          MOVE. L    D5,-116(A6)
            0000FA     2007              MOVE. L    D7,D0
            0000FC     4EAB0000          JSR        ESD24-. FRTPREF(A3)
            000100     2C00              MOVE. L    D0,D6
            000102     2D46FF88          MOVE. L    D6,-120(A6)
            000106     2205              MOVE. L    D5,D1
            000108     41F90000001F      LEA        STR_ESDID-*+31,A0
            00010E     1DBB880010AF      MOVE. B    0(PC,A0. L),-81(A6,D1. W)
            000114     2206              MOVE. L    D6,D1
            000116     41F900000020      LEA        STR_ESDID-*+32,A0
            00011C     1DBB880010AF      MOVE. B    0(PC,A0. L),-81(A6,D1. W)
            000122     42A7              CLR. L     -(A7)
            000124     41F900000000      LEA        STR_ESDID-*+****,A0
            00012A     487B8800          PEA        0(PC,A0. L)
            00012E     42A7              CLR. L     -(A7)
            000130     42A7              CLR. L     -(A7)
            000132     7206              MOVEQ      #6,D1
            000134     48C1              EXT. L     D1
            000136     2F01              MOVE. L    D1,-(A7)
            000138     48780811          PEA        17. W
            00013C     4EAB0000          JSR        ESD17-. FRTPREF(A3)
            000140     4FEF0018          LEA        24(A7),A7
            000144     486EFFB0          PEA        -80(A6)
            000148     48780050          PEA        80. W
            00014C     48780801          PEA        2049. W
            000150     42A7              CLR. L     -(A7)
            000152     4EAB0000          JSR        ESD25-. FRTPREF(A3)
            000156     4FEF0010          LEA        16(A7),A7
```

FIGURE 1. Compilation Listing of Program TESTPROG (Sheet 3 of 4)

```
LINE ISN              SOURCE STATEMENT

               00015A     4EAB0000             JSR       ESD18-. FRTPREF(A3)
 20   20 50                 FORMAT(1H ,80A1)
 21   21                    BASE=BASE+0. 1
 22   22 30             CONTINUE
               00015E     202EFFAC             MOVE. L   -84(A6),D0
               000162     247C3DCCCCCD         MOVE. L   #1036831949,A2
               000168     4EAB0000             JSR       ESD21-. FRTPREF(A3)
               00016C     2D40FFAC             MOVE. L   D0,-84(A6)
               000170     536EFFA6             SUBQ. W   #1,-90(A6)
               000174     52AEFFA8             ADDQ. L   #1,-88(A6)
               000178     6000FECA             BRA       *-308
 23   23                STOP
 24   24                END
               00017C     42A7                 CLR. L    -(A7)
               00017E     4EAB0000             JSR       ESD26-. FRTPREF(A3)
               000182     588F                 ADDQ. L   #4,A7
               000184     4CDF00FE             MOVEM. L  (A7)+,D1/D2/D3/D4/D5/D6/D7
               000188     4E5E                 UNLK      A6
               00018A     4E75                 RTS
 25



                                      SYMBOL TABLE

 NAME         ATTR          ADDR     SIZE          TYPE    COMMON
BASE    LOCAL. V        ffffffac                   R4
COS     INTFUNC         ********                   R4
COSX    LOCAL. V        ffffff98                   R4
COSY    LOCAL. V        ffffff90                   R4
I       LOCAL. V        ffffffa8                   I4
ICOS    LOCAL. V        ffffff88                   I4
INT     INTFUNC         ********                   I2
ISIN    LOCAL. V        ffffff8c                   I4
J       LOCAL. V        ffffffa2                   I4
PRINT   LOCAL. A        ffffffaf 80                C1
SIN     INTFUNC         ********                   R4
SINCOS  PROG            ********
SINX    LOCAL. V        ffffff9c                   R4
SINY    LOCAL. V        ffffff94                   R4



                                      LABEL TABLE

LABEL    ATTR        ADDR

   10    FRMT     00000000
   30    EXEC     00000170
   40    EXEC     00000070
   50    FRMT     00000022

CODE SIZE 18c, SAVE SIZE 4, STACK SIZE 78, CONSTANT SIZE 2c


CURRENT Z=27       Z=6 IS SUFFICIENT

*****   TOTAL ERRORS 0    TOTAL WARNINGS 0
```

FIGURE 1. Compilation Listing of Program TESTPROG (Sheet 4 of 4)

Linkage Editor Example

Using the Linkage Editor command, the next step is to prepare the load module.

```
LINK WORK:..TESTPROG,,#PR;MIXL=FORTLIB
```

```
Options in Effect:  -A,-B,-D,-H,-I,-L,M,O,P,-Q,-R,-S,-U,-W,-X


Load Map:


Segment SEG0: 00000000 000000FF 0,1,2,3,4,5,6,7
Module        S   T  Start     End       Externally Defined Symbols

SINCOS        7      00000000  00000003  .FCBREF    00000000


Segment SEG1(R): 00000100 00004DFF 8,9,10,11,12,13,14
Module        S   T  Start     End       Externally Defined Symbols

.FINIT        8      00000100  00000455  .FINIT     000001AA
.FICOM        8      00000456  00000587  .FICOM     00000456  .FRTPREF   00000588
.FIAFL        8      00000588  000009A7  .FIAFL     00000588
.FIINT        8      000009A8  00000C01  .FIINT     000009A8  .FIIEEP    00000AF4
.FILST        8      00000C02  00000CFD  .FILST     00000C02  .FILST3    00000C6A
.FIFNL        8      00000CFE  00000DFF  .FIFNL     00000CFE
.FICFL        8      00000E00  00000E5D  .FICFL     00000E00
.FINFT        8      00000E5E  00000F8F  .FINFT     00000E5E
.FIFMT        8      00000F90  0000171F  .FIFMT     00000F90
.FISEQ        8      00001720  00001B71  .FISEQ     00001720
.FIDIR        8      00001B72  00001DDD  .FIDIR     00001B72
.FIPST        8      00001DDE  00001EEB  .FIPST     00001DDE
.FIERR        8      00001EEC  00002ADF  .FIERF     00001EF2  .FIERR     00001EEC
.FICLS        8      00002AE0  00002B2D  .FICLS     00002AE0
.FIUBA        8      00002B2E  00002C0D  .FIUBA     00002B2E
.FIUOP        8      00002C0E  00002C9D  .FIUOP     00002C0E
.FICVO        8      00002C9E  00002CE1  .FICVO     00002C9E
.FIFOI        8      00002CE2  00002D17  .FIFOI     00002CE2
.FIFOF        8      00002D18  00002EC9  .FIFOF     00002D18
.FIFOD        8      00002ECA  000031D5  .FIFOD     00002ECA
.FIFOG        8      000031D6  00003203  .FIFOG     000031D6
.FIFOL        8      00003204  00003221  .FIFOL     00003204
.FIFOA        8      00003222  0000325F  .FIFOA     00003222
.FIFOZ        8      00003260  000032B1  .FIFOZ     00003260
.FICOI        8      000032B2  000033F9  .FICOI     000032B2
.FICOR        8      000033FA  00003705  .FICOR     000033FA
.FICVI        8      00003706  00003749  .FICVI     00003706
.FIFII        8      0000374A  00003815  .FIFII     0000374A
.FIFID        8      00003816  00003867  .FIFID     00003816
.FIFIG        8      00003868  00003895  .FIFIG     00003868
.FIFIL        8      00003896  000038EB  .FIFIL     00003896
.FIFIA        8      000038EC  0000393D  .FIFIA     000038EC
.FIFIZ        8      0000393E  00003A0B  .FIFIZ     0000393E
.FICII        8      00003A0C  00003B51  .FICII     00003A0C
.FICIR        8      00003B52  00003F4B  .FICIR     00003B52
.FICTBL       8      00003F4C  00004213  .FICTA     00003F9C  .FICTB     000040FC
.FRCRI        8      00004214  00004271  .FRCRI     00004214
.FRSIR        8      00004272  000043DD  .FRSIR     00004278  .F.RSIR    00004272
.FRCOR        8      000043DE  00004557  .FRCOR     000043E4  .F.RCOR    000043DE
```

FIGURE 2. Linkage Editor Listing of Program TESTPROG (Sheet 1 of 2)

```
. FRSIN         8      00004558  000045AB  . FRSIN       00004558
. FRCOS         8      000045AC  000045F7  . FRCOS       000045AC
. FRMUD         8      000045F8  000047ED  . FRMUD       000045F8
. FRMUR         8      000047EE  000048ED  . FRMUR       000047EE
. FRSUR         8      000048EE  00004907  . FRSUR       000048EE
. FRADR         8      00004908  000049FD  . FRADR       00004908
. FRDIR         8      000049FE  00004B47  . FRDIR       000049FE
. FRIMR         8      00004B48  00004B6B  . FRIMR       00004B48
SINCOS          9      00004B6C  00004CF7  . FMAIN       00004B6C
SINCOS         10      00004CF8  00004D23

Segment SEG2:  00004E00 000051FF 15
Module          S   T  Start     End       Externally Defined Symbols

. FINIT        15      00004E00  000051B1  . FZWRK       00004E50

Unresolved References: None

Multiply Defined Symbols: None

Lengths (in bytes):

     Segment      Hex          Decimal

       SEG0     00000100          256
       SEG1     00004D00        19712
       SEG2     00000400         1024
Total Length    00005200        20992

   No Errors
   No Warnings

Load module has been created.
```

FIGURE 2. Linkage Editor Listing of Program TESTPROG (Sheet 2 of 2)

Example of Load Module Execution

To execute the load module created by the linker in the previous step, use the following command:

```
WORK:..TESTPROG O=#PR
```

```
X          SIN(X)    COS(X)
0.0000000 0.0000000 1.0000000
0.1000000 0.0998334 0.9950042
0.2000000 0.1986693 0.9800667
0.3000000 0.2955202 0.9553366
0.4000000 0.3894183 0.9210610
0.5000000 0.4794255 0.8775826
0.6000000 0.5646424 0.8253356
0.6999999 0.6442177 0.7648424
0.7999999 0.7173560 0.6967069
0.8999999 0.7833269 0.6216102
0.9999998 0.8414708 0.5403026
1.1000000 0.8912073 0.4535964
1.2000000 0.9320390 0.3623581
1.3000000 0.9635582 0.2674993
1.3999990 0.9854497 0.1699677
1.4999990 0.9974951 0.0707379
1.5999990 0.9995737-0.0291938
1.6999990 0.9916650-0.1288437
1.7999990 0.9738480-0.2272012
1.8999990 0.9463005-0.3232886
1.9999990 0.9092979-0.4161458
2.0999990 0.8632100-0.5048451
2.1999990 0.8084972-0.5885001
2.2999990 0.7457062-0.6662750
2.3999980 0.6754643-0.7373929
2.4999980 0.5984734-0.8011428
2.5999980 0.5155028-0.8568879
2.6999980 0.4273815-0.9040715
2.7999980 0.3349899-0.9422219
2.8999980 0.2392512-0.9709578
2.9999980 0.1411220-0.9899923
3.0999980 0.0415828-0.9991351
3.1999980-0.0583720-0.9982949
3.2999980-0.1577434-0.9874802
3.3999970-0.2555388-0.9667989
3.4999970-0.3507809-0.9364576
3.5999970-0.4425182-0.8967596
3.6999970-0.5298339-0.8481015
3.7999970-0.6118557-0.7909695
3.8999970-0.6877641-0.7259344
3.9999970-0.7568004-0.6536462
4.0999970-0.8182752-0.5748268
4.1999960-0.8715740-0.4902639
4.2999960-0.9161645-0.4008025
4.3999960-0.9516010-0.3073364
4.4999960-0.9775294-0.2107995
4.5999960-0.9936906-0.1121564
4.6999960-0.9999232-0.0123927
4.7999960-0.9961650 0.0874949
4.8999960-0.9824535 0.1865082
4.9999960-0.9589256 0.2836591
5.0999960-0.9258164 0.3779737
5.1999960-0.8834568 0.4685127
5.2999950-0.8322700 0.5543705
5.3999950-0.7727675 0.6346893
5.4999950-0.7055439 0.7086663
5.5999950-0.6312706 0.7755628
5.6999950-0.5506899 0.8347100
```

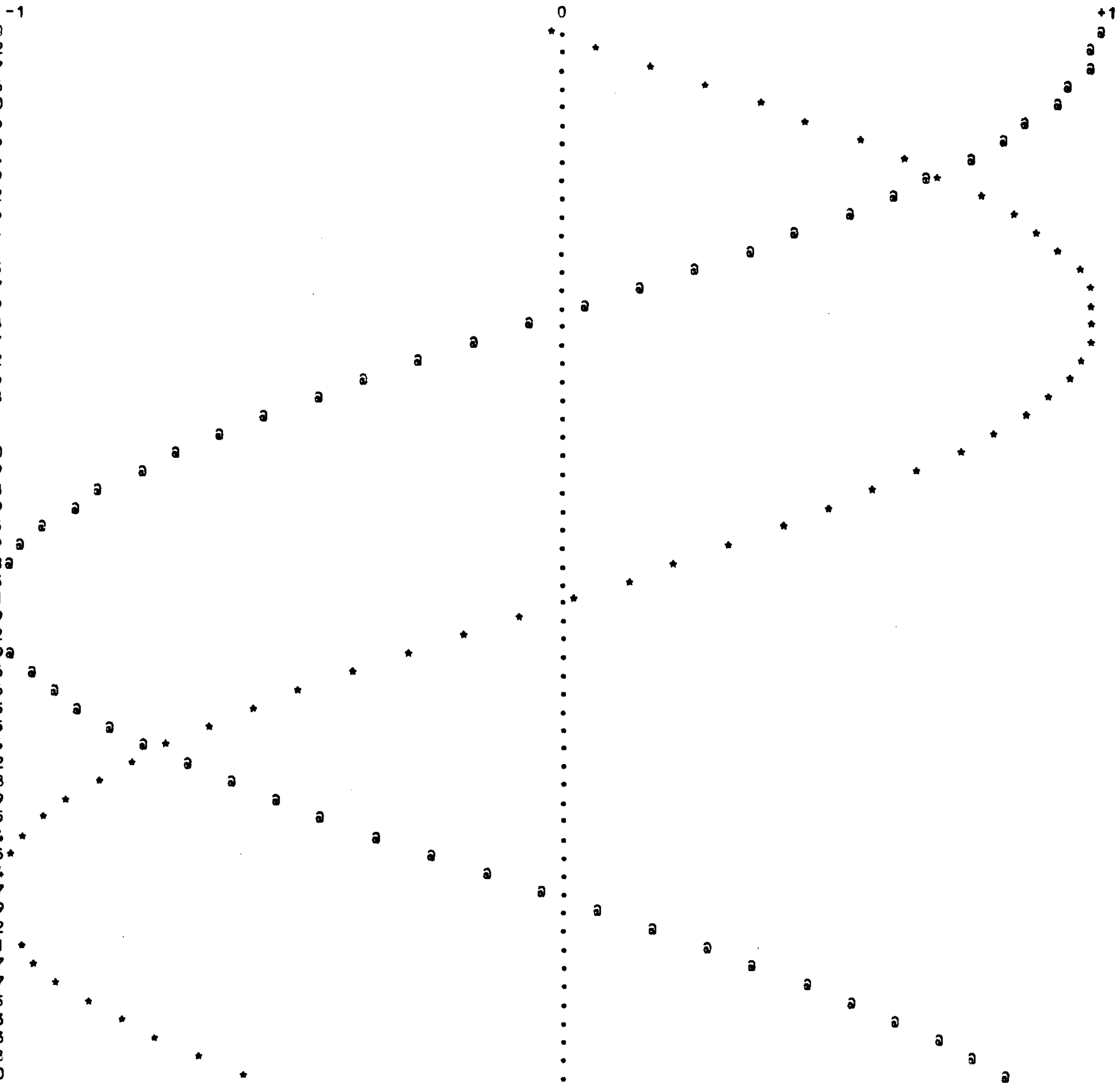


FIGURE 3. Listing of TESTPROG Execution

APPENDIX D

COMPILER LIMITS

NUMBER	CONDITIONAL ITEM	CONDITIONAL CONTENT
1	Set of characters	ASCII character set
2	Continuation lines	9 lines
3	Maximum number of digits in a statement number	5 digits
4	Maximum number of characters in a symbolic name	6 alphanumeric characters - first character must be alphabetic
5	Numeric value limits	Integer 2 bytes = -2**15 to 2**15-1 (largest decimal number = 32,767) 4 bytes = -2**31 to 2**31-1 (largest decimal number = 2,147,483,647) Real 4 bytes = 10**-39 to 10**39 (7 decimal digits) 8 bytes = 10**-309 to 10**309 (15 decimal digits)
6	Maximum number of dimensions	3 dimensions
7	Logical unit limits	SYSGEN-dependent, usually 1 - 8
8	Character data length	1 - 255
9	Number of characters allowed in STOP and PAUSE statement message	5 letters
10	Symbol table size	Dependent upon Z option (see Table 3-1)
11	Label table size	Dependent upon Z option (see Table 3-1)
12	Block nest number (sum of DO block nest + block IF statement nest)	25
13	Maximum sum of characters in all character constants	32K characters

NUMBER	CONDITIONAL ITEM	CONDITIONAL CONTENT
14	Maximum number of common blocks and number of external linker restrictions	240
15	FORTRAN cannot interface with Pascal subprograms.	
16	FORTRAN cannot interface with the fast floating point package without going through a conversion process.	

APPENDIX E

M68000/ANSI 77 FORTRAN SUBSET DIFFERENCES

E.1 INTRODUCTION

This appendix describes the language differences between the M68000 FORTRAN and the ANSI 77 subset standard (ANSI X3.9 - 1978). The M68000 FORTRAN supports the entire ANSI X3.9 FORTRAN subset with the following extensions.

In the following paragraphs, specific sections of the ANSI X3.9 FORTRAN language manual are referenced by:

(ANSI X3.9 - specific section or chapter number [F])

where [F] refers to Full Language definition. Otherwise, the chapter or section is in the Subset Language definition --

```
i.e.,   (ANSI X3.9 - 4) references chapter 4 in the Subset Language
        (ANSI X3.9 - 4.2) references section 4.2 in the Subset Language
        (ANSI X3.9 - 4.2F) references section 4.2 in the Full Language
```

A reference to section 4.5 would also include all the subsections, such as 4.5.1 and 4.5.2.

E.2 DATA TYPES AND CONSTANTS (ANSI X3.9 - 4)

This implementation supports the following data types:

```
INTEGER          - two distinct sizes
REAL             - two distinct sizes
DOUBLE PRECISION - (ANSI X3.9 - 4.5F)
                   also includes the intrinsic functions associated with
                   this data type (ANSI X3.9 - 15.10)

LOGICAL
CHARACTER
```

A constant data type has been added:

HEXADECIMAL

E.2.1 Integer Data Type (ANSI X3.9 - 4.3)

The size of an integer variable is either two bytes or four bytes. Four bytes is the default size. The size of a variable can be specified with the TYPE statement (see E.3).

NOTE

The user must ensure that the size of a dummy argument and its corresponding actual argument agree (i.e., both must be two bytes or both must be four bytes). Integer constants are always passed as four bytes.

E.2.2 Real Data Type (ANSI X3.9 - 4.4)
Double Precision Data Type (ANSI X3.9 - 4.5F)

The size of a real variable is either four bytes or eight bytes. Four bytes is the default size. The size of a variable can be specified with the TYPE statement (see E.3). An 8-byte real variable is equivalent to a double precision variable.

E.2.3 Logical Data Type (ANSI X3.9 - 4.7)

Logical variables are four bytes long, in conformance with the ANSI requirement that logicals and integers be the same length.

E.2.4 Hexadecimal Constant

The form of a hexadecimal constant is:

```
#<string of hexadecimal digits>H
```

The hexadecimal digits include 0-9 and A-F, with the digits A-F corresponding to the values 10-15, respectively.

Hexadecimal constants can be used in DATA statements and anywhere an integer constant could be used --

```
i.e.,  INTEGER INTH
       INTH = #FEH           This assigns the value 254 to INTH.
```

E.3 SPECIFICATION STATEMENTS (ANSI X3.9 - 8)

To support the different sizes of integer and real variables, the specification statements -- IMPLICIT and TYPE -- were enhanced.

E.3.1 TYPE Statement (ANSI X3.9 - 8.4.1)

The form of a TYPE statement is:

```
<type>[*<len>[,]] <name>[,<name>]...
```

where:

type is one of INTEGER, REAL, LOGICAL, or DOUBLE PRECISION.

len specifies the length of a real or integer variable. For real variables, <len> must be 4 or 8, with the default case being 4. For integer variables, <len> must be 2 or 4, with the default case being 4. For data types LOGICAL and DOUBLE PRECISION, the <len> attribute is syntactically incorrect.

name is one of the following:

```
v[*<len>]              v is a variable name.
a[(d)][*<len>]         a(d) is an array declarator.
```

i.e.,

```
INTEGER I,J*4  - I and J are 4-byte integers.
INTEGER*2 L,K  - L and K are 2-byte integers.
INTEGER M,N(10),O*2,P(10)*2,Q   - M, Q, and array N are 4-byte integers.
                                - O and array P are 2-byte integers.

REAL A,B*8     - A is a 4-byte real while B is a double precision real
                 with eight bytes.
REAL*8 C,D(10) - C is an 8-byte real and D is a double precision array
```

E.3.2 IMPLICIT Statement (ANSI X3.9 - 8.5)

The form of the IMPLICIT statement is:

```
IMPLICIT <type>[*<len>] (<a>[,<a>]...)
```

where:

type is one of INTEGER, REAL, LOGICAL, or DOUBLE PRECISION.

len specifies the length of a real or integer variable. For real variables, <len> must be 4 or 8, with the default case being 8. For integer variables, <len> must be 2 or 4, with the default case being 4. For data types LOGICAL and DOUBLE PRECISION, the <len> attribute is syntactically incorrect.

a is either a single letter or a range of single letters in alphabetical order.

E.3.3 INTRINSIC Statement (ANSI X3.9 - 8.8)

The ISA bit manipulation functions -- IOR, IAND, NOT, IEOR, ISHFT, IBSET, IBCLR, and BTEST -- cannot be used as actual arguments.

E.4 FUNCTIONS AND SUBROUTINES (ANSI X3.9 - 15)

To support the different sizes of integer and real variables, the FUNCTION statement was enhanced. Also, the INTRINSIC functions to support the DOUBLE PRECISION data type were added. The ISA 1976 bit string manipulation functions were also added.

E.4.1 FUNCTION Statement (ANSI X3.9 - 15.5.1)

The form of a FUNCTION statement is:

```
<type> FUNCTION <fun>[*<len>] ([<d>[,<d>]...])
```

where:

type specifies the length of a real or integer variable. For real variables, <len> must be 4 or 8, with the default case being 8. For integer variables, <len> must be 2 or 4, with the default case being 4. For data types LOGICAL and DOUBLE PRECISION, the <len> attribute is syntactically incorrect.

fun is the symbolic name of the function subprogram in which the FUNCTION statement appears.

len specifies the length of a real or integer variable. For real variables, <len> must be 4 or 8, with the default case being 4. For integer variables, <len> must be 2 or 4, with the default case being 4. For data types LOGICAL and DOUBLE PRECISION, the <len> attribute is syntactically incorrect.

d is a dummy argument.

E.4.2 INTRINSIC Functions (ANSI X3.9 - 15.10)

E.4.2.1 Additional Functions. The following intrinsic functions have been added to support the DOUBLE PRECISION data type. The definition of each function can be found in the table located in (ANSI X3.9 - 15.10):

IDINT, SNGL, DBLE, DINT, DNINT, IDNINT, DABS, DMOD, DSIGN, DOIM,
DMAX1, DMIN1, DSQRT, DEXP, DLOG, DLOG10, DSIN, DCOS, DTAN, DASIN,
DACOS, DATAN, DATAN2, DSINH, DCOSH, DTANH.

E.4.2.2 Integer Actual Arguments. Wherever an intrinsic function expects an integer actual argument, either a 2-byte or a 4-byte integer may be used.

E.4.3 ISA BIT STRING MANIPULATION

The subprograms which follow allow the programmer to view integer data as ordered sets of bits (a_n, a_n-1,......a_0), where the set is a place positional binary representation of an integer value, thus permitting interrogation and manipulation of integers on a bit-by-bit basis. The value of n is either 16 or 32, depending on the data type of the input variable.

E.4.3.1 Logical Operations. These operations are external functions. In the following functions, j and m are integer expressions. Operations are performed on all bits which represent the value of an integer internal to the processor. Operations are done bit-by-bit on corresponding bits -- that is, the corresponding bits of the actual arguments j and m are used to generate the integer result.

E.4.3.1.1 Inclusive OR - The form of this function reference is:

IOR(j,m)

where the result of IOR(j,m) is:

$$\sum_{k=0}^{n} 2^k * (j_k + m_k - (j_k * m_k))$$

E.4.3.1.2 Logical Product - The form of this function reference is:

IAND(j,m)

where the result of IAND(j,m) is:

$$\sum_{k=0}^{n} 2^k * (j_k * m_k))$$

E.4.3.1.3 Logical Complement - The form of this function reference is:

NOT(j)

where the result of NOT(j) is:

$$\sum_{k=0}^{n} 2^k * (1-j_k)$$

E.4.3.1.4 Exclusive OR - The form of this function reference is:

IEOR(j,m)

where the result of IEOR(j,m) is:

$$\sum_{k=0}^{n} 2^k * (2 - (j_k + m_k)) * (j_k * m_k)$$

E.4.3.2 Shift Operations

This operation is an external function. In the following function, j and m are integer expressions. Operations are performed on all bits which represent the value of an integer internal to the processor, and are used to generate an integer result.

The form of this function reference is:

ISHFT(j,m)

where, if the value of m is positive or zero, the result of ISHFT(j,m) is:

$$\sum_{k=0}^{n-m} 2^{k+m} * j_k$$

where, if the value of m is negative, the result of ISHFT(j,m) is:

$$\sum_{k=m}^{n} 2^{k+m} * j_k$$

E.4.3.3 Bit Testing and Setting. These operations are external functions. In the following functions, j and m are integer expressions.

E.4.3.3.1 Bit Test - This logical function tests a specified bit of an integer.

The form of this function reference is:

BTEST(j,m)

where the result of BTEST(j,m) is:

if IAND(j,2^m) = 0, then FALSE, else TRUE

E.4.3.3.2 Bit Set - This function sets a specified bit of an integer.

The form of this function reference is:

IBSET(j,m)

where the result of the function reference IBSET(j,m) is:

IOR(j,2^m)

E.4.3.3.3 Bit Clear - This function clears a specified bit of an integer.

The form of this function reference is:

IBCLR(j,m)

where the result of the function reference IBCLR(j,m) is:

IAND(j,NOT(2^m))

E.4.4 INPUT Function

This function reads one byte of data from the address specified by its argument n. n is a 4-byte integer expression.

The form of the function is: INPUT(n)

E.4.5 OUTPUT Subroutine

This subroutine outputs the low order byte of data m to address n in memory. n and m are 4-byte integer expressions.

The form of the subroutine is: OUTPUT(n,m)

E.4.6 Block Data Subprograms (ANSI X3.9 - 16F)

Block data subprograms are used to provide initial values for variables and array elements in named common blocks. See the ANSI manual for a complete definition.

E.5 MORE GENERALIZED EXPRESSIONS

E.5.1 Subscript Expressions

A subscript expression is not restricted to integer expressions, as in the Subset Language, but may also contain array element references and function references as in the Full Language. For example, a statement of the following form is allowed:

```
        A(I,J) = B(IT(J)) * C(IFUNC(K))
```

where A, B, IT, and C are arrays and IFUNC is a function.

E.5.2 Expressions as Output List Items (ANSI X3.9 - 12.8.2.2F)

An output list item may be not only a variable name, an array element name, or an array name, but also may be any arithmetic expression. For example, the following is allowed:

```
        WRITE (* '(1H, 10F7.2)') A+B, C*D(I)+E,FUNC(G)
```

E.5.3 Integer Expressions as External Unit Identifiers (ANSI X3.9 - 12.3.3F)

An external unit identifier is not restricted to integer constants or variables, but may be any integer expression with a zero or positive value. For example, the following is allowed:

```
        READ(IFILE(J),100) X,Y,Z
```

E.5.4 Integer Expressions as Record Length Specifiers (ANSI X3.9 - 12.10.1F)

The record length specifier is not restricted to integer constants or variables, but may be any integer expression with a positive value. Furthermore, the value may be up to 65,535, which is the largest record length allowed by VERSAdos. For example, the following is allowed:

```
        OPEN (IUNIT(IFILE), ACCESS = 'DIRECT',RECL = LEN(IFILE))
```

E.5.5 Integer Expressions as Record Specifiers (ANSI X3.9 - 12.5F)

The record specifier is not restricted to integer constants or variables, but may be any integer expression with a positive value. Furthermore, the value is not restricted to less than 32,768, but may be up to 2,147,483,647 (2**31 - 1). For example, the following is allowed:

```
        READ (IUNIT,100, REC - I+40000) A
```

MOTOROLA Semiconductor Products Inc.

P.O. BOX 20912 • PHOENIX, ARIZONA 85036 • A SUBSIDIARY OF MOTOROLA INC.

16834 PRINTED IN USA (11/83) MPS 3M