RT-11
FORTRAN Compiler
and Object Time System
User's Manual

DEC-11-LRFPA-A-D

RT-11
FORTRAN Compiler
and Object Time System
User's Manual

DEC-11-LRFPA-A-D

Order additional copies as directed on the Software Information page at the back of this document.

digital equipment corporation · maynard, massachusetts

First Printing June 1974

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this manual.

The software described in this document is furnished to the purchaser under a license for use on a single computer system and can be copied (with inclusion of DIGITAL's copyright notice) only for use in such system, except as may otherwise be provided in writing by DIGITAL.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

The HOW TO OBTAIN SOFTWARE INFORMATION page, located at the back of this document, explains the various services available to DIGITAL software users.

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

CDP	DIGITAL	INDAC	PS/8
COMPUTER LAB	DNC	KA10	QUICKPOINT
COMSYST	EDGRIN	LAB-8	RAD-8
COMTEX	EDUSYSTEM	LAB-8/e	RSTS
DDT	FLIP CHIP	LAB-K	RSX
DEC	FOCAL	OMNIBUS	RTM
DECCOMM	GLC-8	OS/8	RT-11
DECTAPE	IDAC	PDP	SABR
DIBOL	IDACS	PHA	TYPESET 8
			UNIBUS

PREFACE

This document provides information necessary to compile, link, execute, and debug a FORTRAN program under the RT-11 operating system. Chapter one describes the operational procedures. Chapter two provides information about the Object Time System, a collection of routines selectively linked to the user's program which perform certain arithmetic operations, input/output operations, and system dependent service operations, and detect and report run-time error conditions. Chapter three describes system dependent information not included in the PDP-11 FORTRAN Language Reference Manual. The Appendices provide reference information about internal data representations, system subroutines, and compiler and OTS error diagnostics.

This manual should be used only after some competence with the FORTRAN Language, as implemented on the PDP-11, has been acquired. The associated document which may be used for this purpose is titled PDP-11 FORTRAN Language Reference Manual. The user should also be familiar with RT-11 Operating System as described in the RT-11 System Reference Manual.

DOCUMENTATION CONVENTIONS

All RT-11 monitor and system program command lines are terminated by a RETURN. Since this is a non-printing character, at certain places in the text the notation <CR> represents the RETURN key.

In all examples text that is typed by the monitor or system program is underlined, text typed by the user is not underlined.

CONTENTS

FIGURES

TABLES

CHAPTER 1

OPERATING PROCEDURES

1.1 USING THE FORTRAN SYSTEM

Figure 1-1 outlines the steps required to prepare a FORTRAN source program for execution under the RT-11 Monitor: (1) compilation, (2) linking, and (3) execution.

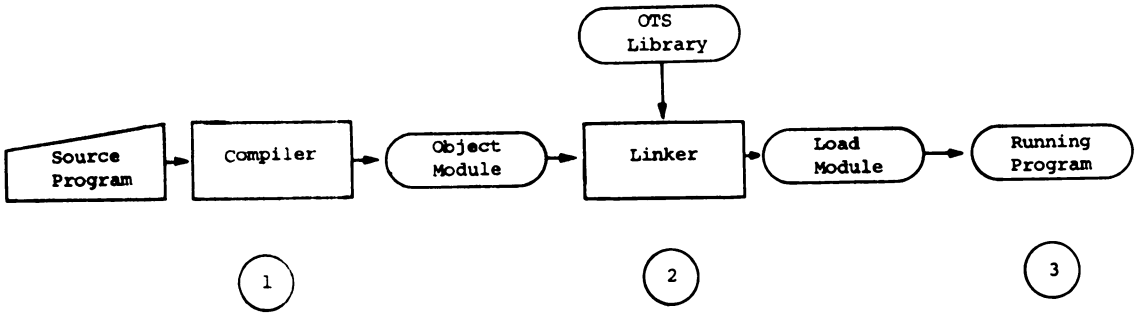


Figure 1-1
Steps in Compiling and Executing
a FORTRAN Program

Step 1 in Figure 1-1 is initiated by running the FORTRAN Compiler, FORTRAN, accompanied by a command string that describes the input and output files, and switch options, if desired, to be used by the Compiler. The Compiler generates an object file which must be linked by the linker prior to execution.

Step 2 is initiated by running the linker, LINK, accompanied by a similar command string. The linker links all program units and the necessary routines from the FORTRAN Library, and generates a core image file.

Step 3 is initiated by the monitor R or RUN commands.

1.1.1 Filename Specifications

The RT-11 FORTRAN Compiler and the Linker accept a standard RT-11 command string of the form:

```
outfile(s) = infile(s) /sw1/sw2.../swn
```

Each filename specification is of the form:

```
dev:filename.ext
```

where dev: is any legal device specification code. The RT-11 device specification codes are described in the Monitor chapter (chapter 2) of the RT-11 System Reference Manual.

Filename may be any 1- to 6- character alphanumeric filename. The filename extension may be any 1- to 3- character alphanumeric sequence. The filename extensions assumed or supplied on default by the FORTRAN Compiler are shown in Table 1-2.

Table 1-1
Filename Extensions

File	Assumed Extension on Input File	Default Extension on Output File
Object file	-	.OBJ
Listing file	-	.LST
Source file	.FOR	-

The optional switches, /sw1/sw2.../swn, are used to request certain functions from the FORTRAN compiler and Linker.

The RT-11 command string sequence is described more thoroughly in the Monitor Chapter of the RT-11 System Reference Manual.

1.2 COMPILATION PROCEDURES

To execute the RT-11 FORTRAN Compiler the command:

```
.R FORTRAN
```

is given. The FORTRAN Compiler then prints an * to indicate that it is ready to accept a command string.

The FORTRAN Compiler can produce two output files: an object file and a listing file. Up to six FORTRAN source language files are permitted as input files. If multiple input files are given they are considered to be logically concatenated. However, source lines should not be broken over file boundaries.

An input file may consist of more than one program unit if that file resides on a file structured device. The object code for all program units will be output to the single object file, and will be properly handled by LINK at link time.

NOTE

In the examples below, characters typed by the system are underlined to differentiate them from characters typed by the user.

A sample FORTRAN Compiler command sequence is shown below:

```
.R FORTRAN
*OBJECT,LIST=FILE1
```

This command string directs the Compiler to take the source file FILE1.FOR from the default device, DK:, and output the files LIST.LST and OBJECT.OBJ to the default device.

Either of the Compiler output files can be eliminated by omitting its file specification from the command string. For example:

```
.R FORTRAN
*FILE1=FILE1
```

produces FILE1.OBJ on the default device but no listing file, while

```
*,LP:=FILE1
```

produces a listing on the line printer, but no object module output.

1.2.1 Compiler Switches

The FORTRAN Compiler command strings may contain switch options on the input and output file specifications. The switches are as follows (they are initialized to the specified default values for each command string):

Switch	Description
/A	Add the compilation statistics to the list file (see section 1.2.2).
/D	Compile lines with a D in column one. These lines are treated as comment lines by default (see section 1.5).
/E	Read a full 80 columns of each record in the source file. Only the first 72 columns are read by default.
/H	Print a list of compiler switches on the listing device specified. If no listing device is specified, the output will be directed to the console terminal.
/L:n	Specify the listing options. The argument n is coded as follows:

/L or /L:0	list diagnostics only
/L:1	list source program only
/L:2	list storage map only
/L:4	list generated code only

Any combination of the above list options may be specified by summing the argument values for the desired list options. For example:

```
/L:7
```

requests a source listing, a storage map listing, and a generated code listing. If this switch is omitted the default list option is 3 (source and storage map). See section 1.2.2.

/N:m — Enable specification of the maximum number of logical units that may be concurrently open at execution time; m is an octal constant between 1 and 17. Defaults to 6 if switch is not specified. This switch functions only when one of the input files contains the main program unit.

/O — Include options-in-effect in list file. This list specifies the state of each compiler option, i.e., on or off. (See section 1.2.2).

/P — Disable the global optimizer. Using this switch may reduce program storage requirements, but will slightly increase execution time.

/R:m — Enable specification of the maximum record size allowed at execution time; m is an octal constant between 4 and 7777. Defaults to 136 (decimal) bytes if this switch is not specified. This switch functions only when one of the input files contains the main program unit.

/S — Suppress ISNs (source line number accounting). This option reduces storage requirements for generated code and slightly increases execution speed but disables line number information during Traceback.

/T — Allocate two words for default length integer variables. Normally, single storage words will be the default allocation for integer variables not given an explicit length specification (i.e. INTEGER*2 or INTEGER*4).

/U — Disable USR (User Service Routines) swapping at runtime. By default the USR is always swapped. This switch will function only when one of the input files contains the main program unit.

/V — Disable all vectoring of arrays (see section 2.6).

/W — Enable compiler warning diagnostics (see section C.1.3).

1.2.2 Listing Formats

There are five optional sections that may be included in the list file. By default the source program and the storage map are included in the list file. The list of options-in-effect, the generated code, and the compiler statistics may also be included. Any combination of these sections can be requested by using switches in the Compiler command string (see section 1.2.1). A description of each section is given below. Figure 1-2 provides a sample of the information included in each section.

1.2.2.1 Options Listing - The options-in-effect list can be used as a quick reference to the status of each possible compiler option. Options in the list preceded by 'NO' are not in effect; those not preceded by 'NO' are in effect. The maximum number of logical units that may be concurrently open (NLCHN) and the maximum record length (LRECL) are given as the default values or the values specified by the /N and /R switches respectively. Also included are the date and a copy of the Compiler command string for identification purposes.

1.2.2.2 Source Listing - The source program is listed in this section just as it appeared in the input file. Internal sequence numbers are added by the compiler for easier reference. Note that internal sequence numbers are not always incremental. For example the statement following a logical IF will have an internal sequence number 2 greater than that of the IF. The IF statement has internally been assigned 2 sequence numbers: one for the comparison and one for the associated statement.

1.2.2.3 Storage Map Listing - This section includes a list of all symbolic names referenced in the program unit. A relocation offset from the base of the program unit (subject to relocation at link time) is given for all local symbols. There is also a description of the symbolic name including usage, data type, and in the case of COMMON blocks and array names, the defined size in storage units.

NOTE

Blank COMMON will be described as COMMON BLOCK / / in the storage map, but will be located on a LINK map as a CSECT name .$$$$.

1.2.2.4 Generated Code Listing - This section of the list file contains a symbolic representation of the object code generated by the compiler (see section 2.2) including a location offset from the base of the program unit, the symbolic Object Time System (OTS) routine name, and routine arguments. The code generated for each statement is labeled with the same internal sequence number as that specified in the source program listing, providing easier cross reference.

1.2.2.5 Compilation Statistics - This section includes a report on core usage during the compilation process and the storage requirements for the object code generated by the compiler.

1.2.3 Compiler Memory Requirements

During the compilation process, the Resident Monitor (RMON), the Compiler root segment, one overlay region, the stack, and the required device handlers (other than the handler for the System Device which is included in the RMON) must be core resident. The remaining core memory is used for the Symbol Table and the internal representation of the program. In a machine with 8K of core memory, this allows the compilation of a program unit as large as several hundred statements in length. However, if the compiler runs out of free core during the compilation process (an error message is typed on the terminal; See section C.1), the program unit must be divided into two or more program units, each of which must be small enough to compile in the available amount of core memory.

Since device handlers and the Symbol Table must be core resident during the compilation process, minimizing the number of different physical devices specified in the command string and reducing the number of variable names, increases the amount of free core available for object code generation.

1.3 LINKING PROCEDURES

The RT-11 Linker, LINK, links one or more user-written program units together with selected routines from any user libraries (see section 1.3.1) and the default FORTRAN System Library, FORLIB. LINK also provides an overlay capability (see section 1.3.2).

LINK generates a single runnable core image file and an optional load map from one or more object files created by an RT-11 assembler or the RT-11 FORTRAN Compiler. An example of the LINK command format as used with FORTRAN is given below.

```
.R LINK
*LOAD,MAP=MAIN,SUB1,SUB2/F
```

This command string requests LINK to link the object module MAIN.OBJ together with the object modules SUB1.OBJ and SUB2.OBJ into the single core image file LOAD.SAV. All files are on the default device, DK:.

Tne switch, /F, specifies that the default FORTRAN Library on the System Device, SY:FORLIB.OBJ, is to be searched for any routines that are not found in the other object modules. These include any Library Functions, System Subroutines, or Object Time System routines. Note that the switch alone, without the explicit file specification, causes the default FORTRAN Library to be searched. This switch should be included if any of the object modules specified in the command string were created by the FORTRAN Compiler. This switch can be omitted, however, if the FORTRAN library file specification, SY:FORLIB, is explicitly included in the command string.

```
RT-11 FORTRAN IV          V01-07   23-MAY-74

,LP:/L:7/A/O=TEST

OPTIONS IN EFFECT:

                    SOURCE
                    MAP
                    CODE
                    RT-11
                    OPTIM
                    LRECL=0136
                    STAT
                    ISNS
                  NOCOL80
                    USRSWAP
                  NODIAG
                  NOINTEGER*4
                    NLCHN=06
                  NODEBUG
                    VECTOR
                  NOWARN




RT-11 FORTRAN IV          V01-07   SOURCE LISTING          PAGE 001

0001          INTEGER INT
0002          REAL REAL
0003          COMPLEX IMAG
0004          DOUBLE PRECISION DBLE
0005          DATA INT/100/
0006          REAL = INT/2 +5.
0007          DBLE = REAL/2. + 3.14159682516
0008          IMAG = CMPLX(REAL, 3.21)
0009          WRITE(5,10) IMAG
0010    10    FORMAT(1X,2F8.5)
0011          STOP
0012          END




RT-11 FORTRAN IV          STORAGE MAP

NAME     OFFSET   ATTRIBUTES

INT      000006   INTEGER*2 VARIABLE
REAL     000030   REAL*4    VARIABLE
IMAG     000034   COMPLEX*8 VARIABLE
DBLE     000044   REAL*8    VARIABLE
CMPLX    000000   COMPLEX*8 PROCEDURE
```

```
RT-11 FORTRAN IV          GENERATED CODE


ISN #0006
000054  LSN$     #000006
000060  MOI$MS   000006
000064  DII$IS   #000002
000070  CFI$
000072  ADF$IS   #040640
000076  MOF$SM   000030

ISN #0007
000102  ISN$
000104  MOF$MS   000030
000110  DIF$IS   #040400
000114  ADF$MS   000020
000120  CDF$
000122  MOD$SM   000044

ISN #0008
000126  ISN$
000130  REL$     000024
000134  REL$     000030
000140  CAL$     #000002 CMPLX+#000000
000146  MOD$RM   000034

ISN #0009
000152  ISN$
000154  REL$     000016
000160  REL$     000010
000164  IFW$
000166  REL$     000034
000172  TVC$
000174  EOL$

ISN #0011
000176  LSN$     #000013
000202  STP$

ISN #0012
000204  ISN$
000206  RET$




***** COMPILATION STATISTICS *****
*                                 *
*                                 *
*------- COMPILER  TABLES -------*
* SYMBOLS:          00079 WORDS *
* PROGRAM:          00041 WORDS *
* FREE CORE:        25018 WORDS *
*                                 *
*--------- OBJECT CODE ----------*
* IMPURE SECTION:   00022 WORDS *
* PURE SECTION:     00046 WORDS *
* TOTAL:            00068 WORDS *
*                                 *
***********************************
```

The optional Load Map file specification, if included, requests the Linker to output a list of module names, Common blocks and Global Symbols together with their absolute memory address assignments.

For a more detailed description of LINK refer to the Linker Chapter of the RT-11 System Reference Manual.

1.3.1 Library Usage

The FORTRAN programmer may want to create a library of commonly used Assembly Language and FORTRAN functions and subroutines. The RT-11 system program LIBR provides a library creation and modification capability. The Librarian Chapter of the RT-11 System Reference Manual describes the LIBR Program in detail.

A library file may be included in the LINK command string simply by adding the file specification to the input file list. LINK recognizes the file as a library file and links only those routines that are required. The LINK command string

```
*LOAD=MAIN,LIB1/F
```

requests LINK to link MAIN.OBJ with any required functions or subroutines contained in LIB1.OBJ. The default FORTRAN System Library, FORLIB, is then searched for any other required routines. The entire core image will be output to the file LOAD.SAV.

All user created libraries are searched before the default FORTRAN System Library, FORLIB, if the /F switch is used.

Under certain circumstances library directories will be made core resident to speed up library searches. This will therefore reduce the amount of core available for the symbol table. Directories will be made resident if the machine has 12K or more core memory, there is room for the particular directory, and the /S Linker switch is not included in the command string. If the Linker fails because it ran out of symbol table space linking object files, one of which was a library file, and the above conditions were also in effect, another attempt may be made including the /S Linker switch. This will slow down the linking process, but will allow the maximum possible symbol table space.

In the interest of maintaining the integrity of the DEC-distributed FORTRAN Library, the creation of a user library is preferred to the modification of, or addition to, the FORTRAN Library (FORLIB).

1.3.2 Overlay Usage

The overlay feature of the Linker allows segmentation of the core image so that the entire program need not be core resident at one time. This allows the execution of a program that otherwise would not fit in the available core memory.

An overlay structure consists of a root segment and one or more overlay regions. The root segment contains the FORTRAN main program and blank COMMON. The root segment may also contain some subroutines and function subprograms. An overlay region is an area of core allocated for two or more overlay segments, only one of which can be core resident at one time. An overlay segment consists of one or more subroutines or function subprograms.

At runtime, if a call is made to a routine that is contained in an overlay segment, the overlay handler checks to see if the segment is resident in its overlay region. If the segment is in memory, control is passed to the routine. If the segment is not resident, the overlay handler reads the overlay segment from the core image file on the system device (or other device of the same type as the system device) into the specified overlay region. This destroys the previous overlay segment in that overlay region. Control is then passed to the routine.

When dividing a FORTRAN program into a root segment and overlay regions, and subsequently dividing each overlay region into overlay segments, routine placement should be carefully considered. The user should always remember that it is illegal to call a routine located in a different overlay segment in the same overlay region, or an overlay region with a lower numeric value (as specified by the Linker overlay switch, /O:n) from the calling routine. The user should divide each overlay region into overlay segments which never need to be resident simultaneously.

The FORTRAN main program unit must be placed in the root segment.

In an overlay environment, Subroutine Calls and Function Subprogram references may refer only to one of the following:

1. A FORTRAN library routine (e.g. ASSIGN, DCOS)

2. A FORTRAN or Assembly Language routine contained in the root segment

3. A FORTRAN or Assembly Language routine contained in the same overlay segment as the calling routine

4. A FORTRAN or Assembly Language routine contained in a segment whose region number is greater than that of the calling routine.

In an overlay environment, COMMON blocks must be placed so that they are resident when referenced. Blank COMMON is always resident since it is always placed in the root segment. All named COMMON must be placed either in the root segment, or into the segment whose region number is lowest of all segments which reference the COMMON block. A named COMMON block can not be referenced by two segments in the same region unless the COMMON block appears in a segment or lower region number. The linker automatically places a COMMON block into the root segment if it is referenced by the FORTRAN main program or a subprogram that is located in the root segment. Otherwise the linker places a COMMON block in the first segment encountered in the linker command string that references that COMMON block.

All COMMON blocks which are data initialized (by use of DATA statements) must be so initialized in the segment in which they are placed.

The entire overlay initialization process is handled by LINK. The command format outlined below and further explained in the Linker Chapter of the RT-11 System Reference Manual is used to describe the overlay structure to the Linker. LINK links the runtime overlay handler with the user program, making the overlay process completely transparent to the user's program.

The size of the overlay region is automatically computed to be large enough to contain the largest overlay segment in that overlay region.

The root segment and all overlay segments are contained in the core image file generated by LINK.

Two switches are used to specify the overlay structure to LINK. The overlay switch is of the form:

```
/O:n
```

where n is an octal number specifying the overlay region number. The command Continuation switch is of the form:

```
/C
```

This switch allows the user to continue long command strings on the next line of input.

The first line of the LINK overlay structure command string should contain as the input list all object modules that are to be included in the root segment. This line should be terminated with the /C and /F switches. The /O:n switch can not appear in the first line of the command string. If all modules which are to be placed in the root segment cannot be specified on the first command line, additional modules may be specified on subsequent command lines each ending with a /C. The entire root segment must be specified before any overlays.

All subsequent lines of the command string should be terminated with an /O:n switch specifying an overlay region and/or a /C switch. The presence of only a /C switch specifies that this is a continuation of the previous line and therefore, a continuation of the specification of that overlay segment. The object modules on each line, or set of continued lines, constitute an overlay segment and share the specified overlay region with all other segments in the same numeric value overlay region. All but the last line of the command string should contain the /C switch.

For example, given the following overlay structure description:

1. a Main program and the object module SUB1 are to occupy the Root Segment

2. the object module SUB2 is to share an overlay region with the object module SUB3 (never co-resident)

3. the object modules SUB4 and SUB5 are to share a second overlay region with the object modules SUB6 and SUB7

the following command string could be used:

```
.R LINK
*LOAD=MAIN,SUB1/F/C
*SUB2/O:1/C
*SUB3/O:1/C
*SUB4/O:2/C
*SUB5/C
*SUB6/O:2/C
*SUB7
```

1.3.3 Stand-Alone FORTRAN

It is possible to generate a stand-alone FORTRAN program which can be executed on any PDP-11 not running RT-11. It should be noted that the stand-alone program should not be executed on any system running RT-11 or system failure will almost certainly result. The program functions as it normally would on an RT-11 system. The only I/O device which can be supported with FORTRAN-level I/O, however, is the terminal. Other devices could be supported via appropriate user-written assembly language routines.

To generate a stand-alone program the source program units should be compiled as usual. At link time special options must be employed to generate the stand-alone program. The /L switch must be included in the LINK command string so that the proper format output file is generated. The /I switch must also be used so that two special modules can be requested to be linked to the program from the FORTRAN library. These two modules are:

```
$SIMRT     ;RT-11 simulator

$nK        ;where n=4,8,12,16,20,24
            or 28 and specifies the amount
            of core available on the target
            machine.
```

The following command sequence generates a file called LOAD.LDA which may be punched on paper tape and loaded, using the Absolute Loader, on any PDP-11 with 8K or more core memory:

```
.R LINK
*LOAD,LP:=MAIN,SUBS/F/L/I

LIBRARY SEARCH:
$SIMRT<CR>
$8K<CR>
<CR>
*↑C
.
```

1.4 EXECUTION PROCEDURES

To start execution of the core image file generated by LINK, use the monitor R or RUN commands. The command:

```
.R MYPROG
```

or

```
.RUN DEV:MYPROG
```

causes the file on the system device (or device DEV:), MYPROG.SAV, to be loaded into core and executed.

The following shows how to take three FORTRAN source files containing a main program and several subroutines through the procedures necessary to Compile, Link, and Execute that program:

```
.R FORTRAN
*MAIN,LP:=MAIN,SUB
*SUB1,LP:=SUB1
*↑C
.R LINK
*MAIN,LP:=MAIN,SUB1/F
*↑C
.R MAIN
```

1.5 DEBUGGING A FORTRAN PROGRAM

The RT-11 debugging program, ODT, usually cannot be effectively used with a FORTRAN program due to the nature of the object code generated by the FORTRAN Compiler (see section 2.2).

However, in addition to the FORTRAN OTS error diagnostics which include the Traceback Feature (see section 2.5), there is another debugging tool available to the FORTRAN programmer. A D in column one of a FORTRAN statement allows that statement to be conditionally compiled. These statements are considered comment lines by the FORTRAN Compiler unless the /D switch is used in the Compiler command string. In this case the lines are compiled as regular FORTRAN statements. Liberal use of PAUSE statements and selective variable print out can provide the user with a method of monitoring program execution. This feature allows the inclusion of debugging aids that can be compiled in the early program development stages and later treated as regular comment lines.

CHAPTER 2

FORTRAN OPERATING ENVIRONMENT

2.1 FORTRAN OBJECT TIME SYSTEM

The FORTRAN Object Time System (OTS) is composed of the following:

1) Math routines, including the FORTRAN library functions and other arithmetic routines (e.g. floating point routines)

2) Miscellaneous utility routines (ASSIGN, DATE, SETERR)

3) Routines which handle various types of FORTRAN I/O

4) Error handling routines which process arithmetic errors, I/O errors, and system errors

5) Miscellaneous routines required by the compiled code.

The FORTRAN Library is designed as a collection of many small modules so that unnecessary routines can be omitted at Link time. For example, if the user program performs only sequential formatted I/O, none of the random access I/O routines are linked to his program.

The OTS version and edit number should be specified when communicating with DEC Software Support concerning the OTS. The version and edit number can be found in the LINK map. They will appear as a symbol and associated value in the .ABS. section of the link map listing and will have the following form:

V00xc = nnnnnn

where x = a decimal digit
c = an alphabetic character
n = an octal digit

2.2 OBJECT CODE

Typical FORTRAN operations often require common sequences of PDP-11 machine instructions. For example, at the end of any DO-loop, the index variable must be incremented, compared with the limit value, and a conditional branch taken. Other standard sequences might be

generated to locate an element of a multidimensional array, initialize an input/output operation, or simulate a floating-point operation not supported by the hardware configuration.

The common sequences of PDP-11 instructions are contained in a library known as the Object Time System. The RT-11 FORTRAN Compiler selects a certain combination of these instruction sequences to implement a FORTRAN program. During program execution, these sequences are threaded together and effect the desired result.

The Compiler references a library instruction sequence by generating a word containing the address of the first instruction in the sequence, followed by information upon which the sequence is to operate. In the case of the end-of-DO-loop sequence the information required is the location of the index variable, the limit value, and the address of the beginning of the loop. At runtime, register 4 is used to thread together the various references to library instruction sequences; the last instruction executed by each instruction sequence is JMP @(R4)+, which transfers control to the next library instruction sequence.

The mnemonics used for the Library routine names follow a logically consistent format. The mnemonics are usually six characters in length. The first two characters specify an operation. The third character specifies the mode of the operation, i.e. integer, floating, double precision, complex, or logical. The fourth character is always a dollar sign ($). The fifth and sixth characters, if present, specify, respectively, a source and destination for the operation. The source element for the operation can be a memory location, the hardware stack, the hardware registers, or an in-line argument which can be referenced through R4. The destination element for an operation can be a memory location, the hardware stack, or a location specified as an in-line argument which can be referenced through R4.

The library routines perform arithmetic operations, compare values, test values, calculate subscripts, convert from one mode type to another, and transfer program control. There are special routines to handle statement line numbers, enabling the FORTRAN Traceback feature, a routine to handle subprogram control transfer, and a routine to push the address of variables on the hardware stack. There are also several routines to handle special FORTRAN runtime operations such as PAUSE, STOP, I/O initialization, and I/O data transfers.

For example, the following FORTRAN program:

```
      C
      C     PROGRAM TO DEMONSTRATE THE CODE GENERATED BY
      C     THE RT-11 FORTRAN COMPILER.
      C
0001        DIMENSION RARRAY (10,10)        ! ALLOCATE A REAL*4 ARRAY
0002        I = (3*2 - 5) + I               ! ADD ONE TO I
0003        J = (I+100)*(N**2)              ! COMPUTE AN EXPRESSION
0004        A = 2.0                         ! ASSIGN A VALUE TO A REAL
0005        RARRAY(2,1) = RARRAY(1,1) + A ! SUM OF TWO REAL VALUES
0006        END
```

would generate object code that can be symbolically represented as follows (the storage map is included for reference):

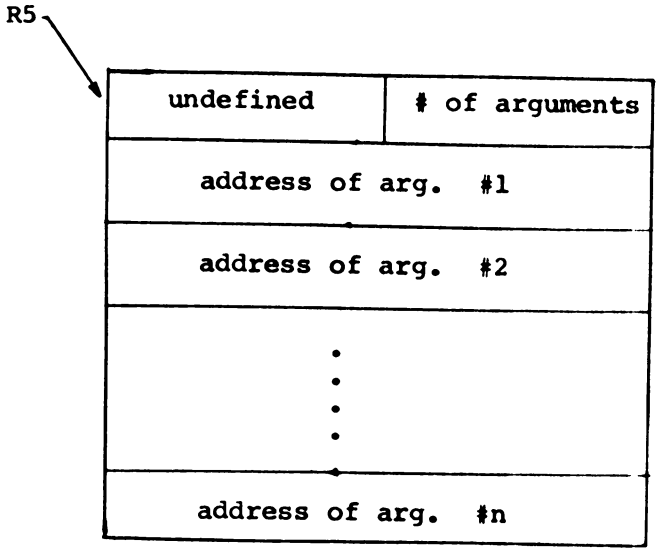
```
RT-11 FORTRAN IV          STORAGE MAP

NAME       OFFSET     ATTRIBUTES

RARRAY     000006     REAL*4         ARRAY (10,10)
I          000626     INTEGER*2      VARIABLE
J          000630     INTEGER*2      VARIABLE
N          000632     INTEGER*2      VARIABLE
A          000634     REAL*4         VARIABLE


RT-11 FORTRAN IV                     GENERATED CODE


ISN #0002

000640  ICI$M  000626                ; increment the integer whose address
                                     ; is 000626 (I)

ISN #0003

000644  MOI$MS  000626               ; move the value of integer I onto stack
000650  ADI$IS  #000144              ; add 100 to value on top of stack
000654  MOI$MS  000632               ; move the value of integer N onto stack
000660  MUI$MS  000632               ; and square it (multiply by itself)
000664  MUI$SS                       ; multiply (I+100) and (N**2)
000666  MOI$SM  000630               ; store value on top of stack in J


ISN #0004

000672  MOF$IM  #040400 000634       ; move an immediate floating constant
                                     ; (2.0) to A


ISN #0005

000700  MOF$MM  000006  000012       ; move RARRAY (1,1) to RARRAY (2,1)
000706  ADF$MM  000634  000012       ; and add A to RARRAY (2,1)


ISN #0006

000714  RET$                        ; return to RT-11 (exit from program)
```

2.3 SUBROUTINE LINKAGE

All instances of subprogram linkage are performed in the same manner, including linkage of user written FORTRAN subprograms, and Assembly language subprograms. Control is passed to the subprogram with the following instruction:

```
JSR     PC,routine
```

Register five points to an argument list having the following format:



Control is returned to the calling program by use of the instruction:

```
RTS     PC
```

An assembly language subroutine to find the sum of any number of integers using the following call:

CALL ADD(num_1,num_2,...,num_n,isum)

might look like the following:

```
            .TITLE    ADDER
            .GLOBL    ADD
ADD:        MOV       (R5)+,R0        ;GET # OF ARGUMENTS
            CLR       R1              ;PREPARE WORKING REG.
            DECB      R0              ;FIND # OF TERMS TO ADD
1$:         ADD       @(R5)+,R1       ;ADD NEXT TERM
            DECB      R0              ;DECREMENT COUNTER
            BNE       1$              ;LOOP IF NOT DONE
            MOV       R1,@(R5)+       ;RETURN RESULT
            RTS       PC              ;RETURN CONTROL
```

2.4 SUBPROGRAM REGISTER USAGE

A subprogram that is called by a FORTRAN program need not preserve any registers. However, the stack must be kept in sync: that is each 'push' onto the stack must be matched by a 'pop' from the stack prior to leaving the routine.

User-written assembly language programs that call FORTRAN subprograms must preserve any pertinent registers before calling the FORTRAN routine and restore the registers, if necessary, upon return.

Function subprograms return a single result in the registers. The register assignments for returning the different variable types are listed below:

Integer, Logical functions - result in R0

Real functions - high order result in R0, low order result in R1

Double Precision functions - result in R0-R3, lowest order result in R3

Complex functions - high order real result in R0, low order real result in R1, high order imaginary result in R2, low order imaginary result in R3

2.5 TRACEBACK FEATURE

RT-11 FORTRAN fatal runtime errors provide the traceback feature. This feature locates the actual program unit and line number of a runtime error. Immediately following the error message, the error handler will list the line number and program unit name in which the error occurred. If the program unit is a subroutine or function subprogram, the error handler will trace back to the calling program unit and display the name of that program unit and the line number where the call occurred. This process will continue until the calling sequence has been traced back to a specific line number in the main program. This allows the exact determination of the location of an error even if the error occurs in a deeply nested subroutine.

```
A=0.0
CALL SUB1(A)
CALL EXIT
END

SUBROUTINE SUB1(B)
CALL SUB2(B)
RETURN
END

SUBROUTINE SUB2(C)
CALL SUB3(C)
RETURN
END

SUBROUTINE SUB3(D)
E=1.0
F=E/D
RETURN
END
```

```
Trace back of Fatal Error:

          ?ERR 12 FLOATING ZERO DIVIDE

          IN    ROUTINE "SUB3   "    LINE 3
          FROM  ROUTINE "SUB2   "    LINE ?
          FROM  ROUTINE "SUB1   "    LINE 2
          FROM  ROUTINE ".MAIN."     LINE 2
```

Figure 2-1
The Traceback Feature

Note in Figure 2-1 that the line number in the traceback of routine 'SUB2' is simply a question mark (?). This is because the module was compiled with the /S switch in effect (see section 1.2.1).

2.6 VECTORED ARRAYS

Array vectoring is a process which decreases the time necessary to reference elements of a multidimensional array by using some additional memory to store the array.

Multidimensional arrays, which are actually stored sequentially in memory, require certain address calculations to determine the location of individual elements of the array. Typically, a mapping function is used to perform this calculation. For example to locate the element LIST(1,2,3) in array dimensioned LIST(4,5,6) a function equivalent to the following may be used. This function identifies a location as an offset from the origin of the array storage.

$$(s_1-1) + d_1 * (s_2 - 1) + d_1 * d_2 * (s_3 - 1) =$$
$$(\ 0\ ) + 4\ * (\ \ 1\ \ ) + \ 4\ *\ \ 5\ * (\ \ 2\ \ ) = 44$$

where s_i = subscript i
d_i = dimension i

Since such a mapping function requires multiplication operation(s), and since some PDP-11 hardware configurations do not have the MUL instruction, the compiler may 'vector' some arrays and thereby reduce execution time at the expense of memory storage.

If an array is vectored, a particular element in the array can be located by a simplified mapping function, without the need for multiplication. Instead, a table lookup is performed to determine the location of a particular element. For example, a vectored, two dimensional array B(5,5) automatically has associated with it a one dimensional vector that would contain relative pointers to each column of array B. The location of the element B(m,n), relative to the beginning of the array, could then be computed as:

```
Vector(n) + m
```

using only addition operations. Figure 2-2 graphically depicts the array vectoring process.

The compiler decides whether to vector a multidimensional array based on the ratio of the amount of space required to vector the array over the total storage space required by the array. If this ratio is greater than 25%, the array is not vectored and a standard mapping function is used instead. Arrays with adjustable dimensions are never vectored. Vectored arrays are noted as such in the storage map listing.

The Compiler /V switch can be used to suppress all array vectoring.

The amount of core required to vector an array can be computed as the sum of all array dimensions except the first. For example, the array X(50,10,30) requires 10+30=40 words of vector table. Note that the array V(5,100) requires 100 words of vector storage, whereas the array Y(100,5) requires 5 words of vector storge. It is therefore advantageous to place an array's largest dimension first.

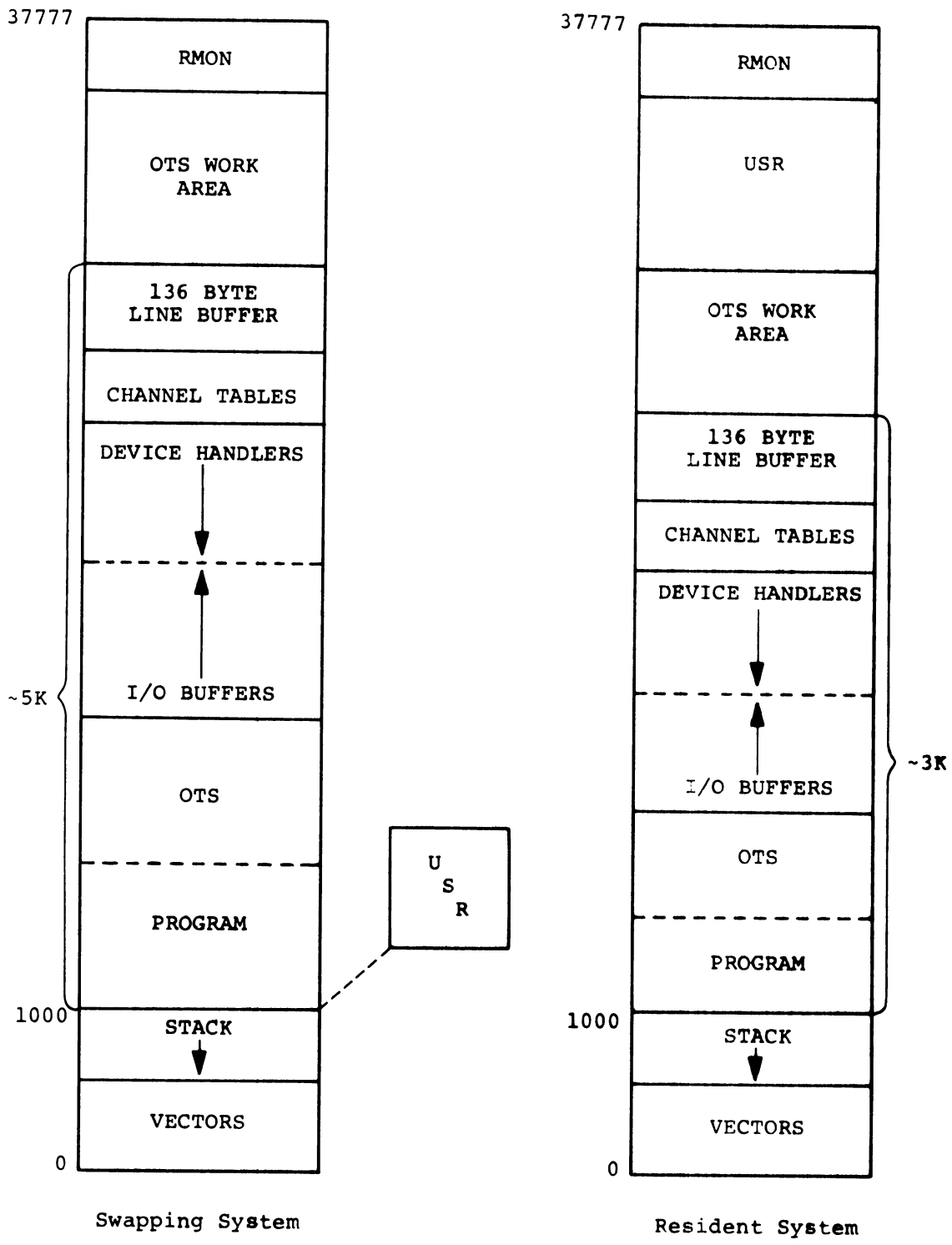


Figure 2-3
RT-11 8K System
Runtime Memory Organization

CHAPTER 3

RT-11 FORTRAN SPECIFIC CHARACTERISTICS

This chapter deals with information specific to RT-11 FORTRAN that is omitted from or contradicts information included in the PDP-11 FORTRAN Language Reference Manual.

It should be noted that deviations from FORTRAN syntax requirements outlined in the PDP-11 FORTRAN Language Reference Manual, even if acceptable in RT-11 FORTRAN, decrease the portability of the program, and may prohibit successful execution on another PDP-11 system.

3.1 VARIABLE NAMES

RT-11 FORTRAN allows variable names to extend past six characters in length. However, only the first six characters are significant and should be unique among all variable names in the program unit. If the /W switch is included in the Compiler command string, a warning diagnostic is given for each variable name which exceeds six characters in length.

3.2 INITIALIZATION OF COMMON VARIABLES

RT-11 FORTRAN allows any variables in COMMON, including blank COMMON, to be initialized in any program unit by use of the DATA statement.

3.3 CONTINUATION LINES

RT-11 FORTRAN does not place any limits on the number of continuation lines that a statement may contain.

3.4 DEFAULT LOGICAL UNIT - DEVICE/FILE ASSIGNMENTS

Listed in table 3-1 are the default logical unit - device and filename assignments. The default device assignments may be changed prior to execution by use of the monitor ASSIGN command. For example the monitor command:

```
.ASSIGN LP 7
```

connects logical unit 7 to the physical device line printer. The device and/or filename assignments may be changed at execution time by use of the ASSIGN system subroutine (see section B.2). Valid logical unit numbers other than those listed below (10-99) are assigned to the system default device, DK:. The default filename conventions hold for logical units not listed below, i.e. unit number 49 will have a default filename of FTN49.DAT.

Table 3-1
FORTRAN Logical Device Assignments

Logical Unit Number	Default Device	Default Filename
1	System disk, SY:	FTN1.DAT
2	System disk, DK:	FTN2.DAT
3	System disk, DK:	FTN3.DAT
4	System disk, DK:	FTN4.DAT
5	Terminal, TT:(Input)	FTN5.DAT
6	Line printer, LP:	FTN6.DAT
7	Terminal, TT:(Output)	FTN7.DAT
8	High-speed paper tape reader, PR:	FTN8.DAT
9	High-speed paper tape punch, PP:	FTN9.DAT

Although any combination of valid logical unit numbers may be used, there is an imposed maximum number of units which may be simultaneously active. By default, six logical units may be concurrently active. The number may be changed by use of the /N switch in the Compiler command string while compiling the main program unit (see section 1.2.1).

A formatted READ statement of the form:

```
READ f,list
```

is equivalent to:

```
READ(1,f)list
```

For all purposes these two forms function identically. Assigning logical unit number 1 to the terminal, for example, in both cases causes input to come from the terminal.

The ACCEPT, TYPE, and PRINT statements also have similar functional analogies. Assigning devices to logical units 5,7, and 6 affects respectively the ACCEPT, TYPE, and PRINT statements.

3.5 MAXIMUM RECORD LENGTHS

In RT-11 FORTRAN the line buffer allocated to temporarily store I/O records is by default 136 bytes long. This restricts all I/O records in formatted I/O statements and ENCODE and DECODE statements to a maximum of 136 characters. The size of this buffer, and consequently the maximum record length, may be changed by including the /R switch in the Compiler command string while compiling the main program unit. The maximum size of the line buffer is 4095 bytes (7777 octal).

3.6 STATEMENT ORDERING RESTRICTIONS

RT-11 FORTRAN does not impose as strict statement ordering requirements as those outlined in the PDP-11 FORTRAN Language Reference Manual. There are only three statement ordering requirements that must be met:

1) In a Subprogram, the first non-comment line must be a FUNCTION, SUBROUTINE or BLOCK DATA statement.
2) The last line in a program unit must be an END sentinel.
3) Statement Functions must be defined before they are referenced.

However, if the statement ordering requirements as outlined in the PDP-11 FORTRAN Language Reference Manual are not followed, and if the /W Compiler switch (enable warning diagnostics) is included in the Compiler command string, a warning diagnostic will be included with the source listing.

3.7 DIRECT-ACCESS I/O

3.7.1 DEFINE FILE

The first parenthesized argument in a DEFINE FILE statement specifies the length, in records, of the direct-access file being initialized. However, if the statement is part of a file creation procedure, this value may not be readily available. RT-11 FORTRAN allows some extra flexibility in this situation. A file length specification of zero records causes a large contiguous file to be allocated initially and the unused portion to be automatically de-allocated when the file is closed. The "END=" construction is particularly useful in this situation for determining the actual length of the file.

3.7.2 Creating Direct-Access Files

The first I/O operation performed on a direct-access file during file creation must be a WRITE operation. A READ or FIND operation under such circumstances produces a fatal error condition.

3.8 INPUT/OUTPUT FORMATS

3.8.1 Formatted I/O

Formatted input/output transfers 7 bit ASCII characters packed 1 character per byte. This form of input/output correctly transfers ASCII files only. An attempt to use formatted input/output to transfer binary files causes a loss of information.

3.8.2 Unformatted I/O

Unformatted input/output is actually a formatted binary input/output. Each binary record includes record header and trailer information. Unformatted input/output transfers full 16 bit words of information. Generally, unformatted input operations correctly function only on a file created using FORTRAN unformatted output operations.

3.8.3 Direct-Access I/O

Direct-access input/output is true unformatted binary input/output. No record header or trailer information is added on output operations and none is expected on input operations. Direct-access input/output transfers fixed length unformatted binary records. Direct-access input/output is the only type of input/output that correctly transfers information to or from a binary file.

3.9 MIXED MODE COMPARISONS

When comparing a single precision number to a double precision number, the double precision number may appear to be greater than the single precision number in magnitude even if they should be equal. For example:

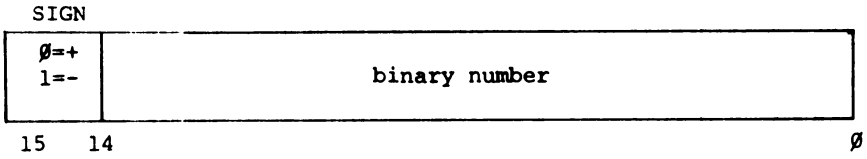
```
DOUBLE PRECISION D

A=55.1

D=55.1D0

IF(A.LT.D)STOP
```

In the example above A compares less than D. This is due to the fact that 55.1 is a repeating binary fraction. Before the comparison, the 24 bit fractional (mantissa) part of A is extended with 32 zero bits. These low order 32 bits are now less than the low order 32 bits of D, and D therefore compares greater than A.

APPENDIX A

FORTRAN DATA REPRESENTATION

A.1 INTEGER FORMAT



Integers are stored in a two's complement representation. If the /T compiler switch (see section 1.2.1) is used an integer is assigned two words, although only the high-order word (i.e., the word having the lower address) is significant. By default integers will be assigned to a single storage word. Explicit length integer specifications (INTEGER*2 and INTEGER*4) will always take precedence over the setting of the /T switch. Integer constants must lie in the range -32767 to +32767. For example:

$+22 = 000026_8$
$-7 = 177771_8$

A.2 FLOATING-POINT FORMATS

The exponent for both 2-word and 4-word floating-point formats is stored in excess 128 (200_8) notation. Binary exponents from -128 to +127 are represented by the binary equivalents of 0 through 255 (0 through 377_8). Fractions are represented in sign-magnitude notation with the binary radix point to the left. Numbers are assumed to be normalized and, therefore, the most significant bit is not stored because it is redundant (this is called "hidden bit normalization"). This bit is assumed to be a 1 unless the exponent is 0 (corresponding to 2^{-128}) in which case it is assumed to be 0. The value 0 is represented by two or four words of zeros. For example, +1.0 would be represented by:

```
40200
0
```

in the 2-word format, or:

```
40200
0
0
0
```

in the 4-word format. -5 would be:

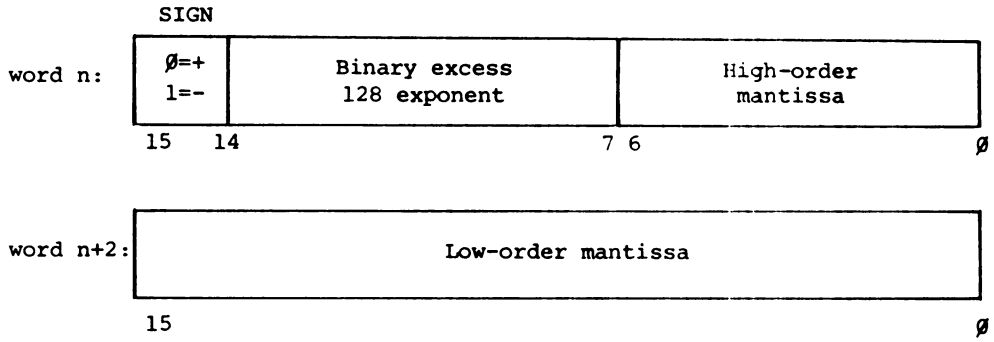
```
140640
0
```

in the 2-word format, or:

```
140640
0
0
0
```

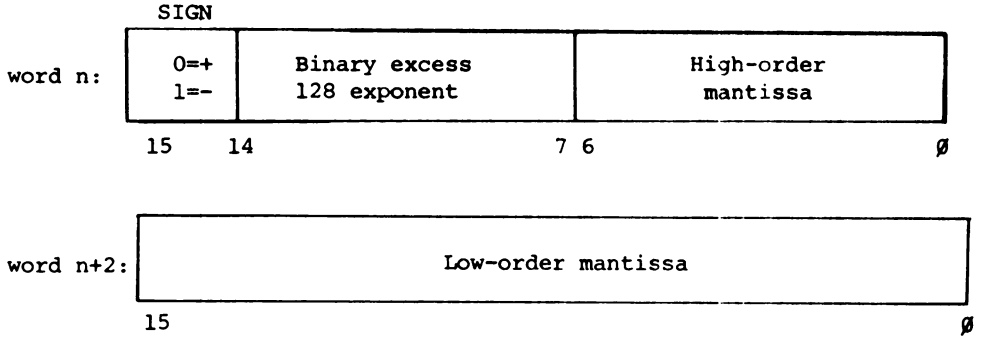
in the 4-word format.

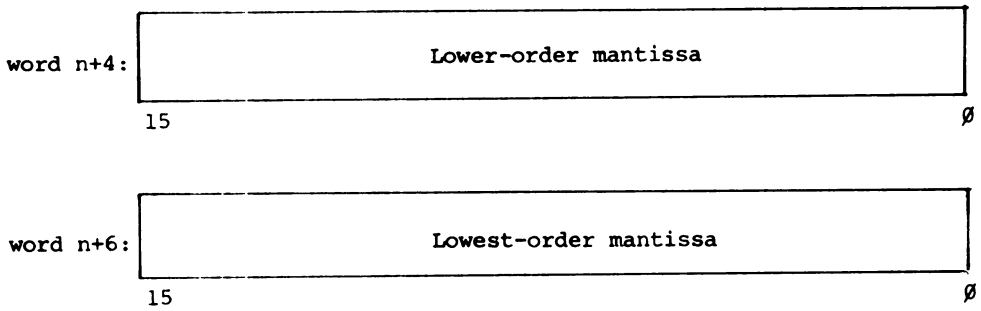
A.2.1 Real Format (2-Word Floating Point)



Since the high-order bit of the mantissa is always 1, it is discarded, giving an effective precision of 24 bits, or approximately 7 digits of accuracy. The magnitude range lies between approximately 1.0×10^{-39} and 1.0×10^{38}.

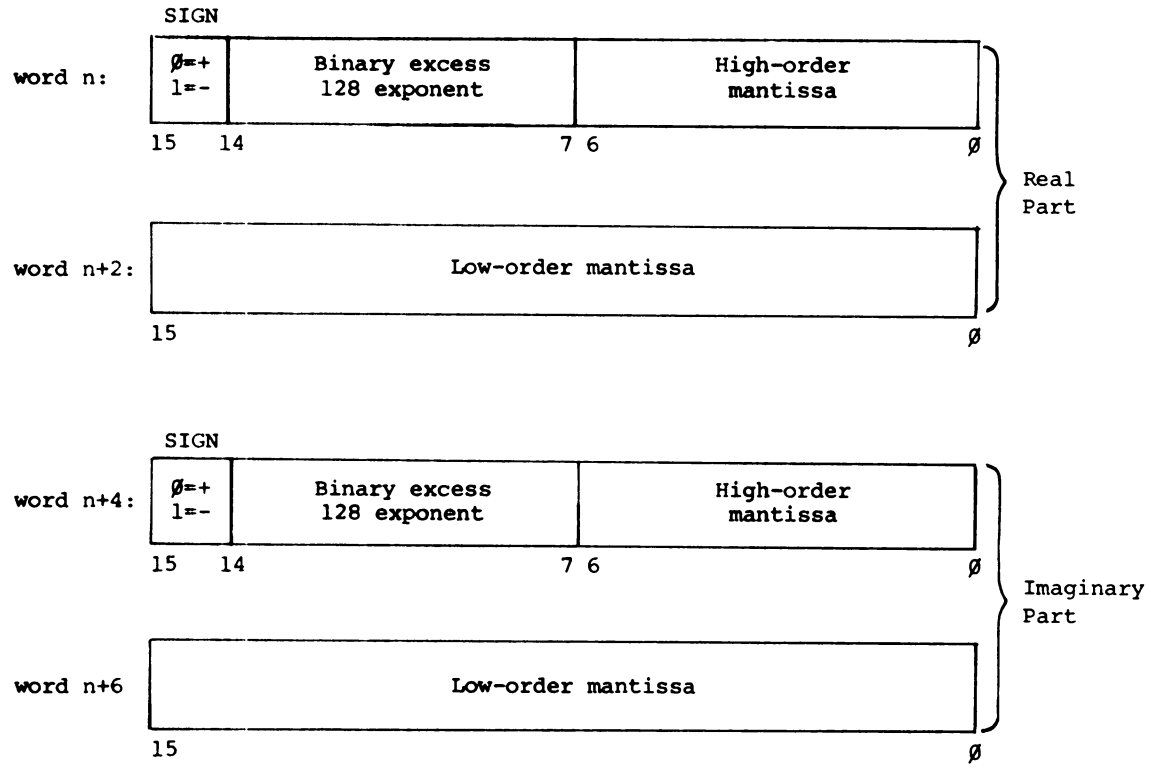
A.2.2 Double Precision Format (4-Word Floating Point)



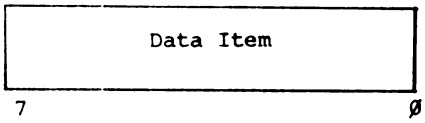


The effective precision is 56 bits or approximately 17 decimal digits of accuracy. The magnitude range lies between 1.0×10^{-39} and 1.0×10^{38}.

A.2.3 Complex Format

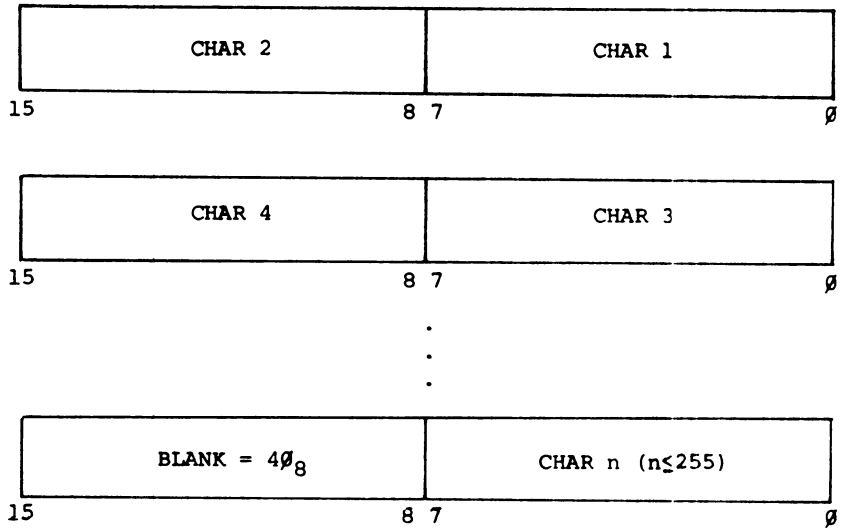


A.3 LOGICAL*1 FORMAT



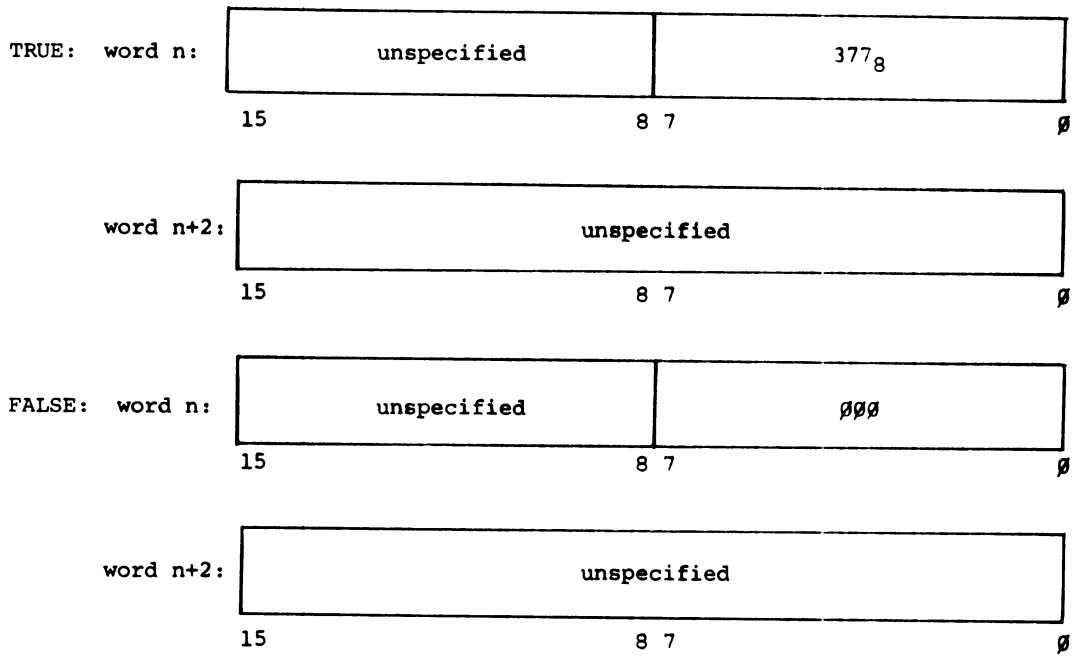
The range of numbers from +127 to -128 can be represented in LOGICAL*1 Format. LOGICAL*1 array elements are stored in adjacent bytes.

A.4 HOLLERITH FORMAT



Hollerith constants are stored internally one character per byte. Hollerith values are padded on the right with blanks to fill the associated data item if necessary. Hollerith constants can only be used in DATA, FORMAT, and CALL statements. Only the quoted form of Hollerith constants can be used in STOP and PAUSE statements.

A.5 LOGICAL FORMAT



Logical (LOGICAL*4) data items are treated as LOGICAL*1 values for use with arithmetic and logical operators. Any non-zero value in the low order byte is considered to have a logical value of true when tested by a logical IF statement.

APPENDIX B

SYSTEM SUBROUTINES

B.1 SYSTEM SUBROUTINE SUMMARY

Like the functions intrinsic to the FORTRAN system, there are subroutines in the FORTRAN library which the user may call in the same manner as a user-written subroutine. These subroutines are:

ASSIGN	Allows specification at run-time of filename or device and filename to be associated with a FORTRAN logical unit number.
RANDU RAN	Returns a random real number with a uniform distribution between 0 and 1.
EXIT	Terminates the execution of a program and returns control to the monitor.
USEREX	Allows specification of a routine name to which control widl be passed as part of the exit operation. This allows the user to disable interrupts enabled by non-FORTRAN routines.
DATE	Returns a 9-byte string containing the ASCII representation of the current date.
IDATE	Returns three integer value representing the current month, day and year.
SETERR	Allows the user to set a count specifying the number of times to ignore a certain error condition.

B.2 ASSIGN

The Assign subroutine allows the association of device and/or filename information with a logical unit number. The ASSIGN call, if present, must be executed before the logical unit is opened for I/O operations (by READ or WRITE) for sequential access files, or before the associated DEFINE FILE statement for random-access files. The assignment remains in effect until the end of the program or until the file is closed by an ENDFILE and a new CALL ASSIGN performed. The call to ASSIGN has a general form as follows:

```
CALL ASSIGN(n, name, icnt, mode, control, numbuf)
```

CALL ASSIGN requires only the first argument, all others are optional, and if omitted are replaced by the default values as noted in the argument descriptions. However, if any argument is to be included, all arguments that precede it must also be included.

A description of the arguments to the ASSIGN routine follows:

n — logical unit number expressed as an integer constant or variable

name — Hollerith or literal string containing any standard RT-11 device/filename specification. If the device is not specified, then the device remains unchanged from the default assignments, or the monitor ASSIGN command. If a filename is not specified, the default names as described in section 3.4 are used. There are three switches which may optionally be included in the file specification. The switches are:

- /N — Specifies no carriage control translation. This switch, if present, will override the value of the 'control' argument.
- /C — Specifies carriage control translation. This switch, if present, will override the value of the 'control' argument.
- /B:n — Specifies the number of buffers, n, to use for I/O operations. The single argument, n, should be of value 1 or 2. This switch, if present, will override the value of the 'numbuf' argument.

If name is simply a device specification e.g. 'DK:', the device will be opened with an RT-11 non-file structured Lookup, and the device will be treated in a non-file structured manner. Indescriminate use of this feature on directory devices such as disk or Dectape can be dangerous. See the RT-11 system Reference Manual, Monitor chapter.

icnt — specifies the number of characters in the string 'name'. If 'icnt' is zero, the string 'name' will be processed until the first blank character is encountered. If 'icnt' is negative, program execution will be temporarily suspended, a prompt character (*) will be sent to the terminal, and a device and/or filename specification, with the same form as 'name' above, terminated by a carriage return, will be accepted from the keyboard.

mode — specifies the method of opening the file on this unit. This argument should be one of the following:

- 'RDO' — the file will be read only. A fatal error occurs if a FORTRAN WRITE is attempted on this unit. The file is opened with an RT-11 LOOKUP.

'NEW' the file is being created. The file is opened with an RT-11 ENTER.

'OLD' the file already exists. The file is opened with an RT-11 LOOKUP.

'SCR' the file is only to be used temporarily and will be deleted when it is closed.

If this argument is omitted the default is determined by the first I/O operation performed on that unit. If a WRITE operation is the first I/O operation performed on that unit, 'NEW' is assumed. If a READ operation is the first I/O operation performed on that unit, 'OLD' is assumed.

control specifies whether carriage control translation is to occur. This argument should be one of the following:

'NC' all characters are output exactly as specified.

'CC' the character in column one of all output records is treated as a carriage control character. (see section 6.2.16 of the PDP-11 FORTRAN Language Reference Manual)

Note that if not specifically changed by the CALL ASSIGN subroutines, by default, the terminal and line printer assume 'CC', and all other devices assume 'NC'.

numbuf specifies the number of internal buffers to be used for the I/O operation. A value of 1 is appropriate under normal circumstances. If this argument is omitted, one internal buffer is used.

If only the unit number argument is specified, all previously specified device and/or filename information concerning that unit is dissociated from the unit number, and the default conditions become effective.

For example in the following situation:

```
CALL ASSIGN(6,'TT:')
WRITE(6,-) ....
ENDFILE 6
WRITE(6,-) ....
```

both WRITE operations will be performed on the terminal. However, in the following situation:

```
CALL ASSIGN(6,'TT:')
WRITE(6,-) ....
ENDFILE 6
CALL ASSIGN(6)
WRITE(6,-) ....
```

will cause the first WRITE operation to be performed on the terminal, and the second on the line printer.

B.3 RANDU,RAN

The random number generator can be called as a subroutine, RANDU, or as an intrinsic function, RAN. The subroutine call is performed as follows:

```
CALL RANDU(i1 ,i2 ,x)
```

where i_1 and i_2 are previously defined integer variables and x is the real variable name in which is returned a random number between 0 and 1. i_1 and i_2 should be initially set to 0. I_1 and i_2 are updated to a new generator base during each call. Resetting i_1 and i_2 to 0 repeats the random number sequence. The values of i_1 and i_2 have a special form; only 0 or saved values of i_1 and i_2 should be stored in these variables.

Use of the random number subroutines is similar to the use of the RAN function where:

```
x=RAN(i1 ,i2 )
```

is the functional form of the random number generator.

B.4 EXIT

A call to the EXIT subroutine, in the form:

```
CALL EXIT
```

is equivalent to the STOP statement. It causes program termination, closes all files, and returns to the RT-11 monitor.

B.5 USEREX

USEREX is a subroutine which allows specification of a routine to which control will be passed as part of program termination. This allows disabling of interrupts enabled in non-FORTRAN routines. If these interrupts are not disabled prior to program exit the integrity of the RT-11 operating system cannot be assured. The form of the subroutine call is:

```
CALL USEREX (name)
```

Where 'name' is the routine to which control will be passed and should appear in an EXTERNAL statement somewhere in the program unit. Control is transferred with a JMP instruction after all procedures

required for FORTRAN program termination have been completed. The transfer of control takes place instead of the normal return to the RT-11 monitor, so if the user desires to have control passed back to the RT-11 Monitor, the routine specified by USEREX must perform the proper exit procedures itself.

B.6 DATE

The DATE subroutine can be used in a FORTRAN program to obtain the current date as set within the system. The DATE subroutine is called as follows:

```
CALL DATE(array)
```

where array is a predefined array able to contain a 9-byte string. The array specification in the call may be expressed as the array name alone:

```
CALL DATE(a)
```

in which the first three elements of the real array a are used to hold the date string, or a':

```
CALL DATE(a(i))
```

which causes the 9-byte string to begin at the i(th) element of the array a.

The date is returned as a 9-byte (9-character) string in the form:

```
dd-mmm-yy
```

where:

dd is the 2-digit date
mmm is the 3-letter month specification
yy is the last two digits of the year

For example:

```
15-NOV-75
```

In the case where the array is a real array, 4-1/2 words are used to contain the data string with the remaining array storage being untouched. Therefore, the date string is stored in the first nine bytes in the elements a(i), a(i+1), and a(i+2). The last three bytes of a(i+2) are untouched and should be made blank by the user if he intends to print the date with a 3A4 format.

B.7 IDATE

IDATE returns three integer values representing the current month, day, and year. The call has the form:

```
CALL IDATE(i,j,k)
```

If the current date were March 19,1975 the values of the integer variables upon return would be:

```
i = 3
j = 19
k = 75
```

B.8 SETERR

SETERR allows the user to specify the disposition of certain OTS detected error conditions. Only OTS error diagnostics 1 - 16 should be changed from their default error classification (see section C.2). If errors 0 or 20 - 63 are changed from the default classification of FATAL, execution will continue but in an undetermined state. The form of the call is:

```
CALL SETERR (number, ncount)
```

Where 'number' is an integer variable or expression specifying the OTS error number (see section C.2), and 'ncount' is an integer variable or expression with one of the following values:

value	meaning
0	ignore all occurrences of the error
1	first occurrence will be fatal
2-127	the nth occurrence will be fatal the first n-1 will be logged on the user terminal
255	Ignore all occurrences after logging them on the user terminal

APPENDIX C

FORTRAN ERROR DIAGNOSTICS

C.1 COMPILER ERROR DIAGNOSTICS

The RT-11 FORTRAN Compiler, while reading and processing the FORTRAN source program, can detect syntax errors (or errors in general form) such as unmatched parentheses, illegal characters, unrecognizable key words, missing or illegal statement parameters.

The error diagnostics are generally clear in specifying the exact nature of the error. In most cases, a check of the general form of the statement in question as described in the PDP-11 FORTRAN Language Reference Manual will help determine the location of the error.

Some of the most common causes of syntax errors, however, are typing mistakes. A typing mistake can sometimes cause the Compiler to give very misleading error diagnostics. The user should be careful of the following common typing mistakes:

1. Missing commas or parentheses in a complicated expression or FORMAT Statement.

2. Misspelling of particular instances of variable names. If the Compiler does not detect this error (it usually cannot), execution may also be affected.

3. An inadvertent line continuation signal on the line following the statement in error.

4. If the user terminal does not clearly differentiate between 0 (zero) and O, what may appear to be identical spellings of variable names may not appear so to the Compiler, and what may appear to be a constant expression may not appear so to the Compiler.

If any errors were detected in a compilation, the message:

```
ERRORS DETECTED: n
```

will be printed on the console terminal; n is the number of errors, not including warnings, detected by the compiler.

The next three sections describe the initial phase and secondary phase error diagnostics and the fatal FORTRAN Compiler error diagnostics.

C.1.1 Errors Reported by the Initial Phase of the Compiler

Some of the easily recognizable FORTRAN syntax errors are detected by the initial phase of the Compiler. These errors are reported in the source program listing. They are not reported if a source listing is not requested.

The error diagnostics are printed after the source statement to which they apply (the L error diagnostic is an exception). The general form of the diagnostic is as follows:

```
***** c
```

Where c is a code letter whose meaning is described below:

INITIAL PHASE ERROR DIAGNOSTICS

Code Letter	Description
B	Columns 1-5 of continuation line not blank. Columns 1-5 of a continuation line must be blank except for a possible 'D' in column 1.
C	Illegal continuation. Comments cannot be continued and the first line of any program unit cannot be a continuation line.
E	Missing END statement. An END statement is supplied by the Compiler if end-of-file is encountered.
H	Hollerith string or quoted literal string longer than 255 characters or longer than the remainder of the statement.
I	Non-FORTRAN character used. The line contains a character that is not in the FORTRAN character set and is not in a Hollerith string or comment line.
K	Illegal statement label definition. Illegal (non-numeric) character in statement label.
L	Line too long. There are more than 80 characters in a line. Note: this diagnostic is issued before the line containing too many characters.
M	Multiply defined label.
P	Statement contains unbalanced parentheses.
S	Syntax error. Multiple equal signs, etc. Statement not of the general FORTRAN statement form.
U	Statement could not be identified as any legal FORTRAN statement.

C.1.2 Errors Reported by Secondary Phases of the Compiler

Those Compiler error diagnostics not reported by the initial phase of the Compiler will appear immediately after the source listing and immediately before the storage map. Since the diagnostics appear after the entire source program has been listed, they must reference the statement to which they apply by using the internal sequence numbers assigned by the Compiler.

The general form of the diagnostic is:

```
IN LINE nnnn MSG#m text
```

Where nnnn is the internal sequence number of the statement in question, m is an integer constant specifying the error number, and text is a short description of the error.

Below, listed alphabetically, are the error diagnostics. Included with each diagnostic is a brief explanation, and in most cases, a reference to the PDP-11 FORTRAN Language Reference Manual section that contains information to help correct the error.

The notation **** signifies that a particular variable name or statement label will appear at that place in the text.

SECONDARY PHASE ERROR DIAGNOSTICS

ADJUSTABLE DIMENSIONS ILLEGAL FOR ARRAY ****
All arrays must be dimensioned with integer constants except as specified in section 7.3.1.

ARRAY **** HAS TOO MANY DIMENSIONS
An array can have up to seven dimensions. See section 7.3.

ATTEMPT TO EXTEND COMMON BACKWARDS
While attempting to equivalence arrays in COMMON, an attempt was made to extend COMMON past the recognized beginning of COMMON storage. See section 7.5.2.

COMMON BLOCK EXCEEDS MAXIMUM SIZE
An attempt was made to allocate more space to COMMON than is physically addressable (>32k words).

DANGLING OPERATOR
An operator (+,-,*,/, etc.) is missing an operand. Example: I=J+.

DEFECTIVE DOTTED KEYWORD
A dotted relational operator was not recognized. Also, possible misuse of decimal point. See section 2.5.2.

DO TERMINATOR **** PRECEDES DO STATEMENT
The statement specified as the terminator of a DO loop must come after the DO statement. See section 4.3.

EXPECTING LEFT PARENTHESES AFTER ****
An array name or Function name reference is not followed by a left parenthesis.

EXTRA CHARACTERS AT END OF STATEMENT

All the necessary information for a syntactically correct FORTRAN statement has been found on this line, but more information exists. Possibly due to inadvertent continuation signal on next line, or a missing comma.

FLOATING CONSTANT TOO SMALL

A floating constant in an expression is too close to zero to be represented in the internal format. Use zero if possible. See section 2.2.2.

ILLEGAL ADJACENT OPERATOR

Two operators (*,/, logical operators, etc.) are illegally placed next to each other. Example: I/*J. See sections 2.5.1.2, 2.5.2.1, and 2.5.3.2.

ILLEGAL ELEMENT IN I/O LIST

An item, expression, or implied DO specifier in an I/O list is of illegal syntax.

ILLEGAL DO TERMINATOR STATEMENT ****

A DO statement terminator must not be a GO TO, arithmetic IF, RETURN, or DO statement or logical IF containing one of these statements. See section 4.3.

ILLEGAL STATEMENT ON LOGICAL IF

The statement contained in a logical IF must not be another logical IF or DO statement. See section 4.2.2.

ILLEGAL TYPE FOR OPERATOR

An illegal variable type has been used with an exponentiation or logical operator. See sections 2.5.1.5, and 2.5.2.3.

ILLEGAL USAGE OF OR MISSING LEFT PARENTHESIS

A left parenthesis was required but not found, or a variable reference or constant is illegally followed by a left parenthesis.

INTEGER OVERFLOW

An integer constant or expression value must not fall outside the range -32767 to +32767. See section 2.2.1.

INVALID COMPLEX CONSTANT

A complex constant has been improperly formed. See section 2.2.4.

INVALID DIMENSIONS FOR ARRAY

An attempt was made while dimensioning an array to explicitly specify zero as one of the dimensions. See section 7.3.

INVALID DO TERMINATOR ORDERING AT LABEL ****

Do loops are incorrectly nested. See section 4.3.1.

INVALID EQUIVALENCE

Illegal equivalence, or equivalence that is contradictory to a previous equivalence. See section 7.5.

INVALID FORMAT SPECIFIER
A format specifier is not the label of a FORMAT statement or an array name. See section 5.1.

INVALID IMPLICIT RANGE SPECIFIER
Illegal implicit range specifier, i.e. non-alphabetic specifier, or specifier range is in reverse alphabetic order. See section 7.1.

INVALID LOGICAL UNIT
A logical unit reference must be an integer variable or constant in the range 1 to 99. See section 5.1.1.

INVALID OCTAL CONSTANT
An octal constant is too large or contains a digit other than 0-7. See section 2.2.5.

INVALID OPTIONAL LENGTH SPECIFIER
A data type declaration optional length specifier is illegal. For example, REAL*4 and REAL*8 are legal, but REAL*6 is not. See section 7.2.

INVALID RADIX50 CONSTANT
Illegal character detected in a RADIX50 constant. See section 2.2.8.

INVALID RECORD FORMAT
The third parenthetical argument in a DEFINE FILE statement must be the single character U. See section 5.5.3.

INVALID STATEMENT IN BLOCK DATA
It is illegal to have any executable or FORMAT statements in a BLOCK DATA Subprogram. See section 8.1.4.

INVALID STATEMENT LABEL REFERENCE
Reference has been made to a statement number that is of illegal construction. GO TO 999999 is illegal since the statement number is too long. See section 1.5.3.

INVALID SUBROUTINE OR FUNCTION NAME
A name used in a CALL statement or function reference is not valid. Example: use of an array name in a CALL statement routine name reference.

INVALID TARGET FOR ASSIGNMENT
The left side of an arithmetic assignment statement is not a variable name or array element reference. See section 3.1.

INVALID TYPE SPECIFIER
An unrecognizable data type was used. See section 7.1.

INVALID USAGE OF FUNCTION OR SUBROUTINE NAME
a function name cannot appear in a DIMENSION, COMMON, DATA, EQUIVALENCE, or Data Type Declaration statement. See section 8.1.2.

INVALID VARIABLE NAME
A variable name contains an illegal character. See section 2.3.

LABEL ON DECLARATIVE STATEMENT
It is illegal to place a label on a declarative statement. See section 7.2.

MISSING ASSIGNMENT OPERATOR
The first operator seen in an arithmetic assignment statement was not an equal sign (=). Example: I+J=K. See section 3.1.

MISSING COMMA
The comma delimiter was expected but was not found. See the section of the FORTRAN Reference Manual that describes the general form of the statement in question.

MISSING DELIMITER IN EXPRESSION
Two operands have been placed next to each other in an expression with no operator between them.

MISSING LABEL
Expecting a statement label but one was not found. Example: ASSIGN J TO I. A valid statement label reference should precede 'TO' but does not.

MISSING RIGHT PARENTHESIS
Expecting a right parenthesis but one was not found. Example: READ(5,100,). The first non-blank character after the format reference should be a right parenthesis but is not.

MISSING QUOTATION MARK
In a FIND statement, the logical unit number and record number must be separated by a single quotation mark. See section 5.5.4.

MISSING VARIABLE
Expecting a variable, but one was not found. Example: ASSIGN 100 TO 1. A variable name should follow the 'TO' but one does not.

MISSING VARIABLE OR CONSTANT
Looking for an operator (variable or constant) but found a delimiter (comma, parenthesis, etc.). Example: WRITE(). A unit number should follow the open parenthesis, but a delimiter (close parenthesis) is encountered instead.

MODES OF VARIABLE **** AND DATA ITEM DIFFER
The data type of each variable and its associated data list item must agree in a DATA Statement. See section 7.7.

MULTIPLE DECLARATION FOR VARIABLE ****
A variable cannot appear in more than one data type declaration statement or dimensioning statement. See section 7.2.

NUMBER IN FORMAT STATEMENT NOT IN RANGE
An integer constant in a FORMAT statement is greater than 255 or is zero. See section 6.1.

PARENTHESES NESTED TOO DEEPLY
Group repeats in a FORMAT Statement have been nested too deeply. See section 6,2,15.

P-SCALE FACTOR NOT IN RANGE -127 TO +127
P-scale factors must fall in the range -127 to +127. See section 6.2.14.

REFERENCE TO INCORRECT TYPE OF LABEL ****
A statement label reference that should be a label on a FORMAT statement is not such a label, or a statement label reference that should be a label on an executable statement is not such a label.

REFERENCE TO UNDEFINED STATEMENT LABEL
A reference has been made to a statement number that has not been defined anywhere in the program unit.

STATEMENT MUST BE UNLABELED
A DATA, SUBROUTINE, FUNCTION, BLOCK DATA, arithmetic statement function definition, or declarative statement must not be labeled.

STATEMENT TOO COMPLEX
An arithmetic statement function has more than 10 dummy arguments. Or the statement is too long to compile. Break it up into 2 or more smaller statements. See section 8.1.1.

SUBROUTINE OR FUNCTION STATEMENT MUST BE FIRST
A SUBROUTINE, FUNCTION or BLOCK DATA Statement, if present, must be the first statement in a program unit. See section 8.1.2.

SUBSCRIPT OF ARRAY **** NOT IN RANGE
Array subscripts that are constants or constant expressions are checked to see if they are within the bounds of the array's dimensions.

SYNTAX ERROR
Check the general form of the statement with the general form outlined in the Language Reference Manual section that describes that type of statement.

TARGET MUST BE ARRAY
The third argument in an ENCODE or DECODE statement must be an array name. See section 5.8.

SYNTAX ERROR IN INTEGER OR FLOATING CONSTANT
An integer or floating constant has been incorrectly formed. For example, 1.23.4 is an illegal floating constant because it contains two decimal points. See section 2.2.

UNLABELED FORMAT STATEMENT
All FORMAT Statements must be labeled. See section 6.1.

USAGE OF VARIABLE **** INVALID

An attempt was made to EXTERNAL a common variable, an array variable, or a dummy argument. Or an attempt was made to place in COMMON a dummy argument or external name. See sections 7.4 and 7.6.

VARIABLE **** INVALID IN ADJUSTABLE DIMENSION

A variable used as an adjustable dimension must be an integer dummy argument in the subprogram unit. See section 7.3.1.

WRONG NUMBER OF SUBSCRIPTS FOR ARRAY ****

An array reference does not have the same number of subscripts as specified when the array was dimensioned.

C.1.3 Warning Diagnostics

Warning diagnostics report conditions which are not true error conditions, but which may be potentially dangerous at execution time, or which may present compatibility problems with FORTRAN Compilers running on other PDP-11 Operating Systems. The warning diagnostics are normally suppressed, but may be enabled by use of the /W Compiler switch. The form and placement of the warning diagnostics are the same as those for the secondary phase error diagnostics (see section C.1.2) except that the line number reference is replaced with '%WARNING%'. A listing of the warning diagnostics follows:

ADJUSTABLE DIMENSIONS ILLEGAL FOR ARRAY ****

Adjustable arrays must be parameter arrays in a subprogram, and the adjustable dimensions must be integer dummy arguments in the subprogram. Any variation from this rule will cause a dimension of 1 to be used and this warning message to be issued.

NON-STANDARD STATEMENT ORDERING

Although the RT-11 FORTRAN Compiler has less-severe statement ordering requirements than those outlined in chapter 7 of the PDP-11 FORTRAN Language Reference Manual, non-adherence to the stricter requirements may cause error conditions on other FORTRAN Compilers. See section 3.6 of this document.

VARIABLE **** IS NOT WORD ALIGNED

Placing a non-LOGICAL*1 variable or array after a LOGICAL*1 variable or array in COMMON or equivalencing non-LOGICAL*1 variables or arrays to logical*1 variables or arrays may cause this condition. An attempt to reference the variable at runtime will cause an error condition. See sections 7.4.1 and 7.5.3 of the PDP-11 FORTRAN Language Manual.

VARIABLE **** NAME EXCEEDS SIX CHARACTERS

A variable name of more than six characters was specified. The first six characters were used as the true variable name. Other FORTRAN Compilers may treat this as an error condition. See section 3.1 of this document.

C.1.4 Fatal Compiler Error Diagnostics

Listed below are the fatal Compiler error diagnostics. These diagnostics, which are sent directly to the terminal, report hardware error conditions, conditions which may require rewriting of the source program, and conditions which may require attention from DEC Software Support. The form of the diagnostic is:

```
FATAL ERROR n
```

where n is an error code having one of the following values:

Code	Meaning
C	Constant subscript overflow. Too many constant subscripts have been employed in a statement. SOLUTION - simplify the statement
O	Unrecoverable error occurred while the Compiler was writing the object file (.OBJ). Possibly, output file space is not large enough. SOLUTION - rectify hardware problem, or make more space for output.
P	Optimizer push down overflow - statement too complex, or too many common subexpressions occurred in one basic block of a program. SOLUTION - simplify complex statements; report the error to your local software support representative.
R	Unrecoverable hardware error occurred while the Compiler was reading source file. SOLUTION - rectify hardware problem.
S	Subexpression stack overflow - statement too complex Attempt to compile a statement which would overflow the runtime stack at execution time. SOLUTION - simplify complex statements.
T	Core Overflow SOLUTION - break up program into subprograms or compile on larger machine.
W	Unrecoverable error occurred while the Compiler was writing listing file. Possibly, listing file space is not large enough. SOLUTION - rectify hardware problem, or make more space for listing file.
Y	Code generation stack overflow - statement too complex. SOLUTION - simplify complex statements.

Z Compiler error

SOLUTION - report this error to your local software support representative. Please include program listing.

C.2 OBJECT TIME SYSTEM ERROR DIAGNOSTICS

The Object Time System detects certain I/O, arithmetic, and system failure error conditions and reports them on the user terminal. These error diagnostics are printed in either a long or short form.

The short form of the message appears as:

?ERR nn

where nn is a decimal error identification number.

The long form of the message appears as:

?ERR nn text

where nn is a decimal error identification number and
text is a short error description.

The default message length is long. The short message error module may be linked to the program instead by using the /I Linker switch (see the Linker chapter of the RT-11 System Reference Manual). The module named $SHORT should be included from the FORTRAN library.

There are four classes of OTS error conditions. Each error condition is assigned to one of these classes. An error condition classification for the error codes 1-16 can be changed by using the System Subroutine SETERR. (See section B.8). Error codes 0 and 20-63 should not be changed from their FATAL classification or undeterminable results will occur. The classifications are:

IGNORE The error is detected but no error message is sent to the terminal. Execution continues.

WARNING The error message is sent to the terminal and execution continues.

FATAL The error message is sent to the terminal and execution is terminated.

COUNT:n The error message is sent to the terminal and execution continues until the nth occurrence of the error, at which time the error will be treated as FATAL.

If a program is terminated by a fatal error condition active files are not closed. When control is returned to the RT-11 monitor a CLOSE command may be given to close all active files, although some of the output to these active files may have been lost.

The OTS error diagnostics are listed below along with the error type and a brief explanation where necessary.

Error number	Error type	Message
0	FATAL	NON-FORTRAN ERROR CALL A TRAP has occurred with an unrecognizable error code.
1	FATAL	INTEGER OVERFLOW During an arithmetic operation an integer's value has exceeded 32767 in magnitude.
2	FATAL	INTEGER ZERO DIVIDE During an integer mode arithmetic operation an attempt was made to divide by zero.
3	FATAL	COMPILER GENERATED ERROR If an attempt is made to Link and run an object file generated by the FORTRAN Compiler, when the Compiler detected and reported error conditions, this diagnostic will be given if the statement in error is ever executed.
4	WARNING	COMPUTED GO TO OUT OF RANGE The integer variable or expression in a computed GO TO statement was less than 1 or greater than the number of statement label references in the list. Control is passed to the next executable statement (See section 4.1.2 of the PDP-11 FORTRAN Language Reference Manual).
5	COUNT:3	INPUT CONVERSION ERROR During a formatted input operation an illegal character was detected in an input field. The value of the field is set to zero.
6	IGNORE	OUTPUT CONVERSION ERROR During a formatted output operation the value of a particular number could not be output in the specified field length without loss of significant digits. The field is filled with *'s.
10	COUNT:3	FLOATING OVERFLOW During an arithmetic operation a real value has exceeded the largest representable real number. The result of the operation is set to zero.
11	IGNORE	FLOATING UNDERFLOW During an arithmetic operation a real value has become less than the smallest representable real number, and has been replaced with a value of zero.

12	FATAL	FLOATING ZERO DIVIDE During a real mode arithmetic operation an attempt was made to divide by zero. The result of the operation is set to zero.
13	COUNT:3	SQRT OF NEGATIVE NUMBER An attempt was made to take the square root of a negative number. The result returned is zero.
14	FATAL	UNDEFINED EXPONENTIATION OPERATION An attempt was made to perform an illegal exponentiation operation. For example -3.**.5 is illegal because the result would be an imaginary number. The result of the operation is set to zero.
15	FATAL	LOG OF NEGATIVE NUMBER An attempt was made to take the logarithm of a negative number. The result returned is zero.
16	FATAL	WRONG NUMBER OF ARGUMENTS One of the FORTRAN Library functions or System Subroutines which checks for such an occurrence, has been called with an improper number of arguments. See section 8.2 of the PDP-11 FORTRAN Language Reference Manual or Appendix B of this document.

The following error diagnostics should not be changed from the FATAL classification by use of the System Subroutine SETERR:

20	FATAL	INVALID LOGICAL UNIT NUMBER An illegal logical unit number has been specified in an I/O statement. A logical unit number must be an integer between 1 and 99.
21	FATAL	OUT OF AVAILABLE LOGICAL UNITS An attempt was made to have too many devices simultaneously open for I/O. The maximum number of active channels is six by default. This may be changed by use of the /N switch when the main program is compiled.
22	FATAL	INPUT RECORD TOO LONG During an input operation a record was encountered that was longer than the maximum record length. The default maximum record length is 136 bytes. This may be changed by use of the /R switch when the main program is compiled.
23	FATAL	HARDWARE I/O ERROR A hardware error has been detected during an I/O operation.
24	FATAL	ATTEMPT TO READ OR WRITE PAST END OF FILE

25	FATAL	ATTEMPT TO READ AFTER WRITE An attempt was made to read after writing on a magnetic device. A WRITE must be followed by a REWIND or BACKSPACE before a read operation can be performed.
26	FATAL	RECURSIVE I/O NOT ALLOWED An expression in the I/O list of a WRITE statement has caused initiation of another READ or WRITE operation. This can happen if a FUNCTION that performs I/O is referenced in an expression in a WRITE statement I/O list.
27	FATAL	ATTEMPT TO USE DEVICE NOT IN SYSTEM
28	FATAL	OPEN FAILED FOR FILE A file could not be found.
29	FATAL	NO ROOM FOR DEVICE HANDLER There is not enough free core left to accommodate a specific device handler.
30	FATAL	NO ROOM FOR BUFFERS There is not enough free core left to set up required I/O buffers.
31	FATAL	NO AVAILABLE RT-11 CHANNEL More than the maximum number of RT-11 channels, 15, were requested to be opened for I/O.
32	FATAL	FMTD-UNFMTD-RANDOM I/O TO SAME FILE An attempt was made to perform any combination of formatted, unformatted, or random access I/O to the same file.
33	FATAL	ATTEMPT TO READ PAST END OF RECORD An attempt was made to read a larger record than actually existed in a file.
34	FATAL	UNFMTD I/O TO TTY OR LPT An attempt was made to perform an unformatted write operation on the terminal or line printer.
35	FATAL	ATTEMPT TO OUTPUT TO READ ONLY FILE
36	FATAL	BAD FILE SPECIFICATION STRING

37	FATAL	RANDOM ACCESS READ/WRITE BEFORE DEFINE FILE A random access READ or WRITE operation was attempted before a DEFINE FILE was performed.
38	FATAL	RANDOM I/O NOT ALLOWED ON TTY OR LPT Random access I/O was illegally attempted on the terminal or line printer.
39	FATAL	RECORD LARGER THAN RECORD SIZE IN DEFINE FILE A record was encountered that was larger than that specified in the DEFINE FILE statement for a random access file.
40	FATAL	REQUEST FOR A BLOCK LARGER THAN 65535 An attempt was made to reference an absolute disk block address greater than 65535.
41	FATAL	DEFINE FILE ATTEMPTED ON AN OPEN UNIT An open file must be closed by an ENDFILE before another DEFINE FILE can be performed on that unit.
42	FATAL	MEMORY OVERFLOW COMPILING OBJECT TIME FORMAT The OTS has run out of free core while scanning an array format that was generated at run time.
43	FATAL	SYNTAX ERROR IN OBJECT TIME FORMAT A syntax error was encountered while the OTS was scanning an array format that was generated at run time.
44	FATAL	2ND RECORD REQUEST IN ENCODE/DECODE ENCODE and DECODE will operate only on a single record at a time.
45	FATAL	INCOMPATIBLE VARIABLE AND FORMAT TYPES An attempt was made to output a real variable with an integer field descriptor or an integer variable with a real field descriptor.
46	FATAL	INFINITE FORMAT LOOP The format associated with an I/O statement that includes an I/O list has no field descriptors to use in transferring those variables.
60	FATAL	STACK OVERFLOWED The hardware stack has overflowed. Proper Traceback may be impaired.
61	FATAL	ILLEGAL MEMORY REFERENCE This may be any type of BUS error, most probably an illegal memory address reference.

62	FATAL	FORTRAN START FAIL The program has been loaded into core but there was not enough free core remaining for the OTS to initialize work space and buffers.
63	FATAL	ILLEGAL INSTRUCTION The program has attempted to execute an illegal instruction, e.g. floating point arithmetic instruction on a machine with no floating point hardware.

APPENDIX D
COMPILER AND OTS ASSEMBLY AND LINKING INSTRUCTIONS

This appendix provides assembly and linking instructions for the RT-11 FORTRAN Compiler and Object Time System libraries for each of the different hardware configurations. This information applies only to those customers who received the source versions of the Compiler and OTS.

Section D.1 describes the assembly and linking of the RT-11 FORTRAN Compiler. Section D.2 describes the assembly of those modules common to all OTS libraries. Section D.3 describes assembly of the hardware dependent modules for the different hardware support OTS libraries. Section D.4 describes the building of each of the different OTS libraries.

In all examples of assembler command strings, a request is made for the list file to be generated. If no listing is desired, the list file specification may be omitted. In all examples, files are assumed to reside on the default device (DK:). If desired an explicit device specification may be included.

All assemblies require that the system macro file SYSMAC.SML be present on the system device.

D.1 COMPILER ASSEMBLY AND LINKING INSTRUCTIONS

D.1.1 Compiler Assembly

Below is an example of the Compiler assembly procedures. Underlined text is typed by the system; other text is typed by the user.

```
_R MACRO
*FROOT,FROOT=FORTHD,FROOT
ERRORS DETECTED: 0
FREE CORE: 5660. WORDS

*F0,F0=FORTHD,F0
ERRORS DETECTED: 0
FREE CORE: 4541. WORDS

*F1,F1=FORTHD,F1,F1S
ERRORS DETECTED: 0
FREE CORE: 4749. WORDS

*F2,F2=FORTHD,F2
ERRORS DETECTED: 0
FREE CORE: 4822. WORDS

*F3,F3=FORTHD,F3,F3S
ERRORS DETECTED: 0
FREE CORE: 4790. WORDS

*F4,F4=FORTHD,F4,F3S
ERRORS DETECTED: 0
FREE CORE: 4736. WORDS

*F5,F5=FORTHD,F5,F3S
ERRORS DETECTED: 0
FREE CORE: 4750. WORDS

*F6,F6=FORTHD,F6
ERRORS DETECTED: 0
FREE CORE: 4957. WORDS

*F7,F7=F7
ERRORS DETECTED: 0
FREE CORE: 4883. WORDS

*F8,F8=F8
ERRORS DETECTED: 0
FREE CORE: 4753. WORDS

*F9,F9=F9
ERRORS DETECTED: 0
FREE CORE: 4889. WORDS

*F10,F10=F10
ERRORS DETECTED: 0
FREE CORE: 5733. WORDS

*F11,F11=F11
ERRORS DETECTED: 0
FREE CORE: 5544. WORDS

*F12,F12=F12P,FBEGIN,F12
ERRORS DETECTED: 0
FREE CORE: 1240. WORDS

*F13,F13=F13P,FBEGIN,F13
ERRORS DETECTED: 0
FREE CORE: 1455. WORDS
```

```
*F14,F14=F14P,FBEGIN,FDRIVE,FWRT,F14A,F14B
ERRORS DETECTED: 0
FREE CORE: 2018. WORDS

*F15,F15=F15P,FBEGIN,FDRIVE,F15,FWRT,FCODE
ERRORS DETECTED: 0
FREE CORE: 1202. WORDS

*F16,F16=F16P,FBEGIN,FDRIVE,F16,FWRT,FCODE
ERRORS DETECTED: 0
FREE CORE: 1174. WORDS

*F17,F17=F17P,FBEGIN,FDRIVE,F17A,F17B,FCODE
ERRORS DETECTED: 0
FREE CORE: 1010. WORDS

*^C
```

D.1.2 Compiler Linking

Below is an example of the Compiler linking procedures. Underlined text is typed by the system; other text is typed by the user.

```
_R LINKV2
*FORTRA=FROOT/C
*F0/O:1/C
*F1/O:1/C
*F2/O:1/C
*F3/O:1/C
*F4/O:1/C
*F5/O:1/C
*F6/O:1/C
*F7/O:1/C
*F8/O:1/C
*F9/O:1/C
*F10/O:1/C
*F11/O:1/C
*F12/O:1/C
*F13/O:1/C
*F14/O:1/C
*F15/O:1/C
*F16/O:1/C
*F17/O:1
ADDITIVE REF OF WRNBAS
ADDITIVE REF OF WRNBAS
ADDITIVE REF OF WRNBAS
ADDITIVE REF OF WRNBAS
ADDITIVE REF OF WRNBAS

*^C
```

D.2 COMMON OTS MODULES ASSEMBLY

Certain OTS library modules are hardware independent and are therefore common to all OTS libraries. Below is an example of the assembly procedures for the common OTS library modules. Underlined text is typed by the system; other text is typed by the user.

```
.R PIP
*F.MAC=FINIT.PRE,OTSWA.PRE,FBLOCK.PRE,ERRORS.PRE/A
*^C

.R MACRO
*ABS.ALL,ABS=F,ABS
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS

*IABS.ALL,IABS=F,IABS
ERRORS DETECTED: 0
FREE CORE: 2594. WORDS

*DABS.ALL,DABS=F,DABS
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS

*CABS.ALL,CABS=F,CABS
ERRORS DETECTED: 0
FREE CORE: 2738. WORDS

*FLOAT.ALL,FLOAT=F,FLOAT
ERRORS DETECTED: 0
FREE CORE: 2782. WORDS

*DIM.ALL,DIM=F,DIM
ERRORS DETECTED: 0
FREE CORE: 2778. WORDS

*IDIM.ALL,IDIM=F,IDIM
ERRORS DETECTED: 0
FREE CORE: 2586. WORDS

*CEXP.ALL,CEXP=F,CEXP
ERRORS DETECTED: 0
FREE CORE: 2754. WORDS

*CSIN.ALL,CSIN=F,CSIN
ERRORS DETECTED: 0
FREE CORE: 2706. WORDS

*TANH.ALL,TANH=F,TANH
ERRORS DETECTED: 0
FREE CORE: 2698. WORDS

*CONJG.ALL,CONJG=F,CONJG
ERRORS DETECTED: 0
FREE CORE: 2790. WORDS
```

```
*IFIX. ALL, IFIX=F, IFIX
ERRORS DETECTED: 0
FREE CORE: 2782. WORDS

*DBLE. ALL, DBLE=F, DBLE
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS

*REAL. ALL, REAL=F, REAL
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS

*AIMAG. ALL, AIMAG=F, AIMAG
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS

*CMPLX. ALL, CMPLX=F, CMPLX
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS

*INT. ALL, INT=F, INT
ERRORS DETECTED: 0
FREE CORE: 2782. WORDS

*AMOD. ALL, AMOD=F, AMOD
ERRORS DETECTED: 0
FREE CORE: 2758. WORDS

*DMOD. ALL, DMOD=F, DMOD
ERRORS DETECTED: 0
FREE CORE: 2758. WORDS

*MOD. ALL, MOD=F, MOD
ERRORS DETECTED: 0
FREE CORE: 2778. WORDS

*MAX0. ALL, MAX0=F, MAX0
ERRORS DETECTED: 0
FREE CORE: 2782. WORDS

*AMIN0. ALL, AMIN0=F, AMIN0
ERRORS DETECTED: 0
FREE CORE: 2754. WORDS

*MIN0. ALL, MIN0=F, MIN0
ERRORS DETECTED: 0
FREE CORE: 2782. WORDS

*DMIN1. ALL. DMIN1=F, DMIN1
ERRORS DETECTED: 0
FREE CORE: 2755. WORDS

*SIGN. ALL, SIGN=F, SIGN
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS

*ISIGN. ALL. ISIGN=F, ISIGN
ERRORS DETECTED: 0
FREE CORE: 2590. WORDS

*DSIGN. ALL, DSIGN=F, DSIGN
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS
```

```
*CSQRT. ALL, CSQRT=F, CSQRT
ERRORS DETECTED: 0
FREE CORE: 2554. WORDS

*SNGL. ALL, SNGL=F, SNGL
ERRORS DETECTED: 0
FREE CORE: 2590. WORDS

*ENDERR. ALL, ENDERR=F, ENDERR
ERRORS DETECTED: 0
FREE CORE: 2438. WORDS

*CONV5. ALL, CONV5=F, CONV5
ERRORS DETECTED: 0
FREE CORE: 2770. WORDS

*WAIT. ALL, WAIT=F, WAIT
ERRORS DETECTED: 0
FREE CORE: 2743. WORDS

*EOF. ALL, EOF=F, EOF
ERRORS DETECTED: 0
FREE CORE: 2428. WORDS

*FNEG. ALL, FNEG=F, FNEG
ERRORS DETECTED: 0
FREE CORE: 2758. WORDS

*XCI. ALL, XCI=F, XCI
ERRORS DETECTED: 0
FREE CORE: 2538. WORDS

*RETD. ALL, RETD=F, RETD
ERRORS DETECTED: 0
FREE CORE: 2575. WORDS

*IFRW. ALL, IFRW=F, IFRW
ERRORS DETECTED: 0
FREE CORE: 2779. WORDS

*FCHNL ALL, FCHNL=F, FCHNL
ERRORS DETECTED: 0
FREE CORE: 2382. WORDS

*CMPF. ALL, CMPF=F, CMPF
ERRORS DETECTED: 0
FREE CORE: 2690. WORDS

*CMPD. ALL, CMPD=F, CMPD
ERRORS DETECTED: 0
FREE CORE: 2694. WORDS

*INITIO. ALL, INITIO=F, INITIO
ERRORS DETECTED: 0
FREE CORE: 2214. WORDS

*UIO. ALL, UIO=F, UIO
ERRORS DETECTED: 0
FREE CORE: 2130. WORDS

*REWIND. ALL, REWIND=F, REWIND
ERRORS DETECTED: 0
FREE CORE: 2432. WORDS
```

```
*DUMPLA ALL,DUMPLA=F,DUMPLA
ERRORS DETECTED: 0
FREE CORE  2610. WORDS

*CLOSE ALL,CLOSE=F,CLOSE
ERRORS DETECTED: 0
FREE CORE: 2336. WORDS

*VTRAN ALL,VTRAN=F,VTRAN
ERRORS DETECTED: 0
FREE CORE: 2542. WORDS

*GETFIL ALL,GETFIL=F,GETFIL
ERRORS DETECTED: 0
FREE CORE: 2254. WORDS

*EOL ALL,EOL=F,EOL
ERRORS DETECTED: 0
FREE CORE: 2438. WORDS

*CALL ALL,CALL=F,CALL
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS

*ISNLSN ALL,ISNLSN=F,ISNLSN
ERRORS DETECTED: 0
FREE CORE: 2551. WORDS

*IPMOVS ALL,IPMOVS=F,IPMOVS
ERRORS DETECTED: 0
FREE CORE: 2762. WORDS

*IADDS ALL,IADDS=F,IADDS
ERRORS DETECTED: 0
FREE CORE: 2770. WORDS

*IPADDS ALL,IPADDS=F,IPADDS
ERRORS DETECTED: 0
FREE CORE: 2774. WORDS

*ISUBS ALL,ISUBS=F,ISUBS
ERRORS DETECTED: 0
FREE CORE: 2770. WORDS

*IPSUBS ALL,IPSUBS=F,IPSUBS
ERRORS DETECTED: 0
FREE CORE: 2774. WORDS

*INCR ALL,INCR=F,INCR
ERRORS DETECTED: 0
FREE CORE  2774. WORDS

*INEG ALL,INEG=F,INEG
ERRORS DETECTED: 0
FREE CORE: 2790. WORDS

*TESTS ALL,TESTS=F,TESTS
ERRORS DETECTED: 0
FREE CORE: 2758. WORDS

*ICMPS ALL,ICMPS=F,ICMPS
ERRORS DETECTED: 0
FREE CORE: 2770. WORDS
```

```
*IPCMPS. ALL, IPCMPS=F, IPCMPS
ERRORS DETECTED: 0
FREE CORE: 2774. WORDS

*IVEC. ALL, IVEC=F, IVEC
ERRORS DETECTED: 0
FREE CORE: 2782. WORDS

*IVECP. ALL, IVECP=F, IVECP
ERRORS DETECTED: 0
FREE CORE: 2782. WORDS

*IPVEC. ALL, IPVEC=F, IPVEC
ERRORS DETECTED: 0
FREE CORE: 2790. WORDS

*BRAS. ALL, BRAS=F, BRAS
ERRORS DETECTED: 0
FREE CORE: 2759. WORDS

*NXT1. ALL, NXT1=F, NXT1
ERRORS DETECTED: 0
FREE CORE: 2750. WORDS

*NXT2. ALL, NXT2=F, NXT2
ERRORS DETECTED: 0
FREE CORE: 2750. WORDS

*NXT3. ALL, NXT3=F, NXT3
ERRORS DETECTED: 0
FREE CORE: 2750. WORDS

*NXT4. ALL, NXT4=F, NXT4
ERRORS DETECTED: 0
FREE CORE: 2778. WORDS

*AIF. ALL, AIF=F, AIF
ERRORS DETECTED: 0
FREE CORE: 2795. WORDS

*FVEC. ALL, FVEC=F, FVEC
ERRORS DETECTED: 0
FREE CORE: 2770. WORDS

*FVECP. ALL, FVECP=F, FVECP
ERRORS DETECTED: 0
FREE CORE: 2770. WORDS

*FPVEC. ALL, FPVEC=F, FPVEC
ERRORS DETECTED: 0
FREE CORE: 2778. WORDS

*DVEC. ALL, DVEC=F, DVEC
ERRORS DETECTED: 0
FREE CORE: 2770. WORDS

*DVECP. ALL, DVECP=F, DVECP
ERRORS DETECTED: 0
FREE CORE: 2770. WORDS

*DPVEC. ALL, DPVEC=F, DPVEC
ERRORS DETECTED: 0
FREE CORE: 2782. WORDS
```

```
*FMOVR.ALL,FMOVR=F,FMOVR
ERRORS DETECTED: 0
FREE CORE: 2782. WORDS

*FMOV1.ALL,FMOV1=F,FMOV1
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS

*FMOV2.ALL,FMOV2=F,FMOV2
ERRORS DETECTED: 0
FREE CORE: 2798. WORDS

*FMOV3.ALL,FMOV3=F,FMOV3
ERRORS DETECTED: 0
FREE CORE: 2786. WORDS

*FMOV4.ALL,FMOV4=F,FMOV4
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS

*FMOV5.ALL,FMOV5=F,FMOV5
ERRORS DETECTED: 0
FREE CORE: 2802. WORDS

*FMOV6.ALL,FMOV6=F,FMOV6
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS

*FMOV7.ALL,FMOV7=F,FMOV7
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS

*FMOV8.ALL,FMOV8=F,FMOV8
ERRORS DETECTED: 0
FREE CORE: 2786. WORDS

*FMOV9.ALL,FMOV9=F,FMOV9
ERRORS DETECTED: 0
FREE CORE: 2762. WORDS

*LOADS.ALL,LOADS=F,LOADS
ERRORS DETECTED: 0
FREE CORE: 2764. WORDS

*DMOVR.ALL,DMOVR=F,DMOVR
ERRORS DETECTED: 0
FREE CORE: 2790. WORDS

*DMOV1.ALL,DMOV1=F,DMOV1
ERRORS DETECTED: 0
FREE CORE: 2802. WORDS

*DMOV2.ALL,DMOV2=F,DMOV2
ERRORS DETECTED: 0
FREE CORE: 2803. WORDS

*DMOV3.ALL,DMOV3=F,DMOV3
ERRORS DETECTED: 0
FREE CORE: 2766. WORDS

*DMOV4.ALL,DMOV4=F,DMOV4
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS
```

```
*DMOV5. ALL, DMOV5=F, DMOV5
ERRORS DETECTED: 0
FREE CORE: 2778. WORDS

*DMOV6. ALL, DMOV6=F, DMOV6
ERRORS DETECTED: 0
FREE CORE: 2762. WORDS

*DMOV7. ALL, DMOV7=F, DMOV7
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS

*LMOVS. ALL, LMOVS=F, LMOVS
ERRORS DETECTED: 0
FREE CORE: 2742. WORDS

*BITDID. ALL, BITDID=F, BITDID
ERRORS DETECTED: 0
FREE CORE: 2782. WORDS

*LTEST. ALL, LTEST=F, LTEST
ERRORS DETECTED: 0
FREE CORE: 2790. WORDS

*LNOTS. ALL, LNOTS=F, LNOTS
ERRORS DETECTED: 0
FREE CORE: 2786. WORDS

*LCMPS. ALL, LCMPS=F, LCMPS
ERRORS DETECTED: 0
FREE CORE: 2782. WORDS

*LCMPSP. ALL, LCMPSP=F, LCMPSP
ERRORS DETECTED: 0
FREE CORE: 2790. WORDS

*LPCMPS. ALL, LPCMPS=F, LPCMPS
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS

*QVEC. ALL, QVEC=F, QVEC
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS

*QVECP. ALL, QVECP=F, QVECP
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS

*QPVEC. ALL, QPVEC=F, QPVEC
ERRORS DETECTED: 0
FREE CORE: 2798. WORDS

*SUBR. ALL, SUBR=F, SUBR
ERRORS DETECTED: 0
FREE CORE: 2574. WORDS

*LCMPSI. ALL, LCMPSI=F, LCMPSI
ERRORS DETECTED: 0
FREE CORE: 2798. WORDS

*CONV6. ALL, CONV6=F, CONV6
ERRORS DETECTED: 0
FREE CORE: 2775. WORDS
```

```
*XFF. ALL. XFF=F, XFF
ERRORS DETECTED: 0
FREE CORE: 2534. WORDS

*XDD. ALL. XDD=F, XDD
ERRORS DETECTED: 0
FREE CORE: 2514. WORDS

*RETS. ALL, RETS=F, RETS
ERRORS DETECTED: 0
FREE CORE: 2562. WORDS

*ASFRET. ALL, ASFRET=F, ASFRET
ERRORS DETECTED: 0
FREE CORE: 2782. WORDS

*RETDSF. ALL, RETDSF=F, RETDSF
ERRORS DETECTED: 0
FREE CORE: 2583. WORDS

*TCMPLX. ALL, TCMPLX=F, TCMPLX
ERRORS DETECTED: 0
FREE CORE: 2558. WORDS

*TRARY. ALL, TRARY=F, TRARY
ERRORS DETECTED: 0
FREE CORE: 2538. WORDS

*FUD. ALL, FUD=F, FUD
ERRORS DETECTED: 0
FREE CORE: 2602. WORDS

*IMOVS. ALL, IMOVS=F, IMOVS
ERRORS DETECTED: 0
FREE CORE: 2730. WORDS

*IMOVR. ALL, IMOVR=F, IMOVR
ERRORS DETECTED: 0
FREE CORE: 2786. WORDS

*LMOVR. ALL, LMOVR=F, LMOVR
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS

*GOTO. ALL, GOTO=F, GOTO
ERRORS DETECTED: 0
FREE CORE: 2590. WORDS

*PUTREC. ALL, PUTREC=F, PUTREC
ERRORS DETECTED: 0
FREE CORE: 2158. WORDS

*RAN. ALL, RAN=F, RAN
ERRORS DETECTED: 0
FREE CORE: 2574. WORDS

*RANDU. ALL, RANDU=F, RANDU
ERRORS DETECTED: 0
FREE CORE: 2574. WORDS

*FIND. ALL, FIND=F, FIND
ERRORS DETECTED: 0
FREE CORE: 2230. WORDS
```

```
*OPEN. ALL, OPEN=F, OPEN
ERRORS DETECTED: 0
FREE CORE: 2121. WORDS

*VCTRAN. ALL, VCTRAN=F, VCTRAN
ERRORS DETECTED: 0
FREE CORE: 2562. WORDS

*CONVL. ALL, CONVL=F, CONVL
ERRORS DETECTED: 0
FREE CORE: 2770. WORDS

*IDATE. ALL, IDATE=F, IDATE
ERRORS DETECTED: 0
FREE CORE: 2539. WORDS

*SETERR. ALL, SETERR=F, SETERR
ERRORS DETECTED: 0
FREE CORE: 2386. WORDS

*ASSIGN. ALL, ASSIGN=F, ASSIGN
ERRORS DETECTED: 0
FREE CORE: 2100. WORDS

*ERRS. ALL, ERRS=F, ERRS
ERRORS DETECTED: 0
FREE CORE: 2235. WORDS

*RIO. ALL, RIO=F, RIO
ERRORS DETECTED: 0
FREE CORE: 2146. WORDS

*AMAX1. ALL, AMAX1=F, AMAX1
ERRORS DETECTED: 0
FREE CORE: 2726. WORDS

*4K. SIZ, 4K=F, 4K
ERRORS DETECTED: 0
FREE CORE: 2803. WORDS

*8K. SIZ, 8K=F, 8K
ERRORS DETECTED: 0
FREE CORE: 2803. WORDS

*12K. SIZ, 12K=F, 12K
ERRORS DETECTED: 0
FREE CORE: 2803. WORDS

*16K. SIZ, 16K=F, 16K
ERRORS DETECTED: 0
FREE CORE: 2803. WORDS

*20K. SIZ, 20K=F, 20K
ERRORS DETECTED: 0
FREE CORE: 2803. WORDS

*24K. SIZ, 24K=F, 24K
ERRORS DETECTED: 0
FREE CORE: 2803. WORDS

*28K. SIZ, 28K=F, 28K
ERRORS DETECTED: 0
FREE CORE: 2803. WORDS
```

```
*SIMRT.UNI,SIMRT=F,SIMRT
ERRORS DETECTED: 0
FREE CORE: 2216. WORDS

*POPR.ALL,POPR=F,POPR
ERRORS DETECTED: 0
FREE CORE: 2794. WORDS

*CNEG.ALL,CNEG=F,CNEG
ERRORS DETECTED: 0
FREE CORE: 2774. WORDS

*COPY.ALL,COPY=F,COPY
ERRORS DETECTED: 0
FREE CORE: 2786. WORDS

*GETREC ALL,GETREC=F,GETREC
ERRORS DETECTED: 0
FREE CORE: 2115. WORDS

*LVEC.ALL,LVEC=F,LVEC
ERRORS DETECTED: 0
FREE CORE: 2782. WORDS

*LVECP.ALL,LVECP=F,LVECP
ERRORS DETECTED: 0
FREE CORE: 2782. WORDS

*LPVEC.ALL,LPVEC=F,LPVEC
ERRORS DETECTED: 0
FREE CORE: 2790. WORDS

*FCALL.ALL,FCALL=F,FCALL
ERRORS DETECTED: 0
FREE CORE: 2798. WORDS

*PAUSE.ALL,PAUSE=F,PAUSE
ERRORS DETECTED: 0
FREE CORE: 2679. WORDS

*BACKSP.ALL,BACKSP=F,BACKSP
ERRORS DETECTED: 0
FREE CORE: 2388. WORDS

*OBJFMT ALL,OBJFMT=F,OBJFMT
ERRORS DETECTED: 0
FREE CORE: 2003. WORDS

*RWBLK.ALL,RWBLK=F,RWBLK
ERRORS DETECTED: 0
FREE CORE: 2198. WORDS

*CLOG.ALL,CLOG=F,CLOG
ERRORS DETECTED: 0
FREE CORE: 2778. WORDS

*STOP.ALL,STOP=F,STOP
ERRORS DETECTED: 0
FREE CORE: 2487. WORDS
```

```
*ENCODE.ALL,ENCODE=F,ENCODE
ERRORS DETECTED: 0
FREE CORE: 2250. WORDS

*DECODE.ALL,DECODE=F,DECODE
ERRORS DETECTED: 0
FREE CORE: 2390. WORDS

*OBJDEC.ALL,OBJDEC=F,OBJDEC
ERRORS DETECTED: 0
FREE CORE: 2390. WORDS

*OBJENC.ALL,OBJENC=F,OBJENC
ERRORS DETECTED: 0
FREE CORE: 2250. WORDS

*FUDGE.ALL,FUDGE=F,FUDGE
ERRORS DETECTED: 0
FREE CORE: 2799. WORDS

*USEREX.ALL,USEREX=F,USEREX
ERRORS DETECTED: 0
FREE CORE: 2398. WORDS

*FIO.ALL,FIO=F,FIO
ERRORS DETECTED: 0
FREE CORE: 2026. WORDS

*SHORT.UNI,SHORT=F,SHORT
ERRORS DETECTED: 0
FREE CORE: 2411. WORDS

*TESTC.ALL,TESTC=F,TESTC
ERRORS DETECTED: 0
FREE CORE: 2782. WORDS

*^C

.R FORTRA
*DATE.ALL,DATE=DATE
*^C
```

D.3 HARDWARE DEPENDENT OTS MODULES ASSEMBLY

Certain OTS library modules are hardware dependent and therefore need to be selectively assembled. Use the following table to determine which section to refer to for final OTS module assembly.

Section	Hardware Configuration
D.3.1	bare machine
D.3.2	EIS
D.3.3	FIS
D.3.4	EAE
D.3.5	FPU

D.3.1 Bare Machine OTS Assembly

Below is an example of the assembly procedures for the hardware dependent modules on a machine with none of the optional arithmetic hardware extensions. Underlined text is typed by the system; other text is typed by the user.

```
_R PIP
*F.MAC=FINIT.PRE,OTSWA.PRE,FBLOCK.PRE,ERRORS.PRE/A
*^C

_R MACRO
*ATAN.BAR,ATAN=F,ATAN
ERRORS DETECTED: 0
FREE CORE: 2673. WORDS

*XII BAR,XII=F,XII
ERRORS DETECTED: 0
FREE CORE: 2538. WORDS

*CONV.BAR,CONV=F,CONV
ERRORS DETECTED: 0
FREE CORE: 2338. WORDS

*CONV2.BAR,CONV2=F,CONV2
ERRORS DETECTED: 0
FREE CORE: 2542. WORDS

*CMUL.BAR,CMUL=F,CMUL
ERRORS DETECTED: 0
FREE CORE: 2725. WORDS

*CDIV.BAR,CDIV=F,CDIV
ERRORS DETECTED: 0
FREE CORE: 2490. WORDS

*DMUL.BAR,DMUL=F,DMUL
ERRORS DETECTED: 0
FREE CORE: 2474. WORDS

*DADD.BAR,DADD=F,DADD
ERRORS DETECTED: 0
FREE CORE: 2386. WORDS

*DDIV.BAR,DDIV=F,DDIV
ERRORS DETECTED: 0
FREE CORE: 2482. WORDS

*XDI.BAR,XDI=F,XDI
ERRORS DETECTED: 0
FREE CORE 2538. WORDS

*CONV1.BAR,CONV1=F,CONV1
ERRORS DETECTED: 0
FREE CORE: 2782. WORDS
```

```
*CONV3. BAR, CONV3=F, CONV3
ERRORS DETECTED:  0
FREE CORE:  2758.  WORDS

*CADD. BAR, CADD=F, CADD
ERRORS DETECTED:  0
FREE CORE:  2734.  WORDS

*ALOG. BAR, ALOG=F, ALOG
ERRORS DETECTED:  0
FREE CORE:  2498.  WORDS

*DEXP. BAR, DEXP=F, DEXP
ERRORS DETECTED:  0
FREE CORE:  2462.  WORDS

*EXP. BAR, EXP=F, EXP
ERRORS DETECTED:  0
FREE CORE:  2518.  WORDS

*DSQRT. BAR, DSQRT=F, DSQRT
ERRORS DETECTED:  0
FREE CORE:  2562.  WORDS

*SIN. BAR, SIN=F, SIN
ERRORS DETECTED:  0
FREE CORE:  2709.  WORDS

*DSIN. BAR, DSIN=F, DSIN
ERRORS DETECTED:  0
FREE CORE:  2669.  WORDS

*DATAN. BAR, DATAN=F, DATAN
ERRORS DETECTED:  0
FREE CORE:  2669.  WORDS

*AINT. BAR, AINT=F, AINT
ERRORS DETECTED:  0
FREE CORE:  2764.  WORDS

*DINT. BAR, DINT=F, DINT
ERRORS DETECTED:  0
FREE CORE:  2764.  WORDS

*ADDA. BAR, ADDA=F, ADDA
ERRORS DETECTED:  0
FREE CORE:  2746.  WORDS

*SQRT. BAR, SQRT=F, SQRT
ERRORS DETECTED:  0
FREE CORE:  2558.  WORDS

*OTI. BAR, OTI=F, OTI
ERRORS DETECTED:  0
FREE CORE:  1867.  WORDS

*FMUL. BAR, FMUL=F, FMUL
ERRORS DETECTED:  0
FREE CORE:  2530.  WORDS

*FDIV. BAR, FDIV=F, FDIV
ERRORS DETECTED:  0
FREE CORE:  2502.  WORDS
```

```
*ADDM.BAR,ADDM=F,ADDM
ERRORS DETECTED: 0
FREE CORE: 2746. WORDS

*ADDF.BAR,ADDF=F,ADDF
ERRORS DETECTED: 0
FREE CORE: 2746. WORDS

*DLOG.BAR,DLOG=F,DLOG
ERRORS DETECTED: 0
FREE CORE: 2490. WORDS

*FADD.BAR,FADD=F,FADD
ERRORS DETECTED: 0
FREE CORE: 2462. WORDS

*XFI.BAR,XFI=F,XFI
ERRORS DETECTED: 0
FREE CORE: 2538. WORDS

*CONV4.BAR,CONV4=F,CONV4
ERRORS DETECTED: 0
FREE CORE: 2590. WORDS

*IMUL.BAR,IMUL=F,IMUL
ERRORS DETECTED: 0
FREE CORE: 2554. WORDS

*IDIV.BAR,IDIV=F,IDIV
ERRORS DETECTED: 0
FREE CORE: 2546. WORDS

*^C
```

D.3.2 EIS OTS Assembly

Below is an example of the assembly procedures for the hardware dependent modules on a machine with the EIS hardware option. Underlined text is typed by the system; other text is typed by the user.

```
_R PIP
*F.MAC=FINIT.EIS,OTSWA.PRE,FBLOCK.PRE,ERRORS.PRE/A
*^C

_R MACRO
*ATAN.EIS,ATAN=F,ATAN
ERRORS DETECTED: 0
FREE CORE: 2665. WORDS

*XII.EIS,XII=F,XII
ERRORS DETECTED: 0
FREE CORE: 2546. WORDS

*CONV.EIS.CONV=F,CONV
ERRORS DETECTED: 0
FREE CORE: 2338. WORDS
```

```
*CONV2.EIS,CONV2=F,CONV2
ERRORS DETECTED: 0
FREE CORE: 2526. WORDS

*CMUL.EIS,CMUL=F,CMUL
ERRORS DETECTED: 0
FREE CORE: 2717. WORDS

*CDIV.EIS,CDIV=F,CDIV
ERRORS DETECTED: 0
FREE CORE: 2482. WORDS

*DMUL.EIS,DMUL=F,DMUL
ERRORS DETECTED: 0
FREE CORE: 2482. WORDS

*DADD.EIS,DADD=F,DADD
ERRORS DETECTED: 0
FREE CORE: 2382. WORDS

*DDIV.EIS,DDIV=F,DDIV
ERRORS DETECTED: 0
FREE CORE: 2474. WORDS

*XDI.EIS,XDI=F,XDI
ERRORS DETECTED: 0
FREE CORE: 2530. WORDS

*CONV1.EIS,CONV1=F,CONV1
ERRORS DETECTED: 0
FREE CORE: 2774. WORDS

*CONV3.EIS,CONV3=F,CONV3
ERRORS DETECTED: 0
FREE CORE: 2750. WORDS

*CADD.EIS,CADD=F,CADD
ERRORS DETECTED: 0
FREE CORE: 2726. WORDS

*ALOG.EIS,ALOG=F,ALOG
ERRORS DETECTED: 0
FREE CORE: 2490. WORDS

*DEXP.EIS,DEXP=F,DEXP
ERRORS DETECTED: 0
FREE CORE: 2454. WORDS

*EXP.EIS,EXP=F,EXP
ERRORS DETECTED: 0
FREE CORE: 2510. WORDS

*DSQRT.EIS,DSQRT=F,DSQRT
ERRORS DETECTED: 0
FREE CORE: 2554. WORDS

*SIN.EIS,SIN=F,SIN
ERRORS DETECTED: 0
FREE CORE: 2701. WORDS

*DSIN.EIS,DSIN=F,DSIN
ERRORS DETECTED: 0
FREE CORE: 2661. WORDS
```

```
*DATAN EIS,DATAN=F,DATAN
ERRORS DETECTED: 0
FREE CORE: 2661. WORDS

*AINT EIS,AINT=F,AINT
ERRORS DETECTED: 0
FREE CORE: 2764. WORDS

*DINT EIS,DINT=F,DINT
ERRORS DETECTED: 0
FREE CORE: 2764. WORDS

*ADDA EIS,ADDA=F,ADDA
ERRORS DETECTED: 0
FREE CORE: 2738. WORDS

*SQRT EIS,SQRT=F,SQRT
ERRORS DETECTED: 0
FREE CORE: 2550. WORDS

*OTI EIS,OTI=F,OTI
ERRORS DETECTED: 0
FREE CORE: 1859. WORDS

*FMUL EIS,FMUL=F,FMUL
ERRORS DETECTED: 0
FREE CORE: 2490. WORDS

*FDIV EIS,FDIV=F,FDIV
ERRORS DETECTED: 0
FREE CORE: 2486. WORDS

*ADDM EIS,ADDM=F,ADDM
ERRORS DETECTED: 0
FREE CORE: 2738. WORDS

*ADDP EIS,ADDP=F,ADDP
ERRORS DETECTED: 0
FREE CORE: 2738. WORDS

*DLOG EIS,DLOG=F,DLOG
ERRORS DETECTED: 0
FREE CORE: 2482. WORDS

*FADD EIS,FADD=F,FADD
ERRORS DETECTED: 0
FREE CORE: 2454. WORDS

*XFI EIS,XFI=F,XFI
ERRORS DETECTED: 0
FREE CORE: 2530. WORDS

*CONV4 EIS,CONV4=F,CONV4
ERRORS DETECTED: 0
FREE CORE: 2582. WORDS

*IMUL EIS,IMUL=F,IMUL
ERRORS DETECTED: 0
FREE CORE: 2570. WORDS

*IDIV EIS,IDIV=F,IDIV
ERRORS DETECTED: 0
FREE CORE: 2570. WORDS

*^C
```

```
*ADDP.FIS,ADDP=F,ADDP
ERRORS DETECTED: 0
FREE CORE: 2746. WORDS

*DLOG.FIS,DLOG=F,DLOG
ERRORS DETECTED: 0
FREE CORE: 2486. WORDS

*FADD.FIS,FADD=F,FADD
ERRORS DETECTED: 0
FREE CORE: 2554. WORDS

*XFI.FIS,XFI=F,XFI
ERRORS DETECTED: 0
FREE CORE: 2534. WORDS

*CONV4.FIS,CONV4=F,CONV4
ERRORS DETECTED: 0
FREE CORE: 2586. WORDS

*IMUL.FIS,IMUL=F,IMUL
ERRORS DETECTED: 0
FREE CORE: 2574. WORDS

*IDIV.FIS,IDIV=F,IDIV
ERRORS DETECTED: 0
FREE CORE: 2574. WORDS

*^C
```

D.3.4 EAE OTS Assembly

Below is an example of the assembly procedures for the hardware dependent modules on a machine with the EAE hardware option. Underlined text is typed by the system; other text is typed by the user.

```
.R PIP
*F.MAC=FINIT.EAE,OTSWA.PRE,FBLOCK.PRE,ERRORS.PRE/A
*^C

.R MACRO
*ATAN.EAE,ATAN=F,ATAN
ERRORS DETECTED: 0
FREE CORE: 2669. WORDS

*XII.EAE,XII=F,XII
ERRORS DETECTED: 0
FREE CORE: 2534. WORDS

*CONV.EAE,CONV=F,CONV
ERRORS DETECTED: 0
FREE CORE: 2334. WORDS

*CONV2.EAE,CONV2=F,CONV2
ERRORS DETECTED: 0
FREE CORE: 2522. WORDS
```

```
*CMUL EAE,CMUL=F,CMUL
ERRORS DETECTED: 0
FREE CORE: 2721. WORDS

*CDIV EAE,CDIV=F,CDIV
ERRORS DETECTED: 0
FREE CORE: 2486. WORDS

*DMUL EAE,DMUL=F,DMUL
ERRORS DETECTED: 0
FREE CORE: 2486. WORDS

*DADD EAE,DADD=F,DADD
ERRORS DETECTED: 0
FREE CORE: 2386. WORDS

*DDIV EAE,DDIV=F,DDIV
ERRORS DETECTED: 0
FREE CORE: 2478. WORDS

*XDI EAE,XDI=F,XDI
ERRORS DETECTED: 0
FREE CORE: 2534. WORDS

*CONV1 EAE,CONV1=F,CONV1
ERRORS DETECTED: 0
FREE CORE: 2778. WORDS

*CONV3 EAE,CONV3=F,CONV3
ERRORS DETECTED: 0
FREE CORE: 2746. WORDS

*CADD EAE,CADD=F,CADD
ERRORS DETECTED: 0
FREE CORE: 2730. WORDS

*ALOG EAE,ALOG=F,ALOG
ERRORS DETECTED: 0
FREE CORE: 2494. WORDS

*DEXP EAE,DEXP=F,DEXP
ERRORS DETECTED: 0
FREE CORE: 2458. WORDS

*EXP EAE,EXP=F,EXP
ERRORS DETECTED: 0
FREE CORE: 2514. WORDS

*DSQRT EAE,DSQRT=F,DSQRT
ERRORS DETECTED: 0
FREE CORE: 2558. WORDS

*SIN EAE,SIN=F,SIN
ERRORS DETECTED: 0
FREE CORE: 2705. WORDS

*DSIN EAE,DSIN=F,DSIN
ERRORS DETECTED: 0
FREE CORE: 2665. WORDS

*DATAN EAE,DATAN=F,DATAN
ERRORS DETECTED: 0
FREE CORE: 2665. WORDS
```

```
*AINT.EAE,AINT=F,AINT
ERRORS DETECTED: 0
FREE CORE: 2760. WORDS

*DINT.EAE,DINT=F,DINT
ERRORS DETECTED: 0
FREE CORE: 2760. WORDS

*ADDA.EAE,ADDA=F,ADDA
ERRORS DETECTED: 0
FREE CORE: 2742. WORDS

*SQRT.EAE,SQRT=F,SQRT
ERRORS DETECTED: 0
FREE CORE: 2554. WORDS

*OTI.EAE,OTI=F,OTI
ERRORS DETECTED: 0
FREE CORE: 1863. WORDS

*FMUL.EAE,FMUL=F,FMUL
ERRORS DETECTED: 0
FREE CORE: 2482. WORDS

*FDIV.EAE,FDIV=F,FDIV
ERRORS DETECTED: 0
FREE CORE: 2494. WORDS

*ADDM.EAE,ADDM=F,ADDM
ERRORS DETECTED: 0
FREE CORE: 2742. WORDS

*ADDP.EAE,ADDP=F,ADDP
ERRORS DETECTED: 0
FREE CORE: 2742. WORDS

*DLOG.EAE,DLOG=F,DLOG
ERRORS DETECTED: 0
FREE CORE: 2486. WORDS

*FADD.EAE,FADD=F,FADD
ERRORS DETECTED: 0
FREE CORE: 2458. WORDS

*XFI.EAE,XFI=F,XFI
ERRORS DETECTED: 0
FREE CORE: 2534. WORDS

*CONV4.EAE,CONV4=F,CONV4
ERRORS DETECTED: 0
FREE CORE: 2586. WORDS

*IMUL.EAE,IMUL=F,IMUL
ERRORS DETECTED: 0
FREE CORE: 2562. WORDS

*IDIV.EAE,IDIV=F,IDIV
ERRORS DETECTED: 0
FREE CORE: 2570. WORDS

*^C
```

D.3.5 FPU OTS Assembly

Below is an example of the assembly procedures for the hardware dependent modules on a machine with the FPU hardware option. Underlined text is typed by the system; other text is typed by the user.

```
.R PIP
*F.MAC=FINIT.FPU,OTSWA.PRE,FBLOCK.PRE,ERRORS.PRE/A
*^C

.R MACRO
*ATAN.FPU,ATAN=F,ATAN
ERRORS DETECTED: 0
FREE CORE: 2669. WORDS

*XII FPU,XII=F,XII
ERRORS DETECTED: 0
FREE CORE: 2534. WORDS

*CONV.FPU,CONV=F,CONV
ERRORS DETECTED: 0
FREE CORE: 2342. WORDS

*CONV2.FPU,CONV2=F,CONV2
ERRORS DETECTED: 0
FREE CORE: 2538. WORDS

*CMUL.FPU,CMUL=F,CMUL
ERRORS DETECTED: 0
FREE CORE: 2721. WORDS

*CDIV.FPU,CDIV=F,CDIV
ERRORS DETECTED: 0
FREE CORE: 2486. WORDS

*DMUL.FPU,DMUL=F,DMUL
ERRORS DETECTED: 0
FREE CORE: 2470. WORDS

*DADD.FPU,DADD=F,DADD
ERRORS DETECTED: 0
FREE CORE: 2382. WORDS

*DDIV.FPU,DDIV=F,DDIV
ERRORS DETECTED: 0
FREE CORE: 2478. WORDS

*XDI.FPU,XDI=F,XDI
ERRORS DETECTED: 0
FREE CORE: 2534. WORDS

*CONV1.FPU,CONV1=F,CONV1
ERRORS DETECTED: 0
FREE CORE: 2778. WORDS

*CONV3.FPU,CONV3=F,CONV3
ERRORS DETECTED: 0
FREE CORE: 2754. WORDS
```

```
*CADD.FPU,CADD=F,CADD
ERRORS DETECTED: 0
FREE CORE: 2730. WORDS

*ALOG.FPU,ALOG=F,ALOG
ERRORS DETECTED: 0
FREE CORE: 2494. WORDS

*DEXP.FPU,DEXP=F,DEXP
ERRORS DETECTED: 0
FREE CORE: 2458. WORDS

*EXP.FPU,EXP=F,EXP
ERRORS DETECTED: 0
FREE CORE: 2514. WORDS

*DSQRT.FPU,DSQRT=F,DSQRT
ERRORS DETECTED: 0
FREE CORE: 2558. WORDS

*SIN.FPU,SIN=F,SIN
ERRORS DETECTED: 0
FREE CORE: 2705. WORDS

*DSIN.FPU,DSIN=F,DSIN
ERRORS DETECTED: 0
FREE CORE: 2665. WORDS

*DATAN.FPU,DATAN=F,DATAN
ERRORS DETECTED: 0
FREE CORE: 2665. WORDS

*AINT.FPU,AINT=F,AINT
ERRORS DETECTED: 0
FREE CORE: 2760. WORDS

*DINT.FPU,DINT=F,DINT
ERRORS DETECTED: 0
FREE CORE: 2760. WORDS

*ADDA.FPU,ADDA=F,ADDA
ERRORS DETECTED: 0
FREE CORE: 2738. WORDS

*SQRT.FPU,SQRT=F,SQRT
ERRORS DETECTED: 0
FREE CORE: 2558. WORDS

*OTI.FPU,OTI=F,OTI
ERRORS DETECTED: 0
FREE CORE: 1847. WORDS

*FMUL.FPU,FMUL=F,FMUL
ERRORS DETECTED: 0
FREE CORE: 2578. WORDS

*FDIV.FPU,FDIV=F,FDIV
ERRORS DETECTED: 0
FREE CORE: 2578. WORDS

*ADDM.FPU,ADDM=F,ADDM
ERRORS DETECTED: 0
FREE CORE: 2746. WORDS
```

```
*ADDF FPU,ADDF=F,ADDF
ERRORS DETECTED: 0
FREE CORE: 2746  WORDS

*DLOG FPU,DLOG=F,DLOG
ERRORS DETECTED: 0
FREE CORE: 2486  WORDS

*FADD FPU,FADD=F,FADD
ERRORS DETECTED: 0
FREE CORE: 2554  WORDS

*XFI FPU,XFI=F,XFI
ERRORS DETECTED: 0
FREE CORE: 2534  WORDS

*CONV4 FPU,CONV4=F,CONV4
ERRORS DETECTED: 0
FREE CORE: 2586  WORDS

*IMUL FPU,IMUL=F,IMUL
ERRORS DETECTED: 0
FREE CORE: 2574  WORDS

*IDIV FPU,IDIV=F,IDIV
ERRORS DETECTED: 0
FREE CORE: 2574  WORDS

*^C
```

D.4 LIBRARY BUILDING PROCEDURES

For final OTS library preparation refer to the appropriate section as listed below.

Section	Hardware Configuration
D.4.1	bare machine
D.4.2	EIS
D.4.3	FIS
D.4.4	EAE
D.4.5	FPU

D.4.1 Building the Bare Machine OTS

Below is an example of the building procedures for the OTS library on a machine with none of the optional arithmetic hardware extensions. Underlined text is typed by the system; other text is typed by the user.

```
.R PIP
*OTS.OBJ=*.ALL,*.BAR/B
*UNI.OBJ=*.UNI,*.SIZ/B
*^C

.R LIBR
*FORLIB=UNI,OTS/G


ENTRY POINT:
$ERRS
$ERRTB


*^C
```

D.4.2 Building the EIS OTS

Below is an example of the building procedures for the OTS library on a machine with the EIS hardware option. Underlined text is typed by the system; other text is typed by the user.

```
.R PIP
*EIS.OBJ=*.ALL,*.EIS/B
*UNI.OBJ=*.UNI,*.SIZ/B
*^C

.R LIBR
*FORLIB=UNI,EIS/G


ENTRY POINT:
$ERRS
$ERRTB


*^C
```

D.4.3 Building the FIS OTS

Below is an example of the building procedures for the OTS library on a machine with the FIS hardware option. Underlined text is typed by the system; other text is typed by the user.

```
.R PIP
*FIS.OBJ=*.ALL,*.FIS/B
*UNI.OBJ=*.UNI,*.SIZ/B
*^C

.R LIBR
*FORLIB=UNI,FIS/G


ENTRY POINT:
$ERRS
$ERRTB


*^C
```

D.4.4 Building the EAE OTS

Below is an example of the building procedures for the OTS library on a machine with the EAE hardware option. Underlined text is typed by the system; other text is typed by the user.

```
.R PIP
*EAE.OBJ=*.ALL,*.EAE/B
*UNI.OBJ=*.UNI,*.SIZ/B
*^C

.R LIBR
*FORLIB=UNI,EAE/G


ENTRY POINT
$ERRS
$ERRTB


*^C
```

D.4.5 Building the FPU OTS

Below is an example of the building procedures for the OTS library on a machine with the FPU hardware option. Underlined text is typed by the system; other text is typed by the user.

```
.R PIP
*FPU.OBJ=*.ALL,*.FPU/B
*UNI.OBJ=*.UNI,*.SIZ/B
*^C

.R LIBR
*FORLIB=UNI,FPU/G


ENTRY POINT:
$ERRS
$ERRTB


*^C
```

INDEX

HOW TO OBTAIN SOFTWARE INFORMATION

SOFTWARE NEWSLETTERS, MAILING LIST

The Software Communications Group, located at corporate headquarters in Maynard, publishes newsletters and Software Performance Summaries (SPS) for the various Digital products. Newsletters are published monthly, and contain announcements of new and revised software, programming notes, software problems and solutions, and documentation corrections. Software Performance Summaries are a collection of existing problems and solutions for a given software system, and are published periodically. For information on the distribution of these documents and how to get on the software newsletter mailing list, write to:

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

SOFTWARE PROBLEMS

Questions or problems relating to Digital's software should be reported to a Software Support Specialist. A specialist is located in each Digital Sales Office in the United States. In Europe, software problem reporting centers are in the following cities.

Reading, England	Milan, Italy
Paris, France	Solna, Sweden
The Hague, Holland	Geneva, Switzerland
Tel Aviv, Israel	Munich, West Germany

Software Problem Report (SPR) forms are available from the specialists or from the Software Distribution Centers cited below.

PROGRAMS AND MANUALS

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center.

Digital Equipment Corporation
Software Distribution Center
146 Main Street
Maynard, Massachusetts 01754

Digital Equipment Corporation
Software Distribution Center
1400 Terra Bella
Mountain View, California 94043

Outside of the United States, orders should be directed to the nearest Digital Field Sales Office or representative.

USERS SOCIETY

DECUS, Digital Equipment Computer Users Society, maintains a user exchange center for user-written programs and technical application information. A catalog of existing programs is available. The society publishes a periodical, DECUSCOPE, and holds technical seminars in the United States, Canada, Europe, and Australia. For information on the society and membership application forms, write to:

DECUS
Digital Equipment Corporation
146 Main Street
Maynard, Massachusetts 01754

DECUS
Digital Equipment, S.A.
81 Route de l'Aire
1211 Geneva 26
Switzerland

RT-11 FORTRAN Compiler
and Object Time System
User's Manual
DEC-11-LRFPA-A-D

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form (see the HOW TO OBTAIN SOFTWARE INFORMATION page).

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

If you do not require a written reply, please check here. ☐

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

DIGITAL EQUIPMENT CORPORATION
MAYNARD, MASSACHUSETTS 01754