Compaq Fortran 77

Release Notes for OpenVMS VAX Systems

November 1999

This document contains information about new and changed features in this version of Compaq Fortran 77 for OpenVMS VAX Systems (formerly DIGITAL Fortran 77).

Software Version: Compaq Fortran 77 6.6-201

Compaq Computer Corporation
Houston, Texas

November 1999

Digital Equipment Corporation makes no representations that the use of its products in the manner described in this publication will not infringe on existing or future patent rights, nor do the descriptions contained in this publication imply the granting of licenses to make, use, or sell equipment or software in accordance with the description.

Possession, use, or copying of the software described in this publication is authorized only pursuant to a valid written license from Digital or an authorized sublicensor.

© Digital Equipment Corporation 1999. All Rights Reserved.

The following are trademarks of Digital Equipment Corporation: Alpha AXP, AXP, Bookreader, DEC, DECnet, DECwindows, DEC Fortran, DIGITAL, OpenVMS, VAX, VAX FORTRAN, VMS, and the DIGITAL logo.

PostScript is a trademark of Adobe Systems, Inc. IBM, System\370 and RS-6000 are trademarks of International Business Machines, Inc. CRAY is a trademark of Cray Research, Inc. Sun is a trademark of Sun Microsystems, Inc.

This document was prepared using VAX DOCUMENT Version 2.1.

Contents

1 Compaq Fortran 77 Version 6.6-201 Release Notes

1.1 Overview of Compaq Fortran 77 Version 6.6-201 1–1
1.2 New and Changed Features in V6.6-201 1–2
1.2.1 Detection of COMMON Name Conflict 1–2
1.2.2 /WARNINGS=[NO]UNREACHABLE Implemented 1–2
1.3 Compatibility 1–2
1.3.1 Coexistence With Earlier Versions 1–2
1.3.2 Source Compatibility 1–3
1.3.3 New Math Library Considerations 1–4
1.3.4 Fortran Run-Time Library Considerations 1–4
1.3.5 Object Compatibility 1–5
1.4 Known Problems and Issues 1–5
1.4.1 Required Post-Installation Tasks on VMScluster 1–5
1.4.2 Routines Called With Omitted Arguments 1–5
1.4.3 Compiler Bugchecks with /VECTOR 1–6
1.4.4 Linker OUTSIMG Error When COLLECT Linker Option Used 1–6
1.4.5 Recursive Character Functions Not Supported 1–6
1.4.6 Restrictions on Use of BLAS Routines 1–6

2 Release Notes for Previous Versions of Compaq Fortran 77

2.1 New and Changed Features in V6.5 2–1
2.1.1 New DATE_AND_TIME Intrinsc 2–1
2.2 New and Changed Features in V6.4 2–2
2.2.1 STRUCTURE and COMMON Alignment 2–2
2.2.1.1 /ALIGN Compile Command Qualifier 2–2
2.2.1.2 /WARNINGS=ALIGNMENT Supports STRUCTUREs 2–3
2.2.1.3 CDEC$ OPTIONS Directive 2–3
2.2.1.4 CDEC$ PSECT MULTILANGUAGE Keyword 2–3
2.2.2 /ASSUME=BYTERECL Command and OPTIONS Qualifier 2–4
2.2.3 CDEC$ PSECT ALIGN= Keywords Supported 2–4
2.2.4 DPROD Intrinsic Now Generic 2–4
2.2.5 Use of POINTER Variables in DEBUG 2–4
2.3 New and Changed Features in V6.3 2–5
2.3.1 /ASSUME=SOURCE_INCLUDE Command Line Keyword 2–5
2.3.2 /DEBUG=PARAMETERS=NONE Command Line Keyword 2–6
2.3.3 /WARNINGS=[NO]INFORMATIONAL Command Line Keyword 2–6
2.3.4 Expanded syntax for Directives 2–6
2.3.5 Sign Extension of Untyped Bit Constants 2–7
2.3.6 AND, OR, XOR, LSHIFT and RSHIFT Intrinsic Functions Supported 2–7
2.3.7 FORSYSDEF.TLB Modules for Callable Utility Routines 2–7
2.4 Overview of Compaq Fortran 77 Version V6.2 2–8
2.5 New and Changed Features in V6.2 2–8

2.5.1 Short Source Lines and /PAD_SOURCE Qualifier 2–8
2.5.2 Pointee Can Be Called Routine 2–9
2.5.3 CDEC$ OPTIONS and END OPTIONS Directives Ignored 2–9
2.5.4 New CDEC$ ALIAS Directive 2–10
2.5.5 # Accepted as Comment Introducer 2–10
2.5.6 Reduction in Compiler Virtual Memory Usage 2–11
2.5.7 Negative Constant Values in FORSYSDEF.TLB 2–11
2.5.8 /WARNINGS=Alpha No Longer Flags REAL*16 2–12
2.6 New and Changed Features in V6.1 2–12
2.6.1 POINTER Statement Now Supported 2–12
2.6.2 Recursive Function References Now Permitted 2–12
2.6.3 New Non-Native Data in I/O CONVERT Options 2–14
2.6.4 New Argument List Inquiry Functions 2–15
2.6.5 LOC Intrinsic Function Alternative to %LOC Built-In 2–16
2.6.6 New /DEBUG=PARAMETERS Command Qualifier 2–16
2.6.7 Diagnostics For Possible Programming Errors 2–16
2.6.8 Directives Now Flagged Using /STANDARD=SOURCE_FORM 2–18
2.6.9 Run-Time Libraries Can Be Installed Separately 2–18
2.6.10 Machine Code Listing and Debugger Changes 2–18
2.6.11 HELP/MESSAGE Database for Fortran Compiler and Run-Time Diagnostics 2–19
2.7 Overview of Compaq Fortran 77 Version V6.0 2–19
2.7.1 COMMON Block Alignment Compatibility 2–21
2.7.2 Command Line Compatibility 2–21
2.7.3 VAX Language-Sensitive Editor Compatibility 2–21
2.7.4 Compatibility With VAX Common Data Dictionary 2–21
2.7.5 Compatibility With VAX DBMS 2–21
2.7.5.1 Using /DML and /PARALLEL Compile Command Qualifiers 2–22
2.7.5.2 COMMON block longword alignment 2–22
2.7.5.3 /CONTINUATIONS Qualifier May Be Necessary with /DML 2–22
2.8 New and Changed Features 2–22
2.8.1 New Command Qualifiers and Qualifier Parameters 2–22
2.8.1.1 /WARNINGS=[NO]ALIGNMENT Default Changed 2–23
2.8.1.2 /CONTINUATIONS Qualifier Obsolete 2–24
2.8.1.3 /HPO Qualifier and FORTRAN-HPO PAK Obsolete 2–24
2.8.1.4 New /MATH_LIBRARY=V5 Qualifier Keyword 2–24
2.8.2 FORT$LIBRARY Logical Name as Search List 2–24
2.8.3 FORT$INCLUDE Logical Name 2–25
2.8.4 Language Syntax 2–25
2.8.5 Non-native Data in I/O Feature 2–26
2.8.6 New Intrinsic Routines 2–26
2.8.7 Automatic Decomposition for Parallel Processing 2–27
2.8.8 Generation of VAX Vector Processor Code 2–28
2.8.9 Recursion and Automatic Storage 2–28
2.8.9.1 Automatic Allocation and Vectorization 2–28
2.8.10 Basic Linear Algebra Subroutines 2–30
2.8.11 Performance Optimization for Large Array Operations 2–30
2.8.12 Optimizations of Operations on COMPLEX Variables 2–30
2.8.13 Optimizations on References to Dummy Arguments and COMMON 2–30
2.8.14 Optimizations on Intrinsic Functions 2–31
2.8.15 Detection of Uninitialized Variables 2–31
2.8.16 Vectorization of DO-Loops with INTEGER*2 Control Variables 2–32
2.8.17 Design Processing Features 2–32

2.8.18 VOLATILE Allowed for RECORD Variables 2–32
2.8.19 Formal Arguments Allowed in NAMELIST Groups 2–32
2.8.20 Limit on Number of COMMON Blocks Increased 2–32
2.8.21 Support for NTT MIA Standard 2–32
2.8.22 Warnings for Features Not Supported on Alpha 2–33
2.8.23 DECwindows Compiler Interface Removed 2–33

3 Documentation and Documentation Corrections

3.1 *DEC Fortran Language Reference Manual* 3–1
3.1.1 Section 8.2.1 - Input Rules for FORMAT Statements 3–1
3.1.2 Section 11.2.6 - OPTIONS Directive 3–2
3.1.3 Table C-7 - BLAS Basic Set Routines 3–2
3.2 *DEC Fortran User Manual for OpenVMS VAX Systems* 3–2
3.2.1 Section 1.2.3 - New qualifiers 3–2
3.2.2 Section 1.2.3.2 - New /ASSUME=SOURCE_INCLUDE Keyword 3–2
3.2.3 Section 1.2.3.8 - /DEBUG Qualifier 3–2
3.2.4 Section 1.2.3.21 - /MATH_LIBRARY Qualifier 3–2
3.2.5 Section 1.2.3.27 - /STANDARD Qualifier 3–2
3.2.6 Section 1.2.3.31 - /WARNINGS Qualifier 3–2
3.2.7 Section 1.x - INCLUDE Files 3–3
3.2.8 Section 1.3.4.1 - User Supplied Default Libraries 3–3
3.2.9 Sections 8.1.1 and 8.1.2 - Sharing Images in Shareable Image Libraries 3–3
3.2.10 Table G-1 - Source Program Diagnostic Messages 3–3
3.2.11 Table G-4 - CRX Error Messages 3–9

A Compaq Fortran Maintenance Changes

Examples

2–1 Example of recursive function 2–13
2–2 Example of IARGCOUNT, IARGPTR and POINTER 2–15

Tables

2–1 Pascal DEBUG Syntax for Fortran Pointee Expressions 2–5
2–2 Variable Name Formats in Machine Code Listing 2–18
2–3 New Intrinsic Routines 2–26
3–1 Source Program Diagnostic Messages 3–4
3–2 CRX Messages 3–9
A–1 Compaq Fortran Maintenance Changes A–1

1

Compaq Fortran 77 Version 6.6-201 Release Notes

Note

The name of this product has been changed from DEC Fortran to DIGITAL Fortran 77 and again to Compaq Fortran 77. These release notes use the new name throughout; however the documentation, other than the Installation Guide, still refers to DEC Fortran. All three names refer to the same product.

Caution

If installed on an OpenVMS VAX system version earlier than V6.1, Compaq Fortran 77 provides a new version of the OpenVMS math library shareable image MTHRTL.EXE which is not backwards-compatible with OpenVMS VAX V6.0 (or earlier), nor with OpenVMS Alpha for images translated with DECmigrate. This affects *all* images linked on systems where Compaq Fortran 77 is installed, even those written in languages other than Fortran. Please make sure that all system users read and understand the information and instructions given in Section 1.3.3.

1.1 Overview of Compaq Fortran 77 Version 6.6-201

Compaq Fortran 77 Version 6.6-201 is an update of Compaq Fortran 77 V6.5-188 (formerly DIGITAL Fortran 77) which contains corrections to problems found in Version V6.5-188 and new features.

For a summary of changes included in this version, see Section 1.2 and Appendix A. For an overview of previous versions, see Chapter 2. For a summary of documentation changes, see Chapter 3. Change bars indicate information that is new or has changed since Version V6.5-188.

Compaq Fortran 77 Version V6.6 requires OpenVMS VAX (formerly VAX/VMS) Version V5.4 or higher. The Compaq Fortran 77 product kit may be installed whether or not any earlier version of Compaq Fortran 77, DIGITAL Fortran 77, DEC Fortran, VAX FORTRAN or VAX FORTRAN-HPO was previously installed. For complete installation details, see the *Digital Fortran Installation Guide for OpenVMS VAX Systems*. See also Section 1.3.1 for information on coexistence with earlier versions.

1.2 New and Changed Features in V6.6-201

This section provides highlights of new and changed features in Compaq Fortran 77 V6.6-201.

1.2.1 Detection of COMMON Name Conflict

The compiler now issues a COMNAMCONF diagnostic if a COMMON block has the same name as another global entity (routine, entry, BLOCK DATA subprogram) in the same program unit. The Fortran 77 standard prohibits two or more global entities from having the same name. While this is harmless on OpenVMS, it may cause unpredictable run-time errors on other platforms.

For portability, make sure that COMMON blocks have names that are unique among globals in the executable program. Note that Compaq Fortran 77 cannot detect conflicts between program units. This diagnostic can be disabled with /WARNINGS=NOUSAGE.

1.2.2 /WARNINGS=[NO]UNREACHABLE Implemented

The compiler now supports the command qualifier /WARNINGS=NOUNREACHABLE to disable diagnostics for unreachable code.

1.3 Compatibility

Compaq Fortran 77 Version V6.6 is compatible with all VAX processors running Version V5.4 or later of the OpenVMS operating system.

The following sections summarizes the compatibility issues, if any, between Compaq Fortran 77 Version V6.6 and previous versions of Compaq Fortran 77, DEC Fortran, VAX FORTRAN and VAX FORTRAN-HPO and versions of other DIGITAL layered products.

1.3.1 Coexistence With Earlier Versions

The VMSINSTAL version of the Compaq Fortran 77 installation procedure offers an option of preserving the previously installed version of Compaq Fortran 77. If this option is selected, the earlier version is renamed to FORTRAN$MAIN-Vnn.EXE where "Vnn" is the previous version number with the period removed. For example, Compaq Fortran 77 V6.5 would be renamed FORTRAN$MAIN-V65.EXE. To use the saved version of the compiler, define the logical name FORTRAN$MAIN to point to the desired version. For example:

```
$ DEFINE FORTRAN$MAIN FORTRAN$MAIN-V65
```

To select the currently installed version, deassign the logical name FORTRAN$MAIN or define it as FORTRAN$MAIN.EXE (the ".EXE" is required in this case.) It is not necessary to save earlier versions of the compiler message files.

If earlier versions of VAX FORTRAN and/or VAX FORTRAN-HPO were previously installed, installation of Compaq Fortran 77 does not replace the compiler and message file images used by the earlier versions.

To use VAX FORTRAN, define the logical name FORTRAN$MAIN as follows:

```
$ DEFINE FORTRAN$MAIN FORTRAN
```

To use FORTRAN-HPO, use the following command instead:

```
$ DEFINE FORTRAN$MAIN FORTRAN-HPO
```

In addition to the compiler images SYS$SYSTEM:FORTRAN.EXE and/or SYS$SYSTEM:FORTRAN-HPO.EXE, the old message files SYS$MESSAGE:FORTERR1.EXE and SYS$MESSAGE:FORTERR2.EXE must be present.

The earlier compilers will ignore new command qualifiers and qualifier values.

The PCSI version of the installation procedure will not automatically save a copy of the previous compiler. If you wish to preserve the older version, copy SYS$SYSTEM:FORTRAN$MAIN.EXE before installation.

1.3.2 Source Compatibility

Compaq Fortran 77 Version V6.6 accepts all valid Fortran language source which was accepted by earlier versions of Compaq Fortran 77, VAX FORTRAN and VAX FORTRAN-HPO. However, a number of new intrinsic routines have been added which may affect programs which have user-supplied routines with the same names. If these names conflict with those of user-supplied functions, it may be necessary to add an EXTERNAL statement for the user-supplied functions. See Sections 2.3.6, 2.6.4, 2.6.5 and 2.8.6 for more details.

Compaq Fortran 77 Version V6.6 also detects some errors in program sources that were not reported in previous versions. This may cause some programs to generate warnings or errors when compiled which previously compiled without diagnostics.

Some of the more significant errors that are now detected by the compiler are:

- The compiler now reports an INVLEXEME error if an internal file I/O statement specifies a parenthesized variable as the internal file specifier. An internal file specifier is required to be a *character scalar memory reference* or a *character array name reference*.
- The compiler now reports an UNDFUNVAL warning if a function's return value is not defined at one or more exit points.
- The compiler now reports an INVCONST error when it finds that the increment expression for a DO loop evaluates to zero. Previously, the compiler would not always report an error for this case. A zero loop increment would cause an infinite loop at run-time.
- If a program specifies a USEROPEN routine in an OPEN statement and the routine is declared explicitly as some type other than INTEGER*4, the compiler will now give a "Name previously used with conflicting data type" (INVTYPUSE) error. If the routine was not explicitly given a type, the compiler now assigns it the INTEGER*4 type, even if an IMPLICIT statement would specify a different type. However, if IMPLICIT NONE is in effect, and the type was not specified explicitly, then an IMPNONE error will be generated. Previously, the type conflict was not detected nor was a warning generated if IMPLICIT NONE was in effect. USEROPEN routines must return an INTEGER*4 function value.
- The compiler now gives an INVRECUSE error when an array or aggregate reference is passed as an expression in an actual argument list. Previously, it could generate incorrect code or cause the compilation to fail.
- The compiler and Run-Time Library now enforce the documented restrictions on the ranges of fields in FORMAT edit descriptors. Previously, incorrect results could be generated.

1.3.3 New Math Library Considerations

Compaq Fortran 77 provides new, revised versions of the VMS Math Library shareable images MTHRTL.EXE and VMTHRTL.EXE. (The shareable image UVMTHRTL.EXE, which is functionally identical to MTHRTL, is also provided.) Due to a significant number of new entry points defined in MTHRTL, required by Compaq Fortran 77, the global section match ident of that image was required to be changed. While previously linked images will run with the new MTHRTL, newly linked images will not run on other OpenVMS VAX systems (V6.0 or earlier) which do not have the new MTHRTL. In addition, images linked against the new MTHRTL which are then translated by the DECmigrate product will not run on OpenVMS Alpha systems. The following paragraphs explain how to compile and link applications so that they may run on these other systems.

The new support was included with OpenVMS VAX Version 6.1. If you install or reinstall versions of OpenVMS VAX prior to version 6.1, you must reinstall the Compaq Fortran 77 kit.

The installation procedure preserves the OpenVMS-supplied shareable images under the names FORTRAN$MTHRTL-VMS, FORTRAN$VMTHRTL-VMS and FORTRAN$UVMTHRTL-VMS. To link an image so that it will run on OpenVMS versions that lack the enhanced RTL images use the following method:

```
$ DEFINE/USER MTHRTL FORTRAN$MTHRTL-VMS
$ DEFINE/USER VMTHRTL FORTRAN$VMTHRTL-VMS
$ LINK program
%LINK-I-IDMISMCH, GSMATCH of 81,00800C in shareable image
   SYS$COMMON:[SYSLIB]MTHRTL-VMS.EXE;1
        differs from GSMATCH of 81,00800D in shareable image
        library
   SYS$COMMON:[SYSLIB]IMAGELIB.OLB;1
```

The LINK-I-IDMISMCH informational message can be ignored.

In addition, any Fortran modules should be compiled with the command qualifier /MATH_LIBRARY=V5. This causes the compiler to not use any of the new entry points, thus allowing error-free linking against the old MTHRTL, but the associated performance enhancements will be lost. This qualifier also allows Fortran object modules to be linked on other OpenVMS VAX systems which don't have the updated math library. See section Section 2.8.1.4 for more details.

1.3.4 Fortran Run-Time Library Considerations

Compaq Fortran 77 provides a new version of the OpenVMS VAX Fortran Run-Time Library shareable image SYS$LIBRARY:FORRTL.EXE and associated object modules to support the new non-native data in I/O feature. The VMS-supplied version is saved as SYS$LIBRARY:FORTRAN$FORRTL-VMS.EXE.

The new support is included in OpenVMS VAX Version V6.1. Until that version is installed, you must reinstall the Compaq Fortran 77 kit each time you install a new version of OpenVMS VAX.

Unlike the new version of the Math Run-Time Library images described above, the new version of FORRTL.EXE does not of itself prevent images linked against the new FORRTL.EXE from running on older versions of OpenVMS VAX as early as V5.0-1. The only situation in which an incompatibility will arise is where the application uses the CONVERT= keyword in an OPEN or INQUIRE statement or does an OPEN in a program unit compiled with the /CONVERT command or OPTIONS statement qualifier specifying a value other than NATIVE. See Section 2.8.5 for more details.

Of course, if the application also uses Math library routines, the restrictions noted in Section 1.3.3 apply.

See also Section 1.4.4 for a description of a known problem regarding linking against the new FORRTL.EXE.

1.3.5 Object Compatibility

When the default /NOPARALLEL command qualifier is used, object modules from Compaq Fortran 77 can be freely mixed with those from any previous version of Compaq Fortran 77, VAX FORTRAN, and VAX FORTRAN-HPO. There are no known compatibility problems with mixing these object modules with those of any other Digital compiler.

If any FORTRAN module in the application is compiled with /PARALLEL, there are no compatibility problems as long as all other modules in the application have also been compiled by Compaq Fortran 77 using the /PARALLEL qualifier.

For applications which are created by mixing FORTRAN modules compiled /PARALLEL with those from other languages, or with FORTRAN modules not compiled with /PARALLEL (including modules compiled with versions of VAX FORTRAN earlier than V5.0), see the chapter on Parallel Processing in the *DEC Fortran User Manual for OpenVMS VAX Systems* for detailed information and restrictions.

1.4 Known Problems and Issues

This section describes known problems and restrictions in this version of Compaq Fortran 77 and provides additional information on important issues.

1.4.1 Required Post-Installation Tasks on VMScluster

If Compaq Fortran 77 is installed on a VMScluster system, it is critically important that the post-installation tasks described in *Digital Fortran Installation Guide for OpenVMS VAX Systems* be performed. In particular, the command procedure SYS$UPDATE:FORTRAN$POST_INSTALL must be executed on all nodes of the cluster immediately after installation; this command file ensures that the latest installed versions of Run-Time Library images are used on all cluster nodes. Failure to follow this step can result in programs getting the error SHRIDMISMAT, "ident mismatch with shareable image" or other run-time errors. The installation procedure also warns the installer to perform this step.

1.4.2 Routines Called With Omitted Arguments

The new optimization for dummy arguments (see Section 2.8.13) may cause problems for applications which contain Fortran routines that can be called with some arguments omitted. Though omission of arguments is not supported by Digital Fortran, many applications have been written which took advantage of knowledge that the compiled code did not access the argument list for dummy arguments, other than those of type CHARACTER or used in adjustable array bounds expressions, until the argument was actually used during execution of the routine. This assumption is no longer valid, and these applications may now fail, typically with an access violation.

To disable the optimization on dummy arguments, specify the /ASSUME=DUMMY_ALIASES command line or OPTIONS statement qualifier. An alternative is to declare the specific dummy arguments to be VOLATILE or to use them as arguments to the %LOC built-in function. (Most existing subroutines

which can be called with omitted arguments use %LOC to test for a zero address; these will not require changes. However, if the test for omission is made solely on the basis of argument count or some other value, one of the above changes is necessary.) For the language requirements regarding Actual Argument and Dummy Argument Association, see the *DEC Fortran Language Reference Manual*.

1.4.3 Compiler Bugchecks with /VECTOR

There remain several known cases where the compiler fails with a bugcheck during the SCHED or BLOCK_SCHED phases for programs compiled with the /VECTOR command qualifier. If you encounter such a case, workarounds include compiling just the failing routine /NOVECTOR and inserting NOVECTOR directives before individual loops in the failing routine. It is not necessary to compile the entire application /NOVECTOR to work around this problem.

1.4.4 Linker OUTSIMG Error When COLLECT Linker Option Used

The updated FORRTL.EXE image provided with Digital Fortran includes message section definitions so that the new messages relating to the non-native data in I/O feature can be displayed if errors occur; these messages are not yet included in the standard message file provided by OpenVMS VAX.

There is one known case where these message sections create a problem during linking. If the user application includes object modules created by the OpenVMS MESSAGE compiler and uses the linker COLLECT options file statement to collect the message section PSECTs into an image cluster, the linker may report an error of the form:

```
%LINK-E-OUTSIMG, attempted store location %X0000B800 is outside
   image binary (%X00000000 to %X00000000)
   in psect MSG$AAAAAAAAAAA
-LINK-E-NOIMGFIL, image file not created
```

This error is caused by the linker attempting to collect both the user message PSECTs and the PSECTS in FORRTL into the same image cluster, which is invalid.

To avoid this problem, link explicitly against FORTRAN$FORRTL-VMS.EXE. The application will still run properly, even against the Digital Fortran supplied FORRTL, but the linker will not complain.

This problem is resolved in OpenVMS VAX V6.1.

1.4.5 Recursive Character Functions Not Supported

The compiler does not currently allow character functions to be referenced recursively.

1.4.6 Restrictions on Use of BLAS Routines

In some cases certain uses of the BLAS intrinsic routines can cause the compiler to fail with a bugcheck. The situations where this has been observed are:

- Combining use of an inlined BLAS routine with a non-inlined intrinsic or external routine in an expression, for example:

```
T = FUNC(X) + DDOT(NT2,Z(J, J), I1,D( J),I1)
```

- Use of an inlined BLAS routine in an implied DO loop in an I/O statement, for example:

```
      write( 12,3000) ((sdot(3,x(1,i),1,f(1,ia ),1),i=1,3),
     *                  ia=iaanf,iaend)
```

A workaround for both cases is to compile with /BLAS=NOINLNE. In the first case another workaround is to separate the expression into two separate assignments as follows:

```
T1 = FUNC(X)
T = T1 + DDOT(NT2,Z(J, J), I1,D( J),I1))
```

2

Release Notes for Previous Versions of Compaq Fortran 77

2.1 New and Changed Features in V6.5

This section provides highlights of new and changed features in Compaq Fortran 77 V6.5.

2.1.1 New DATE_AND_TIME Intrinsc

The non-standard Fortran intrinsic routines DATE and IDATE return date strings in a format using a two-digit year. This can cause problems for applications executing in the year 2000 or later. In order to provide a year-2000-compliant progam interface for obtaining the date, Compaq Fortran 77 now supports the DATE_AND_TIME intrinsic from the Fortran 90 standard. The calling sequence is as follows:

```
CALL DATE_AND_TIME ([date] ,[time] ,[zone] ,[values])

  date - character variable whose first 8 characters
         receive the date in the form yyyymmdd

  time - character variable whose first 10 characters
         receive the time in the form hhmmss.sss

  zone - character variable whose first 5 characters
         receive the timezone differential from UTC in
         the form ±hhmm or blank if the zone is unavailable.

  values - INTEGER*4 (8) array which into which is stored
           the following information:

    VALUES(1) = year
    VALUES(2) = month
    VALUES(3) = day
    VALUES(4) = timezone differential in minutes, or
         -2147483648 if unavailable
    VALUES(5) = hour
    VALUES(6) = minute
    VALUES(7) = second
    VALUES(8) = milliseconds
```

Any of the arguments may be omitted, but any preceding commas for omitted arguments must be specified. If any of the CHARACTER arguments are longer than the required length, the remaining characters are unchanged.

Because the OpenVMS system clock has a 10ms resolution, the number of milliseconds returned will always be a multiple of 10.

The timezone value is derived from the value of the logical name SYS$TIMEZONE_DIFFERENTIAL and is supported on OpenVMS VAX V6.0 and later. The timezone can be set using the SYS$MANAGER:UTC$CONFIGURE_TDF.COM procedure or through DECnet-PLUS (DECnet/OSI) configuration.

Run-Time Library support for DATE_AND_TIME is automatically installed along with the Compaq Fortran 77 compiler as object module FOR$DATE_AND_TIME in SYS$LIBRARY:STARLET.OLB. This routine will be included in a future version of OpenVMS VAX. If your application uses DATE_AND_TIME and you wish to link on another system, that other system must install the Run-Time Library support from the Compaq Fortran 77 kit.

As an alternative to DATE_AND_TIME, consider the OpenVMS date-time system services, such as SYS$ASCTIM.

2.2 New and Changed Features in V6.4

This section provides highlights of new and changed features in Compaq Fortran 77 V6.4.

2.2.1 STRUCTURE and COMMON Alignment

Compaq Fortran 77 now supports all STRUCTURE and COMMON alignment features of Digital Fortran for OpenVMS Alpha Systems, including:

- The ability to request that fields in a STRUCTURE be aligned on natural address boundaries.
- The ability to request that COMMON elements be aligned on natural address boundaries.
- The ability to request that the compiler warn when a STRUCTURE field falls falls on an unaligned address boundary.
- The ability to request that COMMON blocks have their length padded out to multiples of the COMMON block's alignment.

In addition, Compaq Fortran 77 supports the ability to selectively disable alignment warnings for specific STRUCTURE and COMMON declarations. This feature will be added to future versions of Digital Fortran on other platforms.

Note

The defaults for STRUCTURE alignment and alignment warnings have *not* changed and are different from the defaults used by Digital Fortran for OpenVMS Alpha Systems.

The following sections describe the specific alignment-related features in Compaq Fortran 77.

2.2.1.1 /ALIGN Compile Command Qualifier

Compaq Fortran 77 now supports the /ALIGN compile command qualifier using the same syntax as Digital Fortran for OpenVMS Alpha Systems. For further information about the syntax and semantics of the qualifier, see the *DEC Fortran Language Reference Manual* or type:

```
$ HELP FORTRAN /ALIGN
```

at the DCL prompt. Note that the default is that STRUCTUREs are packed, unlike Digital Fortran for OpenVMS Alpha which aligns fields in STRUCTUREs by default. COMMON blocks are packed by default on both platforms.

To provide for correct execution of programs which use packed structures from SYS$LIBRARY:FORSYSDEF.TLB, even in the presence of /ALIGN=STRUCTURES=NATURAL, declarations in FORSYSDEF.TLB have been modified to include bracketing CDEC$ OPTIONS directives to disable alignment for structures declared within.

If you specify /ALIGN=STRUCTURES=NATURAL and use INCLUDE files which declare structures that require packed allocation, be sure to bracket the INCLUDE line as follows:

```
CDEC$ OPTIONS /NOALIGN /WARN=NOALIGN
      INCLUDE 'include-file'
CDEC$ END OPTIONS
```

2.2.1.2 /WARNINGS=ALIGNMENT Supports STRUCTUREs

The /WARNINGS=ALIGNMENT compile command qualifier now controls alignment warnings for uses of structures with misaligned fields. In previous versions, only non-aggregate variables were checked for possible alignment warnings.

2.2.1.3 CDEC$ OPTIONS Directive

Compaq Fortran 77 now supports the CDEC$ OPTIONS (and CDEC$ END OPTIONS) directives as documented in the *DEC Fortran Language Reference Manual*.

In addition, the CDEC$ OPTIONS directive may optionally specify one of the following:

/WARNINGS=ALIGNMENT
/WARNINGS=NOALIGNMENT
/WARNINGS
/NOWARNINGS

/WARN is equivalent to /WARNINGS=ALIGNMENT, /NOWARN is equivalent to /WARNINGS=NOALIGNMENT. This qualifier enables or disables alignment warnings for STRUCTUREs and COMMONs declared within the scope of the CDEC$ OPTIONS directive. Note that alignment warnings are not given if /WARNINGS=NOALIGNMENT is specified (or defaulted) on the compile command line, even if warnings are enabled by a CDEC$ OPTIONS directive. This directive qualifier is useful for suppressing alignment warnings for structures you know contain misaligned fields.

This feature will be supported by future versions of Digital Fortran on other platforms.

2.2.1.4 CDEC$ PSECT MULTILANGUAGE Keyword

The CDEC$ PSECT directive now accepts the keywords MULTILANGUAGE and NOMULTILANGUAGE, in a manner compatible with Digital Fortran on other platforms. This controls whether or not the total length of a COMMON block is padded to a multiple of the block's PSECT alignment. The default is NOMULTILANGUAGE which specifies that no padding is to be done. If MULTILANGUAGE is specified, the COMMON block's length is padded to be an integral multiple of the associated PSECT alignment, up to octaword (16 bytes). For example, if a COMMON block contains three REAL*4 variables, and the default PSECT alignment of quadword (8 bytes) is in effect, the length of the COMMON block would be 12 bytes long with NOMULTILANGUAGE, but padded at the end out to 16 bytes if MULTILANGUAGE was specified.

The default for all COMMON blocks can be specified by using the /ALIGN=COMMONS=[NO]MULTILANGUAGE command line qualifier. Note that this setting is not available on the CDEC$ OPTIONS /ALIGN qualifier.

2.2.2 /ASSUME=BYTERECL Command and OPTIONS Qualifier

Compaq Fortran 77 now supports the /ASSUME=BYTERECL qualifier on the command line and on the OPTIONS statement. This controls the size of the unit used for the RECL= keyword in an OPEN or INQUIRE statement for unformatted files. The default, NOBYTERECL, specifies that RECL is in units of 4-byte longwords, compatible with current and past Digital Fortran products. If BYTERECL is specified, the unit for RECL is one byte, which is used on some non-Digital platforms.

Use of the non-default BYTERECL value requires that the application be running on a system which has the associated Fortran Run-Time Library support. This support is provided by installing one of the following:

Compaq Fortran 77 version V6.4 or later
OpenVMS VAX V7.0 or later
Fortran Run-Time Library ECO FORRTLVE03062. This can be obtained through the Internet at `http://www.service.digital.com/html/patch_public.html` - search for FORRTLVE03062.

If a program which specifies /ASSUME=BYTERECL is run on a system without the proper Run-Time Library support, it will fail with an INVARGFOR, Invalid argument to Fortran Run-Time Library error.

2.2.3 CDEC$ PSECT ALIGN= Keywords Supported

The CDEC$ PSECT ALIGN= specifier now supports the keywords BYTE, WORD, LONG, QUAD, OCTA and PAGE in addition to the integer constant expression previously allowed. These keywords are equivalent to the constants 0, 1, 2, 3, 4 and 9, respectively. This feature will be supported by a future version of Digital Fortran for OpenVMS Alpha Systems, where the keyword PAGE will be treated as if it were the constant 16 (the largest page size on OpenVMS Alpha is 2**16 bytes.)

2.2.4 DPROD Intrinsic Now Generic

The DPROD intrinsic function has been extended to operate as a generic function which can accept arguments of types REAL*4 or REAL*8 (both arguments must be the same type.) If the arguments to DPROD are REAL*8, the function result is of type REAL*16. Note that if the DPROD intrinsic name is passed as an actual argument, the routine passed is the version with a REAL*8 function result.

2.2.5 Use of POINTER Variables in DEBUG

Compaq Fortran 77 now correctly describes pointees, declared in a POINTER statement, in the Debug Symbol Table information for use with the OpenVMS Debugger. However, prior to OpenVMS VAX V7.1, the debugger does not contain full support for pointer-type variables in Fortran. Scalar numeric pointees can be examined by prefacing the pointee variable name with the "@" character. For example, given the following declarations:

```
REAL X
POINTER (P,X)
```

you could use the DEBUG command `E @X` to examine the contents of X (pointed to by P). If the pointee has a character, array or record type, however, the debugger will not properly access the contents. Until the debugger is enhanced in this area, a suitable workaround is to tell the debugger that the language is Pascal, by means of a `SET LANGUAGE PASCAL` DEBUG command, and use Pascal syntax, as follows:

- Place a caret "^" after the pointee variable name.
- Use square brackets "[ ]" instead of parentheses in array and character references.

As in Compaq Fortran 77, a period is used as a record field separator. Also, the debugger will properly use Fortran-style column-major order array indexing even though the language is set to Pascal. Note that the debugger does not support COMPLEX constants in expressions, but can examine COMPLEX variables.

The following table shows equivalent Pascal syntax for accessing various Fortran variable types.

Table 2–1 Pascal DEBUG Syntax for Fortran Pointee Expressions

Fortran Type	Pascal DEBUG Syntax
Whole variable X	X^
Character substring X(2:3)	X^[2:3]
Array reference X(2,3)	X^[2,3]
Field reference X.F	X^.F
Field array reference X(3).F	X^[3].F

As of OpenVMS VAX V7.1, the debugger understands Fortran pointers.

2.3 New and Changed Features in V6.3

This section provides highlights of new and changed features in Compaq Fortran 77 Version V6.3.

2.3.1 /ASSUME=SOURCE_INCLUDE Command Line Keyword

The compiler now supports the /ASSUME=[NO]SOURCE_INCLUDE command line qualifier keyword. This controls where the compiler looks for INCLUDE files (including text libraries) if a complete file specification is not given. The default is NOSOURCE_INCLUDE which causes the compiler to look for INCLUDE files in the user's current default device and directory. If SOURCE_INCLUDE is specified, the device and directory of the current source file is used, a behavior common on platforms other than OpenVMS.

For example, assume that DISK1:[JONES]PROG.FOR contains the following source line:

```
INCLUDE 'PROG1.INC'
```

and that the user's current default directory is DISK2:[SMITH]. The compile command

```
$ FORTRAN/ASSUME=SOURCE_INCLUDE DISK1:[JONES]PROG
```

will cause the compiler to look for PROG1.INC in DISK1:[JONES]. If /ASSUME=NOSOURCE_INCLUDE is specified (or the qualifier is omitted, since that is the default), the compiler will look for PROG1.INC in DISK2:[SMITH].

If the logical name FORT$INCLUDE is defined, it overrides the setting of [NO]SOURCE_INCLUDE.

2.3.2 /DEBUG=PARAMETERS=NONE Command Line Keyword

The compiler now accepts a new keyword NONE for the /DEBUG=PARAMETERS= compile command qualifier keyword. If NONE is specified, the compiler does not write information about PARAMETER constants to the debug symbol table. Other possible values are USED and ALL. Note that if /DEBUG=NOSYMBOLS is in effect, no debug symbol table information of any type is written.

2.3.3 /WARNINGS=[NO]INFORMATIONAL Command Line Keyword

The compiler now accepts a new [NO]INFORMATIONAL keyword for the /WARNINGS compile command qualifier. If NOINFORMATIONAL is specified, compiler diagnostic messages of informational severity are not displayed. The default is INFORMATIONAL, causing informational messages to be displayed. Note that NOGENERAL suppresses both warning and informational messages, regardless of the setting of [NO]INFORMATIONAL.

2.3.4 Expanded syntax for Directives

In previous versions of Digital Fortran, directives were recognized only when they began with CDEC$ or CPAR$ in column 1. This syntax is dependent on the Fortran-77 fixed source form. With the introduction of Fortran 90 and its free source form, it became desirable to use directives in free-form source which does not recognize a C in column 1 as a comment.
Although Compaq Fortran 77 does not support free-form source, the compiler now accepts more forms of directives so that a source program with directives can be written such that it is acceptable in both free-form and fixed-form.

The syntax of a directive tag is now the following sequence of characters:

1. One of the following comment introducers:
   - 'C' or 'c' or '*' in column 1
   - '!' in column 1 or preceded by whitespace only
2. 'DEC$' or 'PAR$' (case is ignored, PAR$ recognized only if /PARALLEL specified)
3. A blank or tab character

The following are examples of valid directives:

```
CDEC$ TITLE 'title'
cdec$   TITLE 'title'
*DEC$  TITLE 'title'
!dec$  TITLE 'title'
         !dec$ TITLE 'title'
```

The following are examples of invalid directives:

```
CDEC$TITLE 'title'   ! No blank or tab after $
ddec$ TITLE 'title'  ! D comment introducer not allowed
#dec$ TITLE 'title'  ! # comment introducer not allowed
      CONTINUE  !DEC$ TITLE 'title'  ! Non-whitespace before tag
```

Directives cannot be continued.

To write source code that is acceptable both as fixed-form and free-form, follow these rules:

- Treat blanks as significant.
- Put statement labels in columns 1 through 5.
- Start statement field in column 7.
- Use only '!' as a comment introducer. Place anywhere except in column 6.
- Use only '&' as a continuation indicator. Place it both in column 73 of the initial line and in column 6 of the continuation line.
- Do not use /EXTEND_SOURCE.

2.3.5 Sign Extension of Untyped Bit Constants

The compiler now properly sign-extends bit constants when used in contexts where they are treated as INTEGER*2 or BYTE types. For example:

```
BYTE X
PARAMETER (X = X'FF')
INTEGER*4 I4
I4 = X
TYPE *,I4
```

Previously, the result of the TYPE statement would be to display 255, rather than -1. The compiler was not properly honoring the declared data type of the constant X and sign-extending when assigning it to an INTEGER*4 value.

Although not related to this correction of behavior, it should be noted that when context requires that a bit constant be assigned a data type, the compiler requires that bits to be truncated be zero; it does not do a signed conversion to a smaller datatype. For example, given the following:

```
INTEGER*2 I2
I2 = X'FFFFFFFF'
```

the compiler will issue a "Non-zero digits truncated in constant" warning for the assignment even though signed conversion would be valid.

2.3.6 AND, OR, XOR, LSHIFT and RSHIFT Intrinsic Functions Supported

The intrinsic generic functions AND, OR, XOR, LSHIFT and RSHIFT, previously described in the *DEC Fortran Language Reference Manual* as being supported on RISC and Alpha platforms only, are now supported for VAX. Except for RSHIFT, each of these is equivalent to an existing generic intrinsic (IAND, IOR, IEOR, ISHFT) and are provided for increased compatibility with other platforms. RSHIFT is equivalent to ISHFT with a negated second argument.

2.3.7 FORSYSDEF.TLB Modules for Callable Utility Routines

The following new modules for the OpenVMS Callable Utility Routines are provided in SYS$LIBRARY:FORSYSDEF.TLB:

ACLEDIT$ROUTINES - Access Control List Editor
CLI$ROUTINES - Command Line Interpreter
CONV$ROUTINES - CONVERT Utility
DCX$ROUTINES - Data Compression and Expansion
EDT$ROUTINES - EDT Editor
FDL$ROUTINES - File Definition Language Editor

LBR$ROUTINES - Librarian Utility
MAIL$ROUTINES - MAIL Utility
PSM$ROUTINES - Print Symbiont Modification Routines
SMB$ROUTINES - Symbiont Routines
TPU$ROUTINES - TPU Utility

2.4 Overview of Compaq Fortran 77 Version V6.2

Compaq Fortran 77 Version V6.2 is a minor update of Compaq Fortran 77 Version V6.1 which contains corrections to problems found in Version V6.1, as well as several new features.

2.5 New and Changed Features in V6.2

This section provides highlights of new and changed features in Compaq Fortran 77 Version 6.2.

2.5.1 Short Source Lines and /PAD_SOURCE Qualifier

The Fortran-77 language source form is defined in terms of 72-character source lines; the standard does not define the behavior when a source line shorter than 72 characters is compiled. Since blanks are generally ignored in Fortran source, this rarely matters, but when character, Hollerith or RAD50 constants are continued across two or more source lines, implementation differences in treatment of short source lines can affect interpretation of the program. Consider the following statement as part of a source file with varying length records (in this example, "b" is used to indicate blanks):

```
bbbbbbTYPE *,'ABC
bbbbb+123'
```

The first source record is 17 characters long. What is displayed when this statement executes? All Digital Fortran compilers honor the record length of short records, and would consider the first source character in the second record ("1") to immediately follow the last source character in the first record ("C"), and would cause

```
ABC123
```

to be printed. However, Fortran compilers on many non-Digital platforms automatically pad short source lines out to column 72, so the same statement processed by these compilers would print

```
ABCbbbbb
```

```
      CHARACTER*(*) LONG_CONST
      PARAMETER (LONG_CONST = 'This is a very long '//
     + 'character constant which is safely continued '//
     + 'across lines')
      CHARACTER*100 LONG_VAL
      DATA LONG_VAL /LONG_CONST/
```

Identifiers longer than 6 characters are used here for clarity; in all other ways, this method is standard Fortran-77.

Note that Hollerith and RAD50 constants cannot be built up in this manner. Hollerith constants should be converted to character constants. RAD50 constants are rarely long enough to require splitting across record boundaries; if necessary, extend the constant to column 72 before continuing it on the next line. Be aware that explicit blanks at the end of source records may be removed by text editors on some platforms.

Digital Fortran will now issue an informational diagnostic if a character, Hollerith or RAD50 constant is continued across two or more source lines AND one or more of the source lines in the current statement was shorter than the statement field width (72, or 132 with /EXTEND_SOURCE). This diagnostic, CHACONCONTD, can be suppressed with /WARNINGS=NOUSAGE.

When compiling source programs that have been ported from other platforms, if you find that the program requires short source lines to be padded with blanks for proper execution, you can specify the new /PAD_SOURCE compile command qualifier. If /PAD_SOURCE is specified, the compiler treats short source lines as if they were padded with blanks to the statement field width (72 or 132 columns, depending on /EXTEND_SOURCE.) The default is /NOPAD_SOURCE to retain compatibility with previous Digital Fortran compilers.

2.5.2 Pointee Can Be Called Routine

A POINTER statement may name as a *pointee* an identifier which is referenced as a routine. When the routine is called, the routine address is taken from the pointer variable. This can be useful as an alternative to the Run-Time Library routine LIB$CALLG.

The following example demonstrates the use of this feature.

```
EXTERNAL LIB$PUT_OUTPUT
POINTER (P,R)
P = %LOC(LIB$PUT_OUTPUT)
CALL R('Hello')
END
```

2.5.3 CDEC$ OPTIONS and END OPTIONS Directives Ignored

The compiler now silently ignores the CDEC$ OPTIONS and CDEC$ END OPTIONS directives which are recognized by Digital Fortran on other platforms. This feature was requested by users who objected to the informational messages displayed by the VAX compiler when compiling common sources that used these directives. A future version of Digital Fortran for OpenVMS VAX may provide full support for these directives.

2.5.4 New CDEC$ ALIAS Directive

The compiler now supports a new directive, CDEC$ ALIAS, which provides the ability to specify that the external name of an external subprogram is different than the name by which it is referenced in the calling subprogram.

The syntax is:

CDEC$ ALIAS internal-name, external-name

where *internal-name* is the name of the subprogram as used in the current program unit and *external-name* is either a quoted character constant or a *symbolic name*. If external-name is a character constant, the value of that constant is used as the external name for the specified internal-name. The character constant is used as it appears, with no modifications for case or removal of punctuation characters. If external-name is a symbolic name, the symbolic name (in upper case) is used as the external name for the specified internal-name. Any other declaration of the specified symbolic name is ignored for the purposes of the ALIAS directive. Note that the OpenVMS linker may have restrictions on what may appear in an external name and that upper and lower case in external names is significant.

For example, in the following program:

```
      PROGRAM ALIAS_EXAMPLE
CDEC$ ALIAS ROUT1, 'ROUT1A'
CDEC$ ALIAS ROUT2, 'rout2_'
CDEC$ ALIAS ROUT3, rout3A
      CALL ROUT1
      CALL ROUT2
      CALL ROUT3
      END
```

the three calls would be to external routines named ROUT1A, rout2_ and ROUT3A.

This feature can be useful when porting code between OpenVMS and UNIX systems where different routine naming conventions are in use. For example, a typical UNIX convention for calls from Fortran is to downcase the routine name and add a trailing underscore. If you wrote a C routine intended to be called from Fortran, you would have to name it in accordance to this convention, so for example ROUT2 would be coded in C as rout2_. If this application were ported to OpenVMS, where routine names are not generally modified, the result would be that the linker would not resolve the reference to ROUT2. By adding the CDEC$ ALIAS directive, you can specify an alternate name without recoding the application.

Another use of the CDEC$ ALIAS directive is to allow a recursive reference to a function as an actual argument. See Section 2.6.2 for an example.

This directive is available in Digital Fortran for other Digital platforms.

2.5.5 # Accepted as Comment Introducer

The compiler now accepts the '#' character in column 1 as indicating a whole-line comment. The purpose of this enhancement is to allow the use of C preprocessors on Fortran source; most such preprocessors add lines with original source file information with a '#' in column 1. Although Compaq Fortran 77 does not currently make use of this information, it may in the future and therefore you should avoid using the '#' comment introducer for other purposes.

The VAX C preprocessor can be used for Fortran source by means of the CC /PREPROCESS_ONLY command. If you also have DEC C for OpenVMS VAX installed, you *must* use CC/VAXC/PREPROCESS_ONLY to select VAX C as the DEC C preprocessor does not work with Fortran source. Any informational messages from the VAX C preprocessor can be ignored.

2.5.6 Reduction in Compiler Virtual Memory Usage

The compiler now uses significantly less virtual memory when compiling large program units than in previous versions. Though the memory use is still higher than with versions prior to V6.0, reductions of 15 to 20 percent as compared to V6.1 have been seen. The additional memory is used for enhanced optimization capability.

If insufficient virtual address space is available, large compilations may fail. A more serious problem can result if a user's page file quota (AUTHORIZE quota PGFLQUO) is too large for the available page file space. This will cause the entire system to slow down as OpenVMS tries to find page file space for the increasing virtual memory. Even if the compilation completes, the system will remain very sluggish for a significant time afterwards.

A process' virtual address space is mainly constrained by the SYSGEN parameter VIRTUALPAGECNT and the process page file quota. Compilations of program units with 20,000 executable source lines or more may need 30,000 or more pages of virtual address space. The peak virtual address space used is shown in the compilation summary at the end of the listing file. If you find that compilations are excessively slow, look to see if the page file space is becoming exhausted.

2.5.7 Negative Constant Values in FORSYSDEF.TLB

In the system symbol definition library FORSYSDEF.TLB, there are a small number of constants defined which have negative values. Previously, all constants were defined in FORSYSDEF as typeless hexadecimal values so as to avoid typing problems. However, these constants could not be assigned to INTEGER*2 or BYTE variables or record fields without the compiler issuing an EXCDIGTRU error because it thought that significant bits were being truncated. For example:

```
INCLUDE '($JPIDEF)'

STRUCTURE /Itmlst/
  UNION
    MAP
      INTEGER*2 BufLen
      INTEGER*2 Code
      INTEGER*4 Addr
      INTEGER*4 RetLen
    END MAP
    MAP
      INTEGER*4 End_List
    END MAP
  END UNION
END STRUCTURE

RECORD /Itmlst/ Itemlist(2)

Itemlist(1).Code = JPI$_CHAIN

END
```

The symbol JPI$_CHAIN has the value -1, but it was defined in FORSYSDEF.TLB as 'FFFFFFFF'X. This caused the EXCDIGTRU error.

Compaq Fortran 77 now builds FORSYSDEF.TLB such that negative constants which are not bit-masks are defined using decimal notation, thus avoiding the problem. If you are writing code that may be compiled on systems which don't have this change, use the following workaround:

```
Itemlist(1).Code = IAND(JPI$_CHAIN,'FFFF'X)
```

2.5.8 /WARNINGS=Alpha No Longer Flags REAL*16

Digital Fortran for OpenVMS Alpha and Digital UNIX ALpha systems now support the REAL*16 declaration, therefore Digital Fortran for OpenVMS VAX Systems has been modified to no longer flag usage of the REAL*16 datatype when the /WARNINGS=Alpha compile command qualifier is specified.

2.6 New and Changed Features in V6.1

This section provides highlights of new and changed features in Compaq Fortran 77 Version 6.1.

2.6.1 POINTER Statement Now Supported

The POINTER statement is now supported as described in the *DEC Fortran Language Reference Manual*. See Example 2–2 for an example using POINTER.

Note

If a *pointee* is an adjustable array and the program unit contains one or more ENTRY statements, variables used in the adjustable array bounds expression(s) must be present in all entry point argument lists or be in COMMON, otherwise results are unpredictable. For example:

```
      SUBROUTINE BADSUB (N)
      REAL P_ARRAY(N)
      POINTER (P,P_ARRAY)
.
.
.
      ENTRY BADENTRY (J)  ! N not defined at entry
.
.
.
```

It is not necessary that the pointer be defined at all entry points, only that the bounds information be determinable.

A *pointee* may also be an assumed-size array.

2.6.2 Recursive Function References Now Permitted

The documented restriction that recursive function references were not permitted, even with /RECURSIVE specified, has been removed. However, within a function subprogram, the name of the function when used other than in a reference (call) to that function, refers to the return value variable and not the function itself. Consider the following modification of routine AST_ROUTINE from example D.2 from the *DEC Fortran User Manual for OpenVMS VAX Systems*:

```
      INTEGER FUNCTION AST_ROUTINE
   .
   .
   .
! Reenable the AST
!
      QIO_STATUS = SYS$QIOW (,
     1  %VAL(CHANNEL),
     2  %VAL(IO$_SETMODE .OR. IOS$M_CTRLCAST),
     3  IOSB,
     4  AST_ROUTINE,,,,,)
   .
   .
   .
```

Since the use of the name AST_ROUTINE in the SYS$QIOW call is not a function reference, the function return variable is passed which is not desired. For uses of this nature, it would still be necessary to use a separate routine to reference the function name as an external symbol.

However, with the CDEC$ ALIAS feature added in version V6.2, this could be successfully rewritten as follows:

```
      INTEGER FUNCTION AST_ROUTINE
CDEC$ ALIAS AST_ROUTINE, LCL_AST_ROUTINE
   .
   .
   .
! Reenable the AST
!
      QIO_STATUS = SYS$QIOW (,
     1  %VAL(CHANNEL),
     2  %VAL(IO$_SETMODE .OR. IOS$M_CTRLCAST),
     3  IOSB,
     4  LCL_AST_ROUTINE,,,,,)
   .
   .
   .
```

The CDEC$ ALIAS directive creates an alternate name LCL_AST_ROUTINE which can be used to reference the enclosing function rather than its return value. See Section 2.5.4 for more information.

Example 2–1 is an example showing a valid use of recursive function references:

Example 2–1 Example of recursive function

```
      PROGRAM FACTORIAL_TEST
!
! This is just a simple example program showing recursion in
! FORTRAN.
!
! The program computes and displays the factorials of the
! integers 1:12; factorial of 13 is too big for a longword
! integer.
!
      INTEGER FACTORIAL
```

(continued on next page)

Example 2–1 (Cont.) Example of recursive function

```
      DO N=1,12
        WRITE (6,*) N,' factorial is ',FACTORIAL(N)
      END DO
      END

      OPTIONS /RECURSIVE
      INTEGER FUNCTION FACTORIAL (N)
!
! The Factorial of an integer N is N * N-1 * N-2 ... * 1.
! This is often written as N!.
!
! If N is greater than 1, call ourselves recursively,
! passing N-1 to perform the calculation.  Note that
! each recursive call to FACTORIAL will get its own
! copy of N.
!
! If N is 1 or less, the result is 1.  (We assume that
! N is not negative.)
!
      IF (N .GT. 1) THEN
        FACTORIAL = FACTORIAL(N-1) * N
      ELSE
        FACTORIAL = 1
      END IF

      RETURN
      END
```

2.6.3 New Non-Native Data in I/O CONVERT Options

To support the exchange of 16-byte IEEE-style floating type data with Digital Fortran for OpenVMS Alpha and Digital UNIX, the non-native data in I/O support has been enhanced to provide for conversion between this new type, called *X_float*, and the native VAX H_float type. If one of the existing IEEE keywords (LITTLE_ENDIAN or BIG_ENDIAN) is specified as the CONVERT type, X_float values in the external data file will be converted to or from VAX H_float when a REAL*16 variable is written or read during unformatted I/O. The Alpha X_float type is little-endian, meaning that the least-significant fraction bit is in the lowest-addressed byte, but BIG_ENDIAN can be used to access REAL*16 data from compatible non-Digital systems which use the big-endian form, where the least-significant fraction bit is in the highest-addressed byte.

In addition, Digital Fortran for OpenVMS Alpha will support the combination of the IEEE X_float type along with VAX F/D or F/G float. These combinations can be specified using the new CONVERT keywords 'FDX' and 'FGX', respectively. These keywords can also be used in the /CONVERT compile command or OPTIONS statement qualifier, as well as the FOR$CONVERT_nnn logical name.

The Alpha X_float type has a 15-bit exponent and 113-bit fraction and has a similar range and precision to VAX H_float.

2.6.4 New Argument List Inquiry Functions

The compiler now supports two new intrinsic functions for returning information about the argument list to a called routine, IARGCOUNT and IARGPTR.

IARGCOUNT takes no arguments; it returns the count of actual arguments supplied by the caller of the routine as an INTEGER*4 value. Note that if the routine is a CHARACTER or COMPLEX*16 function that the count includes the extra argument used for passing the return function value.

IARGPTR takes no arguments; it returns the INTEGER*4 address of the beginning of the actual argument list; that is, the contents of register AP upon entry to the routine. Note that the actual argument list is architecture-specific; on VAX systems it is an array of longwords whereas on Alpha systems it is an array of quadwords.

IARGCOUNT and IARGPTR can be used in the executable section of any program unit, including a main program. They cannot be used in statement functions or in declarations, nor can their names be passed as actual arguments.

Note

Omission of arguments in a call to a Fortran routine violates the Fortran-77 standard and may be non-portable. Digital Fortran does not support omission of arguments which have CHARACTER type or which are used in adjustable array bounds expressions. See also Section 1.4.2 for additional requirements when arguments are omitted.

Example 2–2 demonstrates the use of the IARGCOUNT and IARGPTR intrinsics as well as of the POINTER statement:

Example 2–2 Example of IARGCOUNT, IARGPTR and POINTER

```
C     PROGRAM ARGLIST_TEST
      INTEGER*4 X,Y,Z
      X = 10
      Y = 20
      Z = 100
      PRINT 800, %LOC(X), %LOC(Y), %LOC(Z)
800   FORMAT (' Argument addresses: ',3(1x, Z8.8))
      CALL ARGLIST_SUB (X, Y, Z)
      END

      SUBROUTINE ARGLIST_SUB
      ! Declare pointer to argument list
      POINTER (ARGLIST_P, ARGLIST_ENTRY)
      INTEGER*4 ARGLIST_ENTRY
      ! Declare pointer to individual argument
      POINTER (VALUE_P, VALUE)
      INTEGER VALUE

      ! Get base address of argument list and skip over count
      ! Use SIZEOF to be somewhat architecture independent
      !
      ARGLIST_P = IARGPTR()+SIZEOF(ARGLIST_P)
```

(continued on next page)

Example 2–2 (Cont.) Example of IARGCOUNT, IARGPTR and POINTER

```
      ! Display count of arguments and then, for each argument
      ! passed, display its value.
      !
      PRINT *,'Called with ',IARGCOUNT(), ' arguments'
      DO I = 1, IARGCOUNT()
        VALUE_P = ARGLIST_ENTRY ! Get address of argument
        PRINT 900, I, VALUE_P, VALUE
900     FORMAT ( ' Argument ', I2, ' address = ', Z8.8,
     1    ', contents = ', I5)
 ! Get address of next argument position
        ARGLIST_P = ARGLIST_P + SIZEOF(ARGLIST_P)
        END DO
      RETURN
      END
```

2.6.5 LOC Intrinsic Function Alternative to %LOC Built-In

For improved compatibility with Fortran implementations on non-Digital platforms, Version V6.1 adds the LOC intrinsic function which has the same purpose and can be used in the same manner as the %LOC built-in function. See the description of %LOC in the *DEC Fortran Language Reference Manual* for additional information and restrictions. The LOC function name may not be passed as an actual argument.

2.6.6 New /DEBUG=PARAMETERS Command Qualifier

In previous versions, the compiler would make known to the debugger only those PARAMETER constants which were actually used in a program unit. This was to prevent the object module and thus the executable image file from becoming very large if a large number of PARAMETER constants were defined, even if few were used (for example, if the DECwindows Motif INCLUDE files were used.)

In response to a number of users who requested the ability to have the compiler make known to the debugger all defined PARAMETER constants, a new keyword, PARAMETERS, has been added to the /DEBUG command qualifier.

If PARAMETERS=USED is specified, the compiler makes known to the debugger only those PARAMETER constants which were actually used. This is the default. If PARAMETERS=ALL is specified, all PARAMETER constants are made known to the debugger.

As of Compaq Fortran 77 Version 6.3, the NONE keyword is also accepted. See Section 2.3.2 for details.

/DEBUG, without any value, is now equivalent to /DEBUG=(PARAMETERS=USED,SYMBOLS,TRACEBACK). /DEBUG=ALL is equivalent to /DEBUG=PARAMETERS=ALL,SYMBOLS,TRACEBACK), as is /DEBUG=PARAMETERS.

2.6.7 Diagnostics For Possible Programming Errors

Version 6.1 adds four new keywords to the /WARNINGS qualifier:

- [NO]UNCALLED - If UNCALLED is in effect, the compiler issues informational diagnostics for any statement functions which are defined but never called. Checking for uncalled statement functions does not occur when /DESIGN=PLACEHOLDERS is in effect and placeholders are seen. NOUNCALLED suppresses such diagnostics. UNCALLED is the default.

- [NO]UNINITIALIZED - The compiler's detection of uninitialized variables, introduced in Version 6.0 (see Section 2.8.15) can now be controlled from the command line. If UNINITIALIZED is in effect, the compiler warns of uses of uninitialized variables. Checking for uninitialized variables does not occur when /DESIGN=PLACEHOLDERS is in effect and placeholders are seen, nor when /NOOPTIMIZE is in effect. NOUNINITIALIZED suppresses such warnings. UNINITIALIZED is the default.
- [NO]UNUSED - If UNUSED is in effect, the compiler issues informational diagnostics for variables which are declared but not used. Variables in COMMON or which appear in an EQUIVALENCE statement are not flagged. Checking for unused variables does not occur when /DESIGN=PLACEHOLDERS is in effect and placeholders are seen. NOUNUSED suppresses such diagnostics. UNUSED is the default.
- [NO]USAGE - If USAGE is in effect, the compiler issues informational diagnostics when it detects programming usages which, though allowed by Digital Fortran, are often indicative of a programming error. At present, these are:
  - Passing an identifier, which was previously used as an intrinsic routine, as an actual argument without having named that identifier in an INTRINSIC statement. The FORTRAN–77 standard requires such intrinsics to be named in an INTRINSIC statement, but Digital Fortran relaxes that rule. In many cases, such use indicates a programming error where an array name was intended instead. If an intrinsic routine was intended, name it in an INTRINSIC statement.
  - A branch into an inner DO-loop or an IF block. Although this might be valid if the FORTRAN 66 "extended range of a DO loop" feature was being used, it generally indicates a programming error. A common case involves two or more DO loops which share a common termination. In such cases, the shared termination statement is considered to belong to the innermost DO loop. For example:

```
      DO 100 I=1,10
      IF (condition) GOTO 100
        DO 100 J=1,10
   .
   .
   .
100   CONTINUE
```

    In this example, the shared loop termination label 100 is considered to belong to the innermost loop, thus the GOTO is invalid. This error can lead to unpredictable results, or sometimes, programs which appear to work but which break when moved to other platforms or some other change is made.

  NOUSAGE suppresses such diagnostics. USAGE is the default.

2.6.8 Directives Now Flagged Using /STANDARD=SOURCE_FORM

The compiler now issues the "Extension to FORTRAN–77: nonstandard statement type" (EXT_STMT) diagnostic for CDEC$ and CPAR$ directives only if /STANDARD=SOURCE_FORM is in effect. In previous versions, directives, which are statements appearing as a special form of comment, would be flagged if /STANDARD=SYNTAX was in effect. The change was made so as to more reasonably reflect the degree of standards conformance in the source program. Since directives look like comments, they are not interpreted as statements by non-Digital compilers. Note that if /STANDARD is specified without any keywords, /STANDARD=(SYNTAX,SEMANTICS,NOSOURCE_FORM,NOMIA) is used.

A related change is that if SOURCE_FORM is specified, SYNTAX is implied. Previously, /STANDARD=SOURCE_FORM would mean NOSYNTAX,NOSEMANTICS.

2.6.9 Run-Time Libraries Can Be Installed Separately

The installation procedure now allows the user to install only the updated Run-Time Libraries; previously the Run-Time Libraries were provided only if the compiler was also installed. This option is useful if it is desired to run programs linked against the enhanced Run-Time Libraries on other OpenVMS VAX systems which do not have the compiler installed, or to reinstall the Run-Time Library support after a new version of OpenVMS VAX was installed. See the Installation Guide for more details.

2.6.10 Machine Code Listing and Debugger Changes

The machine code section of the listing file has been enhanced to indicate variables which have been allocated to registers. In VAX FORTRAN V5, the compiler did not list the names of such variables, simply denoting them as "Rn". Digital Fortran V6.0 would often use the name of the variable instead, but this could be confusing if the reader was not aware that the variable was in a register. The compiler now suffixes such variable names with "%Rn".

Table 2–2 shows how different types of variables are shown in the machine code listing. Note that the machine code listing is meant as informational to the reader, and is not intended to be processed by the VAX MACRO assembler.

Table 2–2 Variable Name Formats in Machine Code Listing

Name Format	Explanation
VARNAME	Variable in COMMON
VARNAME(AP)	Formal argument
VARNAME(FP)	Local variable on stack
VARNAME(R11)	Local static variable
VARNAME(Rn)	COMMON or formal accessed by base register
VARNAME%R5	Local variable in register
VARNAME r	Real part of expanded COMPLEX variable
VARNAME i	Imaginary part of expanded COMPLEX variable

In many cases, the compiler will create a local copy of formal arguments and COMMON variables and will use the notation of local variables for them.

Another change, which was introduced in VAX FORTRAN-HPO, is that the compiler can now tell the debugger when a variable was allocated to a register. Previously, the debugger would give a message that the variable was "optimized away" for such cases. This makes it somewhat easier to debug optimized code, but the debugger information for such variables is not completely reliable and Digital still recommends using the /NOOPTIMIZE compile command qualifier, where possible, when debugging.

2.6.11 HELP/MESSAGE Database for Fortran Compiler and Run-Time Diagnostics

OpenVMS VAX V6.0 provides a new command HELP/MESSAGE which provides simple access to the description of error messages and other diagnostics, including the ability to automatically display a description of the most recent error (whose status value is in the DCL symbol $STATUS). The Digital Fortran product provides the file SYS$HELP:FORTRAN$MSGHLP.MSGHLP$DATA which supplies descriptions of the Fortran compiler and run-time diagnostics for use with HELP/MESSAGE.

To add the Fortran messages to the list of known HELP/MESSAGE databases, you must define the logical name `MSGHLP$LIBRARY` to be a search list that names each of the available databases, including the OpenVMS VAX supplied database SYS$HELP:MSGHLP$LIBRARY.MSGHLP$DATA. This logical name can be defined in the system startup procedure. For example:

```
$ DEFINE/SYSTEM MSGHLP$LIBRARY SYS$HELP:MSGHLP$LIBRARY,-
   SYS$HELP:FORTRAN$MSGHLP
```

See the *OpenVMS System Messages: Companion Guide for Help Message Users* for more information on the HELP/MESSAGE feature.

2.7 Overview of Compaq Fortran 77 Version V6.0

Compaq Fortran 77 Version V6.0 for OpenVMS VAX (Compaq Fortran 77) is a new, major release of the VAX FORTRAN product. Its name has been changed to reflect the high degree of compatibility between it and Digital Fortran on other Digital platforms such as OpenVMS Alpha. It is a compatible superset of both VAX FORTRAN Version 5.9 and VAX FORTRAN High-Performance Option (FORTRAN-HPO) Version 1.3.

This section briefly describes the new and changed features of Compaq Fortran 77 Version 6.0. For more details, refer to the indicated section of the Release Notes and the Compaq Fortran 77 documentation set as listed in Chapter 3.

Compaq Fortran 77 offers the following major enhancements and changes as compared to VAX FORTRAN-HPO Version 1.3:

- Generation of reentrant (or recursion-capable) code and automatic (stack-based) allocation of data (Section 2.8.9)
- The ability to read and write numeric data in non-native formats using unformatted I/O (Section 2.8.5)
- An option to generate specially optimized code, which can significantly reduce the number of pagefaults and improve application performance, for loops which traverse large arrays (Section 2.8.11)
- Improved generated code for references to dummy arguments and variables in COMMON (Section 2.8.13)

- Improved generated code for operations involving COMPLEX variables (Section 2.8.12)
- Improved generated code for intrinsic functions (Section 2.8.14)
- Vectorization of DO loops with INTEGER*2 loop control variables (Section 2.8.16)
- Vectorization of additional intrinsic functions (AMOD, ATAN2, ATAND2, SIGN) and vectorization of COMPLEX intrinsic functions (Section 2.8.14)
- Reporting of uninitialized variables (Section 2.8.15)
- Removal of need for /CONTINUATIONS compile command qualifier (Section 2.8.1.2)
- Removal of need for /HPO compile command qualifier and FORTRAN-HPO Product Authorization Key (PAK) (Section 2.8.1.3)
- Change in default from /WARNINGS=ALIGNMENT to /WARNINGS=NOALIGNMENT (Section 2.8.1.1)
- Support for the definition of the logical name FORT$LIBRARY as a search list (Section 2.8.2)
- Support for the definition of the logical name FORT$INCLUDE to specify a default location for INCLUDE files (Section 2.8.3)
- An increase of the limit on the number of named COMMON blocks from 250 to 508. (Section 2.8.20)
- The ability to provide diagnostics for language features not supported by Digital Fortran on Alpha platforms. (Section 2.8.22)
- Support for NTT Technical Requirement TR550001, Multivendor Integration Architecture (MIA) Version 1.1, Division 2, Part 3-2, Programming Language FORTRAN (Section 2.8.21)

In addition, Compaq Fortran 77 provides the following enhanced features of VAX FORTRAN-HPO Version 1.3 as compared to VAX FORTRAN Version 5.9:

- Automatic decomposition of DO loops for parallel execution on VAX multiprocessor systems (Section 2.8.7)
- Automatic vectorization of DO loops for execution on VAX processors that support the VAX Vector Architecture (Section 2.8.8)
- Inline code expansion of the Basic Linear Algebra Subroutines (BLAS) Level 1 and Level 1 Extended (Section 2.8.10)
- The /WARNINGS=TRUNCATED_SOURCE qualifier which enables warnings for source lines which are longer than the statement field width (Section 2.8.1)
- The /TERMINAL=STATISTICS qualifier which enables informational messages about each compiled program unit to be displayed to the terminal (Section 2.8.1)

2.7.1 COMMON Block Alignment Compatibility

The default memory alignment for COMMON blocks is quadword alignment; in versions of VAX FORTRAN prior to V5.5 it was longword alignment (unless the COMMON block contained REAL*8 or COMPLEX*16 variables). (But see also Section 2.7.5.2.)

In most cases, the difference is not visible to a correctly-written application. Only applications which depended on there being no more than three bytes of unused space between consecutive COMMON blocks will behave differently. In some cases, this is due to an error in the program, but some applications use a specially named COMMON block that is intended to directly follow another one in order to determine the total size of the first block.

If the default of quadword alignment is not acceptable, there are two methods of obtaining longword alignment. The first is to use the CDEC$ PSECT directive to explicitly specify longword alignment for each COMMON block declared. This should be done in each program unit where the COMMON block is used. For example:

```
CDEC$ PSECT /MY_COMMON/ ALIGN=2
```

To specify blank COMMON, use "//" in the directive. See the *DEC Fortran Language Reference Manual* for more information on the CDEC$ PSECT directive.

Another method is to use a linker options file and to specify the PSECT_ATTR statement for each COMMON block to indicate the alignment. For example:

```
PSECT_ATTR=MY_COMMON,LONG
```

To specify blank COMMON, use the PSECT name *$BLANK*. See the *VMS Linker Utility Manual* for more information on linker options files.

2.7.2 Command Line Compatibility

The command line definition for Compaq Fortran 77 6.0 is a compatible superset of that for previous versions of VAX FORTRAN and VAX FORTRAN-HPO. Note that the /CONTINUATIONS and /HPO qualifiers, which were recognized by previous versions, are now ignored. See Section 2.8.1 for more details.

2.7.3 VAX Language-Sensitive Editor Compatibility

VAX Language-Sensitive Editor Version V3.0 or later must be installed on the compiling system if the /DESIGN=COMMENTS compile command qualifier is used.

2.7.4 Compatibility With VAX Common Data Dictionary

Compaq Fortran 77 supports VAX Common Data Dictionary definitions in DMU format. It supports the CDO format available in VAX CDD/Plus V4.0 or later through a compatibility interface. VAX CDD/Plus audit trails are not generated by Compaq Fortran 77. For more information, refer to the *DEC Fortran User Manual for OpenVMS VAX Systems* and the CDD/Repository documentation.

2.7.5 Compatibility With VAX DBMS

The following are known restrictions for applications using the VAX DBMS product.

2.7.5.1 Using /DML and /PARALLEL Compile Command Qualifiers

Because of the restriction on performing I/O operations within a parallel loop (see the chapter on Parallel Processing in the *DEC Fortran User Manual for OpenVMS VAX Systems*), care must be taken when using parallel loops with data base applications. Applications using the /DML qualifier should not use the CPAR$ DO_PARALLEL directive for any loop that executes data base operations. Such use will cause unpredictable results due to inconsistent I/O control state among the processes executing the loop iterations.

2.7.5.2 COMMON block longword alignment

COMMON blocks declared in subroutines compiled with /DML qualifier still have the longword alignment attribute. That is, the base of a COMMON block is aligned on a longword boundary.

2.7.5.3 /CONTINUATIONS Qualifier May Be Necessary with /DML

Although the Digital Fortran compiler no longer requires the /CONTINUATIONS qualifier to be used to raise the limit on continuation lines above the standard 19, the Fortran DML preprocessor, part of the VAX DBMS product and invoked with the FORTRAN/DML command, does still require this qualifier. If you are using /DML and have more than 19 continuation lines, specify /CONTINUATIONS=nn, where nn is an integer value between 19 and 99, on the FORTRAN/DML command line. This restriction may be removed in a future version of VAX DBMS.

2.8 New and Changed Features

This section provides highlights of new and changed features in Compaq Fortran 77

2.8.1 New Command Qualifiers and Qualifier Parameters

Several new qualifiers and qualifier parameters to the FORTRAN command have been implemented to invoke and control the new features mentioned in the previous section. In addition, the /CONTINUATIONS and /HPO qualifiers are now ignored; see Sections 2.8.1.2 and 2.8.1.3 for more details.

The new qualifiers and qualifier parameters are listed below. See the *DEC Fortran User Manual for OpenVMS VAX Systems* for complete details on qualifier syntax and meanings.

- /ASSUME=(ACCURACY_SENSITIVE,DUMMY_ALIASES)

  Specifies assumptions the compiler is allowed to make when considering potential optimizations.

- /BLAS=(INLINE,MAPPED)

  Controls recognition of and code generation for routines from the Basic Linear Algebra Subroutines Level 1 group.

- /CHECK=(ALIGNMENT,ASSERTIONS)

  Controls compiler generated code for recovery from vector alignment faults and checking of ASSERT conditions.

- /DIRECTIVES=(DEPENDENCE)

  Controls recognition of dependence directives.

- /ERROR_LIMIT=n

Controls whether compilation terminates after a specified number of errors has been found.

- /MATH_LIBRARY=(ACCURATE|FAST,V5)

  Controls selection of vector math library routines and whether or not the compiler restricts use of math library routines to those supplied by OpenVMS V5.x. (See section Section 2.8.1.4 for more information on the V5 keyword.)

- /OPTIMIZE=LEVEL=n

  Specifies the optimization level.

- /PARALLEL=(AUTOMATIC,MANUAL)

  Controls whether the compiler peforms automatic decomposition of DO loops for parallel processing and/or enables recognition of the manual decomposition directives from VAX FORTRAN V5.

- /RECURSIVE

  Controls generation of reentrant (recursive) code and data.

- /SHOW=(DATA_DEPENDENCES,LOOPS)

  Controls presentation of data dependences in the listing and diagnostics file and loop structure in the listing.

- /STANDARD=MIA

  Controls whether the compiler performs standards compliance against the NTT MIA standard.

- /SYNCHRONOUS_EXCEPTIONS

  Controls when vector arithmetic exceptions are reported.

- /TERMINAL=STATISTICS

  Specifies display of the module name and combined code and data bytes as each module is compiled.

- /VECTOR

  Controls generation of VAX vector processor code.

- /WARNINGS=(ALIGNMENT,Alpha,INLINE,TRUNCATED_SOURCE)

  Specifies whether the compiler provides warnings of: unaligned variables in COMMON or EQUIVALENCE statements, uses of language features not supported by Digital Fortran on Alpha platforms, references to BLAS routines which were not expanded inline, source lines longer than the statement field width.

2.8.1.1 /WARNINGS=[NO]ALIGNMENT Default Changed

The default for /WARNINGS=[NO]ALIGNMENT has been changed to NOALIGNMENT so as to reduce unnecessary messages when vector processing is not being used. /WARNINGS=ALIGNMENT is recommended for compilations which declare data which is to be used by vector instructions, for applications which are to be ported to the Alpha environment, and for users who wish to eliminate unaligned data for maximum scalar performance on VAX systems.

2.8.1.2 /CONTINUATIONS Qualifier Obsolete

In previous versions of VAX FORTRAN, the /CONTINUATIONS compile command qualifier was used to specify the minimum number of continuation lines which the compiler would accept. The default was 19 and the maximum was 99. This qualifier is now obsolete and is ignored if specified; the compiler will always accept a minimum of 99 continuation lines of 132 characters each. The number of continuation lines accepted may be greater if some of the lines are shorter than 132 characters. The compiler will still issue a diagnostic if more than 19 continuation lines (of any length) are found and the /STANDARD=SOURCE_FORM compile command qualifier is specified.

If the FDML preprocessor, a component of the VAX DBMS product and invoked with the FORTRAN/DML command, is used, the /CONTINUATIONS qualifier may still be necessary. See Section 2.7.5.3 for more information.

2.8.1.3 /HPO Qualifier and FORTRAN-HPO PAK Obsolete

The /HPO compile command qualifier, which was used to select between the VAX FORTRAN and VAX FORTRAN-HPO compilers, is now obsolete and is ignored if specified. Also, the FORTRAN-HPO Product Authorization Key (PAK) is no longer necessary in order to use the automatic vectorization or decomposition features.

2.8.1.4 New /MATH_LIBRARY=V5 Qualifier Keyword

The /MATH_LIBRARY qualifier accepts a new [NO]V5 keyword which is not currently documented in the *DEC Fortran User Manual for OpenVMS VAX Systems*. This keyword controls whether or not the compiler restricts itself to using only those Math Library routines which were used by VAX FORTRAN V5.x and supplied by OpenVMS VAX V5.0-V6.0 (V5.4 for vector routines.) The default is NOV5 which allows the compiler to use the new routines for intrinsic function references as described in Section 2.8.14. If V5 is specified, the compiler does not use the new routines so that the object module may be linked against an OpenVMS VAX V5.x math library; the result will be performance comparable to that obtained with VAX FORTRAN-HPO.

2.8.2 FORT$LIBRARY Logical Name as Search List

The FORT$LIBRARY logical name is used to define an additional default library for INCLUDEs from text libraries. In previous releases, if FORT$LIBRARY were defined with multiple translations (a *search list*), only the first translation would be used. As of Version 6.0, the compiler will recognize the multiple translations and will search each library specified before searching the system default library FORSYSDEF.TLB. Within the list of translations, file name defaulting occurs as it would for a plus-list of input files specified on the command line with the /LIBRARY qualifier. For example, given the following definition:

```
$ DEFINE FORT$LIBRARY MYDISK:[MYDIR]LIB1,[MYDIR2]LIB2,LIB3
```

The compiler would search the following libraries in order:

```
MYDISK:[MYDIR]LIB1.TLB
MYDISK:[MYDIR2]LIB2.TLB
MYDISK:[MYDIR2]LIB3.TLB
```

If any of the specified libraries do not exist or are not valid text libraries, a warning diagnostic will be displayed. See the *DEC Fortran User Manual for OpenVMS VAX Systems* for more information on using text libraries.

2.8.3 FORT$INCLUDE Logical Name

Compaq Fortran 77 Version 6.0 provides the ability to specify a default location for INCLUDE files. If the logical name FORT$INCLUDE is defined, the compiler uses 'FORT$INCLUDE:.FOR' as the default file specification for non-library files and 'FORT$INCLUDE:.TLB' for library files specified in an INCLUDE statement. Definition of FORT$INCLUDE does not affect default library files specified by the logical name FORT$LIBRARY nor the system default library FORSYSDEF.TLB. FORT$INCLUDE may be defined as a search list.

Note

If the current default disk and directory is to be searched, 'SYS$DISK:[ ]' must be explicitly included in the search list definition of FORT$INCLUDE.

For example, given the existence of the following files:

```
MYDISK:[DIR1]INC1.FOR
MYDISK:[DIR2]INC2.FOR,LIB1.TLB
MYDISK:[DIR3]INC2.FOR
```

and the logical name definition:

```
$ DEFINE FORT$INCLUDE MYDISK:[DIR1],[DIR2],[DIR3]
```

the following INCLUDE statements would access the indicated files:

```
INCLUDE 'INC1'       ! MYDISK:[DIR1]INC1.FOR
INCLUDE 'INC2'       ! MYDISK:[DIR2]INC2.FOR
INCLUDE 'LIB1(MOD1)' ! MYDISK:[DIR2]LIB1.TLB
```

2.8.4 Language Syntax

Several new language statements, directives and syntax extensions have been implemented; a summary is provided below. Some of the syntax extensions have been provided for greater compatibility with Digital Fortran on other platforms; others were introduced in VAX FORTRAN-HPO. See the *DEC Fortran Language Reference Manual* for more details.

- ASSERT statement and directive
- AUTOMATIC statement
- INIT_DEP_FWD directive
- INTEGER*1 type declaration (synonym for BYTE)
- NOVECTOR directive
- POINTER statement
- STATIC statement
- '1010'B syntax for binary typeless constants, '10AB'Z as alternative to '10AB'X for hexadecimal typeless constants
- B'1010', O'2377', X'10AB', Z'10AB' forms of typeless constants

2.8.5 Non-native Data in I/O Feature

Compaq Fortran 77 Version 6.0 includes the ability to read and write numeric data which is in non-native formats using unformatted I/O. The formats available, in addition to the native VAX floating and little-endian integer types are: IEEE little-endian floating and integers (compatible with Alpha and RISC DECstation systems), IEEE big-endian floating and integers (Sun, IBM RS-6000, etc.), CRAY big-endian floating and integers and IBM System\370 big-endian floating and integers.

To enable this feature for individual logical units opened for unformatted I/O use the new CONVERT= keyword in an OPEN statement. The INQUIRE statement also allows a CONVERT= keyword to inquire of the current conversion type. The /CONVERT compile command or OPTIONS statement qualifier provides a default conversion type for all units opened in the program unit. It is also possible to define a logical name at run-time to specify the convert type for individual program units.

If an application uses the CONVERT= OPEN or INQUIRE keyword, or uses /CONVERT and OPEN, the program will fail with an "Invalid argument to FORTRAN Run-Time Library" error if it is run on an OpenVMS VAX system which does not include the enhanced Fortran Run-Time Library support provided with Compaq Fortran 77. If /CONVERT is used but a unit is implicitly opened, or if the logical name method is used, no error will occur but neither will the feature be enabled if the program is run on a system without the new support. To test in a program whether or not the non-native data in I/O support is available, execute an INQUIRE statement with a CONVERT= keyword for any unit. If no error occurs (IOSTAT=0 or no ERR= branch), then the support is present.

For more information on OPEN and INQUIRE, see the *DEC Fortran Language Reference Manual*. For more information on the /CONVERT qualifier and use of logical names to access this feature, see the *DEC Fortran User Manual for OpenVMS VAX Systems*.

Caution

Applications which use this feature cannot currently be translated by DECmigrate to run on OpenVMS Alpha systems.

2.8.6 New Intrinsic Routines

A number of new intrinsic routines are recognized by the Compaq Fortran 77 compiler. If your application has user-supplied routines with the same names as intrinsic routines, you should add EXTERNAL declarations for those routines in modules which use them, otherwise the compiler will attempt to recognize them as intrinsics. The names of the new intrinsic routines are shown in Table 2–3 and are described in more detail in the *DEC Fortran Language Reference Manual*.

Table 2–3 New Intrinsic Routines

ISAMAX	IDAMAX	ICAMAX	IZAMAX
ISAMIN	IDAMIN	ICAMIN	IZAMIN

(continued on next page)

Table 2–3 (Cont.) New Intrinsic Routines

ISMAX	IDMAX	ICMAX	IZMAX
ISMIN	IDMIN	ICMIN	IZMIN
SAMAX	DAMAX	CAMAX	ZAMAX
SAMIN	DAMIN	CAMIN	ZAMIN
SASUM	DASUM	CASUM	ZASUM
SAXPY	DAXPY	CAXPY	ZAXPY
SCOPY	DCOPY	CCOPY	ZCOPY
SDOT	DDOT	CDOTC	ZDOTC
		CDOTU	ZDOTU
SMAX	DMAX	CMAX	ZMAX
SMIN	DMIN	CMIN	ZMIN
SNORM2	DNORM2	SCNORM2	DZNORM2
SNRM2	DNRM2	SCNRM2	DZNRM2
SNRSQ	DNRSQ	SCNRSQ	DZNRSQ
SROT	DROT	CSROT	ZDROT
SROTG	DROTG	CROTG	ZROTG
SSCAL	DSCAL	CSCAL	ZSCAL
		CSSCAL	ZDSCAL
SSET	DSET	CSET	ZSET
SSUM	DSUM	CSUM	ZSUM
SSWAP	DSWAP	CSWAP	ZSWAP
SVCAL	DVCAL	CVCAL	ZVCAL
		CSVCAL	ZDVCAL
SZAXPY	DZAXPY	CZAXPY	ZZAXPY
HFIX	IMAG	ZABS	ZCOS
ZEXP	ZLOG	ZSIN	ZSQRT

2.8.7 Automatic Decomposition for Parallel Processing

The Compaq Fortran 77 compiler has the ability to automatically determine where it is effective and safe to execute DO-loops in parallel on VAX multiprocessor systems. This feature, previously introduced in VAX FORTRAN-HPO, is selected by specifying the compile command qualifier /PARALLEL=AUTOMATIC. The directed (manual) decomposition feature which was provided by VAX FORTRAN V5 is selected by the /PARALLEL=MANUAL qualifier. Both automatic and manual decomposition may be selected for a given compilation. See the *DEC Fortran User Manual for OpenVMS VAX Systems* and *DEC Fortran Performance Guide for OpenVMS VAX Systems* for more details.

2.8.8 Generation of VAX Vector Processor Code

The /VECTOR compile command qualifier causes the compiler to generate vectorized code for DO-loops when it is effective and safe to do so; this feature was previously introduced in VAX FORTRAN-HPO. The vector code can execute on VAX processors that support the VAX Vector Architecture (the VAX 6000-400 series, VAX 6000-500 series and VAX 9000 series with optional VAX vector processor boards), or on any VAX system running OpenVMS VAX V5.4 or later with the *VAX Vector Instruction Emulator Facility (VVIEF)* enabled. For more information on vectorization, see the *DEC Fortran User Manual for OpenVMS VAX Systems* and the *DEC Fortran Performance Guide for OpenVMS VAX Systems*. For more information on VVIEF, see the *VMS V5.4 New Features Manual*.

2.8.9 Recursion and Automatic Storage

The /RECURSIVE compile command or OPTIONS statement qualifier causes the compiler to generate recursion-capable or reentrant code and data, which permits a given routine to be reentered while a call to it is currently active. When the /RECURSIVE qualifier is in effect, the compiler's default memory allocation is changed from *static*, in which local variables and compiler-generated data structures such as descriptors and argument lists are allocated at fixed locations, to *automatic*, in which these items are allocated on the stack each time the routine is entered. Automatic allocation prevents conflicts between multiple calls to the same routine at the same time; a situation which might occur if, for example, multitasking software such as DEC Ada or DECthreads was in use.

The /RECURSIVE qualifier also allows a call to a subroutine or a function from within that subroutine or function; such use is invalid if /NORECURSIVE, the default, is in effect.

Automatic allocation can also be selected for individual variables even if /NORECURSIVE is in effect by means of the AUTOMATIC specification statement. Similarly, static allocation can be forced for individual variables by means of the STATIC statement. See the *DEC Fortran Language Reference Manual* for more details on AUTOMATIC and STATIC.

Automatic allocation is not permitted for variables named in EQUIVALENCE, SAVE or STATIC statements, which are data-initialized, or are named as the *pointee* in a POINTER statement.

/RECURSIVE is not permitted if /PARALLEL is in effect. See the *DEC Fortran User Manual for OpenVMS VAX Systems* for more information on the /RECURSIVE qualifier.

2.8.9.1 Automatic Allocation and Vectorization

The requirement that double-precision data used by VAX vector instructions be quadword aligned poses a problem when such data is to be allocated on the stack, either implicitly due to the /RECURSIVE qualifier being specified or due to an AUTOMATIC declaration. As the VAX architecture guarantees only longword alignment for the stack upon entry to a subroutine, the compiled code must force the stack to be quadword aligned. This is accomplished by a special instruction sequence which is generated when one or more double-precision or double-complex variables is to be stack-allocated and the /VECTOR qualifier is specified. The instruction sequence appears in the machine code listing as follows:

```
0000  SUB::
0000        .WORD       ^M<IV,R2,R3,R4,R5,R6,R11> ❶
0002        BITL        #^X7, SP ❷
0005        BEQL        L$1
0007        CALLG       (AP), L$0 ❸
000B        RET
000C        NOP ❹
000D        NOP
000E  L$0:
000E        .WORD       ^M<IV>
0010  L$1: ❺
0010        MOVAL       $LOCAL, R11
0017        SUBL2       #20, SP
001A        MOVC3       #12, $LOCAL(R11), -20(FP)
```

❶ This is the entry mask of the routine or main program entry point, as seen by an external caller. It specifies all of the registers to be saved before the routine is entered.

❷ The stack pointer register is tested to see if it is quadword aligned; that is, a multiple of 8. If it is, the routine's regular prologue code is branched to at label L$1.

❸ If the stack is not quadword aligned, a call is made to a special entry mask at L$0 which specifies no registers to be saved. As the fixed part of the stack frame is an odd number of longwords (5), the stack will now be quadword aligned.

❹ Two non-executed NOP instructions are inserted to force the L$1 branch target to quadword alignment, which improves performance when the branch to it is taken. The compiler specifies quadword alignment for the $CODE PSECT.

❺ The normal routine prologue code begins here. After the base register for statically-allocated storage is loaded, the next two instructions allocate the required space on the stack and initialized with any link-time constant data (descriptors, argument lists, etc.) if needed.

In order to reduce the overall performance impact of this additional code, the compiler tries to ensure that the alignment call is performed as few times as possible during the life of the application. It does this by forcing the stack to be non-quadword aligned (but longword aligned) after the stack area is allocated and ensuring that the register save mask causes an even number of registers to be saved. This will tend to keep the stack quadword aligned for nested calls, assuming that the called routines also use automatic storage and are compiled with /VECTOR.

There are some important considerations to be aware of when using automatic storage and vectorization:

- If recursion or automatic storage is used in any of the program units, all program units should be compiled with the /VECTOR qualifier to ensure proper stack alignment (and to reduce the need for alignment to take place.)
- Exception handlers which count call frames may need to be aware of the possibility of the presence of an extra call frame.

2.8.10 Basic Linear Algebra Subroutines

The Compaq Fortran 77 compiler recognizes as intrinsics the routines comprising the Basic Linear Algebra Subroutines library, Level 1 and Level 1 Extended; this feature was first introduced in VAX FORTRAN-HPO. In most cases, the compiler generates inline code for references to the BLAS routines; this code can be further vectorized and/or decomposed for improved performance if the /VECTOR or /PARALLEL qualifiers are in effect. For more information, see the *DEC Fortran Language Reference Manual* and the *DEC Fortran Performance Guide for OpenVMS VAX Systems* for more details.

2.8.11 Performance Optimization for Large Array Operations

In many applications which perform operations on large arrays, performance is often limited by the behavior resulting from the typical pattern of memory access which causes numerous page faults. The Compaq Fortran 77 compiler can analyze program units and, when it determines that it is safe and effective to do so, can rearrange DO-loops and insert "chunking" loops so as to significantly reduce the number of memory pages being accessed in close succession and hence improve performance by reducing page faults. This optimization is done automatically when /VECTOR is specified, and it is now also available for scalar processors. The new /OPTIMIZE=LEVEL=4 compile command qualifier enables this optimization.

When using /OPTIMIZE=LEVEL=4, it is recommended that you also specify /DIAGNOSTICS and /SHOW=(LOOPS,DATA_DEPENDENCE) so that the compiler can report on inhibitors to this additional level of optimization. Not all applications will benefit from this optimization, but those which do can see significant elapsed time improvements. See the *DEC Fortran Performance Guide for OpenVMS VAX Systems* for more information.

2.8.12 Optimizations of Operations on COMPLEX Variables

In previous versions of VAX FORTRAN, most operations on complex-typed variables were performed by calls to the OpenVMS Run-Time Library. Compaq Fortran 77 Version V6.0 now does many of these operations, such as complex multiply, inline, and separates the real and imaginary parts of complex variables for further optimization.

2.8.13 Optimizations on References to Dummy Arguments and COMMON

The Compaq Fortran 77 compiler can now make local copies of dummy arguments and/or variables in COMMON blocks for improved performance. Previously, the generated code would always reference such variables through the argument list or COMMON block. The compiler is careful to always copy values back or fetch new values when the language semantics require it, but some applications may have taken advantage of the previous implementation's limitation in this area and may produce incorrect results.

For dummy arguments, it is important to specify /ASSUME=DUMMY_ALIASES or to declare the variable as VOLATILE if a dummy argument can be addressed through some path other than the argument name; for example, if a dummy argument is also a COMMON variable, or if the routine can be called with some arguments omitted. By default, Compaq Fortran 77 assumes that the application conforms to the FORTRAN–77 standard in that it does not alias dummy arguments and that all actual arguments agree in order, number and data type (or structure, for record arguments), with their corresponding dummy

arguments. See the *DEC Fortran User Manual for OpenVMS VAX Systems* for more information on the /ASSUME=DUMMY_ALIASES option.

For variables in COMMON, it is important to name in a VOLATILE statement any variables or COMMON blocks that can be referenced (either read or written) other than by direct reference or during a routine call. For example, if a variable in COMMON is referenced in an OpenVMS AST routine, condition handler or exit handler, or is in a shared global section, you must declare as volatile that variable, or the COMMON block to which it belongs. See the *DEC Fortran Language Reference Manual* for more information on the VOLATILE statement.

2.8.14 Optimizations on Intrinsic Functions

Compaq Fortran 77 performs additional optimizations for references to many intrinsic functions. A side-effect of these optimizations is that more statement functions which reference these intrinsics can be expanded inline.

- Inline and/or vectorized code can be generated for references to the SIGN intrinsic with integer operands.
- New higher-performance *JSB* Mathematics Library entry points are used for references to the AMOD, ATAN, ATAN2, CABS, CCOS, CEXP, CLOG, COSH, CSIN, CSQRT, SINH and TANH intrinsics, as well as for the SIGN intrinsic when real operands are specified. This optimization is disabled if the compile command qualifier /MATH_LIBRARY=V5 is specified.
- The compiler can now vectorize references to the ATAN2, CCOS, CEXP, CLOG, COSH, CSIN, CSQRT, MOD (integer and real), SINH and TANH intrinsics. This optimization is disabled if the /MATH_LIBRARY=V5 compile command qualifier is specified.

Note

The values returned by the CEXP intrinsic may differ in the least significant bit from those returned by earlier versions of VAX FORTRAN, but the new values are more accurate.

2.8.15 Detection of Uninitialized Variables

The Compaq Fortran 77 compiler will now issue a FORT-W-USEUNIVAR warning when it detects a use of a local scalar variable for which it has determined that the variable is uninitialized at the point of use. The compiler cannot detect all cases of uninitialized variables, and does not produce a warning for variables which are named in a SAVE or VOLATILE statement, data-initialized, or equivalenced to a variable which is data-initialized, nor in a program unit for which /DESIGN=PLACEHOLDERS was in effect and placeholders were seen. The warnings are not issued if /NOOPTIMIZE or /WARNINGS=NOUNINITIALIZED is in effect. (/NOOPTIMIZE prevents the compiler from collecting the information it needs in order to properly determine a variable's initialization status.)

A warning is given only once for a given variable in a program unit. The warning message includes the line number on which the uninitialized use occurred; if the /DIAGNOSTICS qualifier was also specified and the diagnostics are reviewed in the Language-Sensitive Editor, the GOTO SOURCE (CTRL/G) command will position the editor at the offending use.

It is particularly important to find and correct uses of uninitialized variables when automatic storage is used, as the initial values of automatic variables are unpredictable, in contrast to that of static variables which are initialized to zero.

2.8.16 Vectorization of DO-Loops with INTEGER*2 Control Variables

The Compaq Fortran 77 Version V6.0 compiler can now vectorize DO-loops which have INTEGER*2 control variables. Previously in VAX FORTRAN-HPO, such loops were not eligible for vectorization.

2.8.17 Design Processing Features

Compaq Fortran 77 supports the design processing features of VAX Language-Sensitive Editor/Source Code Analyzer. These features allow you to embed design information in Fortran source, compile program units which contain pseudocode and other LSE placeholders, and generate documentation, help text and LSE syntax templates from your application source. See the description of the /DESIGN qualifier in the *DEC Fortran User Manual for OpenVMS VAX Systems* for more details.

2.8.18 VOLATILE Allowed for RECORD Variables

The VOLATILE declaration is now allowed for variables declared in RECORD statements; previously, this was not allowed. It is important to declare as volatile any variable which can be modified by means other than an assignment statement or being passed as an actual argument. Variables in COMMON blocks should also be declared volatile if they can change at times other than during routine calls.

2.8.19 Formal Arguments Allowed in NAMELIST Groups

Routine formal arguments may now appear in NAMELIST groups; previously, this was not allowed.

2.8.20 Limit on Number of COMMON Blocks Increased

In previous versions, the compiler allowed a maximum of 250 named COMMON blocks. This limit has been raised to 508; if blank COMMON is also used, a total of 509 COMMON blocks are allowed.

2.8.21 Support for NTT MIA Standard

Compaq Fortran 77 Version 6.0 provides the necessary support to comply with Nippon Telephone and Telegraph (NTT) Technical Requirement TR550001, Multivendor Integration Architecture (MIA) Version 1.1, Division 2, Part 3-2, Programming Language FORTRAN. The primary change needed to provide this support was the addition of the /STANDARD=MIA qualifier to enable standards checking against the MIA requirements. The HFIX and IMAG intrinsic function names were also added. See the *DEC Fortran User Manual for OpenVMS VAX Systems* for more information on /STANDARD=MIA.

2.8.22 Warnings for Features Not Supported on Alpha

The new /WARNINGS=Alpha qualifier causes the compiler to issue warnings whenever a language feature which is not supported by Digital Fortran on Alpha platforms is used. These features are: RAD50 constants and the PDP–11 compatibility routines ASSIGN, CLOSE, ERRSET, ERRTST, FDBSET, IRAD50, RAD50, R50ASC, USEREX.

2.8.23 DECwindows Compiler Interface Removed

The DECwindows Compiler Interface, a method of selecting compile options through a graphical interface activated by the DECwindows FileView application, is no longer provided. Digital will investigate alternate methods of providing similar functionality in the future.

3
Documentation and Documentation Corrections

The Compaq Fortran 77 documentation set includes the following:

- *DEC Fortran Language Reference Manual* (AA-PU45B-TK)
- *DEC Fortran User Manual for OpenVMS VAX Systems* (AA-PUYPA-TE)
- *DEC Fortran Performance Guide for OpenVMS VAX Systems* (AA-PUYQA-TE)
- *Digital Fortran Installation Guide for OpenVMS VAX Systems* (AA-PUYRB-TE)
- *Read Before Installing or Using Digital Fortran Version 6.5 for OpenVMS VAX Systems* (AV-PUYSF-TE)

Digital Fortran Installation Guide for OpenVMS VAX Systems and the Read Before letter are available with the Compaq Fortran 77 media on the Consolidated Online Distribution (ConDist) CD in ASCII and PostScript format.

All documents except the Read Before Installing letter are available on Consolidated Online Documentation (ConOLD) CD in Bookreader format.

This remainder of this section contains information that does not appear in the Compaq Fortran 77 documentation. Included in each description is a notation of the first version to which it applies.

3.1 *DEC Fortran Language Reference Manual*

3.1.1 Section 8.2.1 - Input Rules for FORMAT Statements

The FORTRAN–77 standard does not specify what the default BLANK value should be for preconnected units or internal files. Digital Fortran for OpenVMS (both VAX and Alpha) has chosen 'ZERO' in order to preserve compatibility with programs depending on the FORTRAN-66 behavior which was to always treat blanks as zeroes. However, the Fortran 90 standard explicitly specifies the default for internal files as 'NULL', which conflicts with the default used by Digital Fortran for OpenVMS (which conforms to FORTRAN–77). The Fortran 90 standard does not specify the default BLANK= behavior for preconnected units. The result is that programs which depend on a particular implementation's defaults for BLANK= may produce unexpected results on other platforms or when compiled using Fortran 90.

This issue can be especially troublesome when a program depends on short field termination (see section 8.8 of *DEC Fortran Language Reference Manual*) and is compiled using a Fortran 90 compiler which, by default, blank pads short records.

In order to avoid problems caused by inconsistencies in the default, Digital recommends adding an explicit BN or BZ format edit descriptor, as desired, in the format used for internal READ or DECODE statements. For preconnected units, BN or BZ should also be added, or these units can be explicitly opened so that the BLANK= behavior is consistent.

3.1.2 Section 11.2.6 - OPTIONS Directive

This directive is now supported on VAX systems. (V6.3-155)

In addition, a new /WARNINGS qualifier may be specified on the OPTIONS directive. See Section 2.2.1.3 for details. (V6.3-155)

3.1.3 Table C-7 - BLAS Basic Set Routines

For routines xROT and xROTG, the datatype for the first rotation element argument *c* is REAL*4 for CSROT and CROTG and REAL*8 for ZDROT and ZROTG. The datatype for the second rotation element argument *s* is the same type as the array argument.

3.2 *DEC Fortran User Manual for OpenVMS VAX Systems*

3.2.1 Section 1.2.3 - New qualifiers

New /ALIGNMENT (V6.3-155) and /PAD_SOURCE (V6.3-141) qualifiers have been added. See Section 2.2.1.1 and Section 2.5.1, respectively, for details.

3.2.2 Section 1.2.3.2 - New /ASSUME=SOURCE_INCLUDE Keyword

The /ASSUME qualifier now has a new SOURCE_INCLUDE keyword. See Section 2.3.1 for details. (V6.3-141)

3.2.3 Section 1.2.3.8 - /DEBUG Qualifier

The /DEBUG qualifier now accepts the PARAMETERS keyword, which may have values. See Section 2.6.6 for details. (V6.1-68)

3.2.4 Section 1.2.3.21 - /MATH_LIBRARY Qualifier

The description of the [NO]V5 keyword was omitted. See Section 2.8.1.4 for more details. (V6.0-1)

3.2.5 Section 1.2.3.27 - /STANDARD Qualifier

The keyword SOURCE_FORM now selects extension diagnostics for use of CDEC$ and CPAR$ directives. (V6.0-40)

3.2.6 Section 1.2.3.31 - /WARNINGS Qualifier

Replace the first paragraph with "Controls whether the compiler generates certain classes of optional diagnostic messages." as some of the diagnostics controlled by this qualifier have E-level severity.

Add the sentence "Specifying the qualifier /WARNINGS without any keywords is equivalent to specifying /WARNINGS=ALL; specifying /NOWARNINGS is equivalent to specifying /WARNINGS=NONE."

In the description of the ALIGNMENT keyword, delete the sentence "RECORD variables are not checked." (V6.3-155)

In the description of the DECLARATIONS keyword, omit the text "W-level". The diagnostic message for use of an untyped data item has E-level severity.

3.2.7 Section 1.x - INCLUDE Files

The manual omits an explicit description of how INCLUDE files are located. If the logical name FORT$INCLUDE is not defined, the default file specification for non-library INCLUDE files is '.FOR'; for library INCLUDE files it is '.TLB'. Section 2.8.3 describes the effect if the logical name FORT$INCLUDE is defined. (V6.0-1)

As of Version V6.2-130, if /ASSUME=SOURCE_INCLUDE is specified, portions of the INCLUDE file specification not supplied in the INCLUDE line or by the definition of FORT$INCLUDE, if any, are taken from the file specification of the current source file.

3.2.8 Section 1.3.4.1 - User Supplied Default Libraries

The statement that definition of the logical name FORT$LIBRARY as a search list is unsupported is no longer true. See Section 2.8.2 for more details. (V6.0-1)

3.2.9 Sections 8.1.1 and 8.1.2 - Sharing Images in Shareable Image Libraries

If an application is linked against a shareable image, such as a COMMON block to be used for sharing data, and that shareable image does not reside in the directory given by the logical name SYS$SHARE, you must define a logical name for the name of the shareable image which translates to the complete file specification. In the example given, the following definition would be needed:

```
$ DEFINE INC_COMMON DISK$USER:[INCOME.DEV]INC_COMMON.EXE
```

If the application is INSTALLed with any privilege, you must also specify the /EXEC and /SYSTEM qualifiers when issuing the DEFINE command.

3.2.10 Table G-1 - Source Program Diagnostic Messages

Table 3–1 describes new and changed source program diagnostic messages.

Table 3–1 Source Program Diagnostic Messages

Mnemonic	Error Code	Text/Explanation

ALNNOMAT	W	Alignment settings of common block are inconsistent with previous declaration
		Occurs when using CDEC$ OPTIONS/ALIGN (or /WARN=ALIGN) if the same COMMON is declared in multiple places within a single compilation unit and /ALIGN or /WARN=ALIGN has changed between places. For example:
		`CDEC$ OPTIONS /ALIGN=COMMON=NATURAL`
		`COMMON /COM1/ A,B`
		`! common elements are naturally aligned`
		`CDEC$ END OPTIONS`
		`CDEC$ OPTIONS /ALIGN=COMMON=PACKED`
		`COMMON /COM1/ C,D`
		`! common elements are packed`
		`CDEC$ END OPTIONS`
		`END`
		Note that COMMON declarations can also take place in SAVE and VOLATILE statements and in CDEC$ PSECT, CPAR$ PRIVATE and CPAR$ SHARED directives. All declarations must be in the scope of a consistent setting for alignment and alignment warnings.
		User Action: Modify the program to remove inconsistent alignment directives. (V6.3-154)

ARGLISEXE	F	IARGCOUNT/IARGPTR used in non-executable statement One of the argument list inquiry intrinsic functions, IARGCOUNT and IARGPTR, was used in a non-executable statement such as a statement function declaration. (V6.1)
ASFUNUSED	I	Statement function was defined but not used The specified statement function was defined but never used. This message can be suppressed with /WARNINGS=NOUNCALLED. (V6.1)
AUTDATINI	W	Variable is data-initialized; AUTOMATIC ignored A variable was declared as AUTOMATIC but was also data-initialized in a DATA or type specification statement. AUTOMATIC variables cannot be data-initialized. The AUTOMATIC attribute was ignored. (V6.0-16)

(continued on next page)

Table 3–1 (Cont.) Source Program Diagnostic Messages

Mnemonic	Error Code	Text/Explanation
AUTSAVALL	W	SAVE of all variables specified; AUTOMATIC ignored A variable was declared as AUTOMATIC in a program unit which contained a SAVE statement that, by omitting a list of names of variables to be SAVEd, specified that all variables should be SAVEd. AUTOMATIC variables cannot be SAVEd. The AUTOMATIC attribute was ignored. (V6.0-16)
AUTSHARED	W	SHARED or SHARED_ALL specified; AUTOMATIC ignored A variable was declared as AUTOMATIC was named in a CPAR$ SHARED directive or was in a program unit which contained a CPAR$ SHARED_ ALL directive. AUTOMATIC variables cannot be shared among parallel task processes. The AUTOMATIC attribute was ignored. (V6.0-16)
BRNCHINTOBLK	I	Questionable branch into loop or block A branch into a DO loop or IF block was detected. Although this might be valid if the FORTRAN II "extended range of a DO loop" feature was being used, it generally indicates a programming error. A common case involves two or more DO loops which share a common termination. In such cases, the shared termination statement is considered to belong to the innermost DO loop. This message can be suppressed with /WARNINGS=NOUSAGE. (V6.1)

(continued on next page)

Table 3–1 (Cont.) Source Program Diagnostic Messages

Mnemonic	Error Code	Text/Explanation

CHACONCONTD	I	Character, Hollerith or RAD50 constant continued across lines; may be non-portable
		
		A character, Hollerith or RAD50 constant was continued across two or more source lines. This is potentially non-portable as some implementations pad source lines with blanks and others, including Digital Fortran, do not, which can result in different constant values. This diagnostic is issued only if one or more of the source lines which made up the statement was shorter than the statement field width (72 or 132 columns, depending on /EXTEND_SOURCE), and can be suppressed with /WARNINGS=NOUSAGE.
		
		For character constants, use the concatenation operator to build up long values. Replace Hollerith or RAD50 constants with character constants if feasible or ensure that they are complete on one line. For character constants used for data initialization, PARAMETER constants must be used to maintain FORTRAN–77 standard conformance. For example:
		
		`CHARACTER*(*) CHRCON`
		`PARAMETER (CHRCON = 'This is a '//`
		`1 'continued character value')`
		`CHARACTER*80 CHRVAL`
		`DATA CHRVAL /CHRCON/`
		
		(V6.1-74)

COMALNERR	F	COMMON alignment error, too small for variable A field in the COMMON block is larger than the size of alignment requested by the CDEC$ PSECT directive. User Action: Specify a larger alignment value in the CDEC$ PSECT directive for the COMMON block. (V6.3-154)
ENDOPTMIS	W	No matching CDEC$ END OPTIONS for CDEC$ OPTIONS A CDEC$ OPTIONS directive must be terminated by CDEC$ END OPTIONS. User Action: Make sure that each CDEC$ OPTIONS directive is properly terminated by a CDEC$ END OPTIONS directive. (V6.3-154)
EQVSAVCOM	E	EQUIVALENCE may not be used to put a SAVE variable into COMMON An EQUIVALENCE group was found which included a COMMON variable and a variable named in a SAVE statement. SAVE variables may not be placed in COMMON, although an entire COMMON block may be named in a SAVE statement. (V6.0-5)

(continued on next page)

Table 3–1 (Cont.) Source Program Diagnostic Messages

Mnemonic	Error Code	Text/Explanation
EXT_ASFARGNAM	I	Extension to FORTRAN–77: Statement function argument name same as non-variable A statement function dummy argument had the same name as an entity other than a variable or a common block (for example, a PARAMETER constant). (V6.1)
EXT_INVINTARG	I	Extension to FORTRAN–77: Nonstandard use of intrinsic function as actual argument The FORTRAN–77 standard does not permit the use of the type conversion (INT, DBLE, etc.), lexical relationship (LGE, LGT, etc.) or minimum and maximum functions (MIN, MAX, etc.) as actual arguments. (V6.1)
EXT_KEYCON	I	Extension to FORTRAN–77: nonstandard combination of keywords/values A statement included a nonstandard combination of keywords or keyword values, for example: • In an OPEN statement, FILE= specified when STATUS='SCRATCH' (V6.4)
FEAUNSAXP	W	This feature is unsupported on Alpha systems Detected a language feature supported on this platform that is not supported on Alpha systems. (Text correction)
FLDMISALN	W	Record contains one or more misaligned fields One or more fields are not naturally aligned in a RECORD structure. If the record is or contains a record array, one or more of the array elements may not be naturally aligned. Specifying /NOALIGN or /ALIGN=RECORDS=PACKED, the default, causes Digital Fortran to pack the fields within records instead of naturally aligning them. User Action: Consider specifying the /ALIGN qualifier or rearrange fields so that they fall on natural boundaries. (V6.3-154)
FUNVALUND	W	Function value undefined at end of routine A function did not have its return value defined at the end of the routine. (V6.0-24)
INCNOTSUP	F	INCLUDE not supported for current source file device An INCLUDE statement was found while the current source device was not random-access, for example a tape drive or a terminal. The compiler requires that it be able to close and later re-open and re-position the source file before processing an INCLUDE statement. (V6.0-1)

(continued on next page)

Table 3–1 (Cont.) Source Program Diagnostic Messages

Mnemonic	Error Code	Text/Explanation
INTACTARG	W	Intrinsic routine used as actual argument should be named in INTRINSIC statement An identifier which had been previously used as an intrinsic routine was used as an actual argument, but was not named in an INTRINSIC statement. The compiler assumed that the intrinsic routine of that name was intended. If the identifier is intended to be a routine name, declare it in an EXTERNAL or INTRINSIC statement as appropriate. This message can be suppressed with /WARNINGS=NOUSAGE. (V6.1)
OPTMISSING	W	No matching CDEC$ OPTIONS for CDEC$ END OPTIONS A CDEC$ END OPTIONS directive terminates a CDEC$ OPTIONS section. User Action: Remove extraneous CDEC$ END OPTIONS directives or add CDEC$ OPTIONS directives as appropriate. (V6.3-154)
ROUREFREC	F	Routine referenced recursively; /RECURSIVE required A subroutine, function or entry name was referenced recursively in the same program unit, but the /RECURSIVE command or OPTIONS statement qualifier was not specified. (V6.0-22)
STREMPTY	I	Structure is empty A STRUCTURE declaration contains no fields. (V6.4)
TOOMNYOPT	W	CDEC$ OPTIONS directives nested too deeply - this one ignored CDEC$ OPTIONS directives cannot be nested beyond 100 levels. User Action: Modify source so that CDEC$ OPTIONS nesting depth does not exceed 100. (V6.3-154)
USEBEFDEF	I	Use of variable before definition; name in SAVE statement if appropriate A variable in a subprogram was used before its value was defined. This may have been intentional, with an assumption of implicit SAVE semantics, but may also have been a programming error. This message can be suppressed with /WARNINGS=NOUNINITIALIZED. If intentional, name the variable in a SAVE statement and make sure that it is properly initialized. Initialization to zero is not guaranteed on all implementations. (V6.1-78)

(continued on next page)

Table 3–1 (Cont.) Source Program Diagnostic Messages

Mnemonic	Error Code	Text/Explanation
VARUNUSED	I	Variable was declared but not used The specified variable was declared but never used. This message can be suppressed with /WARNINGS=NOUNUSED. (V6.1)
WRONGCLD	F	Wrong command definition installed The current DCL command tables are incompatible with the compiler version. This may occur if an old copy of the command tables are being used, or if a new version of the compiler was installed and a user process is still using an old version. See the *Digital Fortran Installation Guide for OpenVMS VAX Systems* for instructions on updating the command tables. (V6.0-1)

3.2.11 Table G-4 - CRX Error Messages

Table 3–2 describes new and changed CRX diagnostic messages which are issued during processing of DICTIONARY statements.

Table 3–2 CRX Messages

Mnemonic	Error Code	Text/Explanation
BADRGTJUST	E	Non-text field description specifies right justification This message is no longer issued; the compiler ignores the justification attribute. (V6.0-9)

In addition, the messages NOPPARVEC, OPTLV4CHK, PARCHKCON, RECPARCON and VECCHKCON, which relate to conflicts between certain compile command or OPTIONS statement qualifiers, now have severity W instead of E.

A

Compaq Fortran Maintenance Changes

Table A–1 contains a short descriptive summary of each change made for this update of the Compaq Fortran 77 product. The edit level number, which appears at the end of the compiler version number, is incremented by one for each change. The version number is printed at the top of each page of source listing generated by the compiler. The edit level appearing to the left of each description was the level number after that change was made. Thus, any compiler versions with level numbers equal to or greater than the one given for a certain change, contain that change.

Table A–1 Compaq Fortran Maintenance Changes

Version	Description
V6.0-1	DEC Fortran V6.0 Release; contains all corrections to VAX FORTRAN through version V5.9-175 and to VAX FORTRAN-HPO through version V1.3-252
V6.0-2	The compiler now detects as an extension to FORTRAN–77 the use of a non-integer expression as a logical unit number in an I/O statement.
V6.0-3	A number of problems related to the "non-native data in I/O" feature were corrected, including: • The FOR$CONVERTnnn logical names were not properly recognized. • Big-endian IEEE and all big-endian integer conversion did not work correctly. • Conversion in I/O statements with implied DO-loops did not always work correctly. • Conversion of COMPLEX variables did not work. • Applications which executed an ENDFILE or FIND statement on a unit which was not open could get an INVARGFOR error at run-time.
V6.0-4	The compiler no longer goes into a loop if a statement function has two or more formals which are declared as RECORDs using the same STRUCTURE.
V6.0-5	The compiler now issues the new diagnostic EQVSAVCOM if EQUIVALENCE is used to attempt to put a SAVE variable into COMMON.

(continued on next page)

Table A–1 (Cont.) Compaq Fortran Maintenance Changes

Version	Description
V6.0-6	The DEC Fortran kit now provides an updated copy of the VMSRTL.EXE shareable image, which is required by images linked against VAX/VMS versions earlier than V4.0. Without the updated VMSRTL.EXE, images which use it may fail unpredictably due to registers not being properly saved across calls to Fortran or Math Run-Time Library routines. The new VMSRTL.EXE specifies that all registers are to be saved so that changes to register usage in routines now in the BASRTL, COBRTL, FORRTL, LIBRTL and MTHRTL shareable images will not break images linked against VMSRTL. Images linked on VAX/VMS V4.0 or later do not use VMSRTL and are not affected. The updated VMSRTL.EXE is not provided on OpenVMS VAX Version V6.0 or later as it is not needed there.
V6.0-7	A problem in which the compiler would fail with a bugcheck in the PACK phase was corrected.
V6.0-8	The Digital-provided INCLUDE files SYS$LIBRARY:FORDEF.FOR, SYS$LIBRARY:FORIOSDEF.FOR and the $FORMSG module of SYS$LIBRARY:FORSYSDEF.TLB have been updated to include the new FOR$_FLOCONFAI status value and associated IOSTAT number definitions.
V6.0-9	The compiler no longer issues the CRX error message BADRGTJUST for a CDD non-text field that specifies right-justification. The justification attribute is ignored by DEC Fortran.
V6.0-10	The correct line number is now displayed for the EXTCHASOU and TOOMANCON compiler diagnostics.
V6.0-11	The compiler no longer gives spurious USEUNIVAR warnings for references to COMPLEX variables which were data-initialized using character constants, and now only gives one warning per COMPLEX variable instead of separate warnings for the real and imaginary parts.
V6.0-12	The compiler now properly displays the attempted file specification when it gives an INCOPEFAI error on failing to open an INCLUDE library.
V6.0-13	The compiler now allows record variables to appear in AUTOMATIC, SAVE and STATIC statements.
V6.0-14	The compiler was changed to emit the quadword stack alignment routine entry code sequence (see Section 2.8.9) only when the /VECTOR compile command qualifier is specified.
V6.0-15	The compiler no longer generates bad code when a LOGICAL*2 or LOGICAL*4 array element is used in an arithmetic-IF statement.
V6.0-16	A nested STRUCTURE with a data-initialized field in in a program unit compiled with /RECURSIVE no longer causes the linker to issue an UDEFPSC error. In addition, the compiler now issues the warnings AUTDATINI and AUTSAVALL if a variable declared AUTOMATIC was data-initialized or appears in a program unit which specifies that all variables are to be SAVEd.
V6.0-17	The compiler no longer generates incorrect code when the only store to a formal argument occurs by its reference in an IOSTAT specification. In some cases, the modified value was not written back through the argument list.
V6.0-18	The compiler no longer causes the linker to issue a UDEFPSC error for program units which reference a local variable in a READ or ACCEPT statement but don't pass that variable as an actual argument.
V6.0-19	The compiler now properly passes a copy of an actual argument when that argument is an expression consisting of a reference to an intrinsic function that returns its argument unchanged, such as INT of an INTEGER argument.

(continued on next page)

Table A–1 (Cont.) Compaq Fortran Maintenance Changes

Version	Description
V6.0-20	The Run-Time Library no longer reports an INVARGFOR error when the PDP–11 compatibility routines ERRSET, ERRSNS or ERRTST are used for error number 95, FLOCONFAI.
V6.0-21	The compiler no longer fails with a bugcheck in the CSE phase when /VECTOR is specified and the program contains certain uses of COMPLEX intrinsics.
V6.0-22	The compiler now allows recursive function references as well as recursive references to ENTRY points which are defined after the function reference or CALL statement. The new error message ROUREFREC, "Routine referenced recursively; /RECURSIVE required" is given if the compiler detects a recursive reference and /RECURSIVE was not specified. The compiler now produces correct cross-reference and SCA information for recursive function references.
V6.0-23	The compiler no longer generates incorrect code in a situation whose symptom was that register R0 was believed to still hold a value that had been loaded prior to a subroutine call which modified R0.
V6.0-24	The compiler no longer issues the EXCCHASOU, "Extra characters in source" warning, enabled by /WARNINGS=TRUNCATED_SOURCE, for source lines containing whole-line comments or only blank characters, even if those lines exceed the statement field width. The compiler issues the new warning message FUNVALUND, "Function value undefined at end of routine" if a function return value is not defined at one or more exit points of the function. The standards extension diagnostic EXT_UNDFUNC is no longer issued, being replaced by FUNVALUND. The compiler now more accurately displays statement line numbers associated with error messages and the /SHOW=LOOP_STRUCTURE display.
V6.0-25	The compiler no longer generates incorrect code for certain unusual cases involving a routine with a character formal argument whose address was taken with %LOC and another argument was used as a KEY value in an indexed READ statement. The symptom was that one argument's address was loaded into register R0 or R1 for later use, but the copy of the character argument's descriptor overwrote those registers.
V6.0-26	The compiler's limit for the amount of internal representation allowed per statement has been doubled, making it much less likely than the STATOOCOM, "Statement too complex" error will be generated.
V6.0-27	The compiler no longer fails with a bugcheck during LEXSYN when a type declaration statement has an end-of line comment, is followed by a blank line, and /ANALYSIS_ DATA is specified.
V6.0-28	The compiler now generates correct code for the case where a subprogram has one or more ENTRY statements, a non-CHARACTER dummy argument appears in two or more of the dummy argument lists, and that argument is passed as an actual argument to another routine using %DESCR.
V6.0-29	The compiler no longer fails with a bugcheck during PACK in certain cases where the double-precision MOD intrinsic is used in conjunction with array references.
V6.0-30	Superseded by edit V6.0-51.
V6.0-31	The compiler no longer fails with a bugcheck during LOCAL when the %LOC operator is used on an external procedure which has been typed as COMPLEX or COMPLEX*16.
V6.0-32	The compiler no longer fails with a bugcheck during PACK when a record field that is longer than 65535 bytes is assigned to another record field.
V6.0-33	The compiler now generates correct Debug Symbol Table information for program units which contain Variable Format Expressions. The incorrect information only affected the DEC Performance and Coverage Analyzer's COVERAGE BY LINE feature, where it might cause the program to fail in unpredictable ways while run under PCA.

(continued on next page)

Table A–1 (Cont.) Compaq Fortran Maintenance Changes

Version	Description
V6.0-34	The compiler no longer fails with a bugcheck during LIFE in certain cases involving the ISIGN intrinsic called with expression arguments.
V6.0-35	The compiler now properly handles multiple translations of the FORT$LIBRARY logical name in all cases.
V6.0-36	When FORTRAN/DML is used, the compiler no longer improperly ignores DICTIONARY statements which are followed by an INCLUDE statement or an FDML preprocessor statement.
V6.0-37	The compiler no longer fails with a bugcheck during LOCAL for certain cases involving mixed complex-real arithmetic operations.
V6.0-38	The compiler no longer fails with a bugcheck during FINAL nor generates an incorrect object module if a program unit defines FOR$_ASSERTFAIL as a PARAMETER constant (for example, by including module $FORDEF from FORSYSDEF.TLB) and contains an ASSERT statement.
V6.0-39	The math library routine which implements the D_floating version of the ATAN2D intrinsic no longer causes unexpected program behavior (such as a program exiting with a status of 43B4) when the first argument is positive and the second argument is zero. This problem would only be seen by programs which were compiled using DEC Fortran V6.
V6.0-40	The compiler now issues warnings about CDEC$ and CPAR$ directives being extensions to Fortran-77 only if /STANDARD=SOURCE_FORM is in effect. Previously, the compiler would diagnose these directives, which are special forms of comments, if /STANDARD=SYNTAX was in effect.
V6.0-41	The compiler no longer generates incorrect code for certain program units which contain uses of the character concatenation operator.
V6.0-42	The compiler no longer generates incorrect code which does not account for a COMMON variable changing across a routine call when the /ASSUME=DUMMY_ALIASES compile command qualifier is in effect.
V6.0-43	The compiler no longer issues the FUNVALUND warning for a CHARACTER function whose only definition of the function return value is by means of passing the return value variable as an actual argument in a routine call.
V6.0-44	The compiler no longer gives unexpected warnings about features not implemented on ULTRIX when /WARNINGS=Alpha_AXP is specified.
V6.0-45	The compiler no longer generates incorrect code for the following combination of occurrences: • A triply-nested DO-loop • The end expression for the innermost loop is of the form A(I)-1 where I is the loop control variable of the outermost loop. • There are no other uses of A(I) inside the loops.
V6.0-46	The compiler no longer generates incorrect code involving improper reuse of registers following a character operation.
V6.0-47	The compiler no longer generates incorrect code which substitutes a multiply by -2**31 with an arithmetic shift.
V6.0-48	For cases where a reference to a BLAS1 extended routine was inlined, the compiler now produces a correct object module that does not cause the linker to issue an informational diagnostic about an undefined symbol.

(continued on next page)

Table A–1 (Cont.) Compaq Fortran Maintenance Changes

Version	Description
V6.0-49	The compiler no longer takes an excessive amount of time to compile a program unit containing one or more large arrays which are DATA-initialized.
V6.0-50	The compiler no longer applies the split-lifetime optimization to formal arguments and variables in COMMON if a direct-access I/O statement is seen, due to the possibility of their having been named as an ASSOCIATEVARIABLE in another program unit.
V6.0-51	The compiler no longer consumes excessive memory and CPU resources when compiling a program unit which contains complicated expressions of COMPLEX values.
V6.0-52	The compiler no longer issues the FUNVALUND warning if a function contains ENTRY statements and at least one of the function return variables was defined at all exit points.
V6.0-53	The compiler no longer allows a parenthesized variable as an internal file unit specifier, as it does not qualify as a "character scalar memory reference".
V6.0-54	The compiler no longer fails with a bugcheck during CODE for certain large program units which use the character concatenation operator.
V6.0-55	The compiler no longer bugchecks when a statement function formal argument has the same name as a PARAMETER constant declared with CHARACTER type. According to the Fortran-77 standard, a statement function formal may not share a name with an entity other than a variable or a COMMON block. See also V6.0-62.
V6.0-56	The compiler no longer generates incorrect code for routines which contain direct I/O statements and for which an associated variable was passed as an argument to that routine.
V6.0-57	The compiler no longer generates invalid code for certain program units which have a series of redundant assignments to a variable and a conditional assignment to that variable before the final redundant assignment.
V6.0-58	The compiler no longer fails with a bugcheck during LIFE for certain program units which contain redundant assignments of expressions to a variable and where that expression is also used later in the program unit.
V6.0-59	The compiler no longer generates incorrect code whose effect was to overwrite the imaginary part of a COMPLEX value with the real part, typically before use as an argument to an intrinsic function.
V6.0-60	The compiler no longer generates incorrect descriptor initialization code for COMPLEX array arguments to routines with multiple entry points.
V6.0-61	The compiler will issue an INTACTARG warning if an identifier which has been used as an intrinsic routine, without having been named in an INTRINSIC statement, is passed as an actual argument. This usually indicates a programming error; if it is intended that an intrinsic routine be passed, name it in an INTRINSIC statement. Also, the compiler will now issue a standards extension diagnostic EXT_INVINTARG if one of the following classes of intrinsic functions is passed as an actual argument: type conversion, lexical relationship and minimum or maximum.
V6.0-62	The compiler will now issue the standards extension diagnostic EXT_ASFARGNAM if the name of a statement function dummy argument is the same as an entity other than a variable or a common block, such as a PARAMETER constant. If the name is shared with a FORTRAN–77 style (typed) PARAMETER constant, the declared type is used, otherwise the argument type is implicit based on the first character of the name.
V6.0-63	The compiler no longer generates incorrect code for certain calls to the ATAN2, ATAN2D, SIGN (real) and MOD (real) functions when the second argument is an array reference.
V6.0-64	The compiler no longer produces an invalid SCA analysis data file which causes SCA to report that the module has an "old ANA format" if a program unit contains the CDEC$ IDENT directive and the /DESIGN=COMMENTS qualifier is in effect.

(continued on next page)

Table A–1 (Cont.) Compaq Fortran Maintenance Changes

Version	Description
V6.0-65	The compiler no longer generates incorrect register usage code for programs which use the SIGN intrinsic with REAL*4 or REAL*8 operands. The problem was most apparent with REAL*8 in that the generated code could use registers R4-R6, but the compiler might not cause those registers to be saved in the routine entry mask or might mistakenly allocate other values to those registers.
V6.0-66	In a case where a vectorized loop contained a call to a mathematical intrinsic function, such as LOG, using a scalar argument that was an induction variable, and where the call to the intrinsic conditional, the compiler now correctly merges in "safe values", those for which the intrinsic will not report an error, for the appropriate elements in the vector register argument where the conditional test would, in the scalar case, have prevented the intrinsic function from being called.
V6.0-67	The compiler no longer fails with a bugcheck in the LEXSYN phase if a BLOCK DATA subprogram contains a variable which is named in an EQUIVALENCE statement but not in a COMMON statement and the /ANALYSIS_DATA qualifier is used. Since such variables are of no use, BLOCK DATA subprograms should contain only COMMON variables, the compiler now also issues an "unused variable" diagnostic if it finds non-COMMON variables in a BLOCK DATA subprogram.
V6.1-68	The compiler no longer creates an incorrect layout for CDD structure arrays which have an alignment attribute. In this case, the compiler now declares a fill field of the proper total array size and issues the CDDALNARY diagnostic.
V6.1-69	The compiler no longer fails with a bugcheck during LEXSYN if an attempt is made to declare an array with 10 or more dimensions. The maximum number of dimensions allowed for an array is 7.
V6.1-70	The compiler no longer generates incorrect code for certain cases where a COMPLEX variable was named in an I/O statement or passed as an actual argument and subsequently only the imaginary part was used.
V6.1-71	The compiler no longer generates incorrect vector code for certain DO loops which have non-constant increment expressions. Previously, the compiler could incorrectly calculate the necessary number of iterations.
V6.1-72	The compiler now diagnoses as an extension to the FORTRAN–77 standard the omission of a delimiter in a FORMAT between a scale factor and a repeated edit descriptor, for example: 1P3E12.6. The status of this usage has been uncertain in the past, but X3J3, the ANSI FORTRAN Standards Committee, has approved as a draft interpretation Defect Item 50, stating that omission of the comma is indeed nonstandard for FORTRAN–77.
V6.1-73	The compiler no longer generates incorrect code for certain programs that have array references in loops. The symptom was that a register was used for an array index that was loaded earlier in the program, but the compiler was not taking into account the fact that the register was modified by an operation later in the loop.
V6.1-74	The compiler was enhanced to detect character, Hollerith or RAD50 constants which were continued across source lines; a potentially non-portable usage.
V6.1-75	The compiler no longer issues a FUNVALUND diagnostic for function return values which become defined by their appearance in an input statement.
V6.1-76	The compiler now issues the warning ASSDOVAR if a function return value is used as a DO-loop control variable and is assigned to inside the loop. Previously, the compiler issued this warning for other types of variables but not function return values.
V6.1-77	The compiler no longer reports that a variable is unused if its only use is in a NAMELIST group.

(continued on next page)

Table A–1 (Cont.) Compaq Fortran Maintenance Changes

Version	Description
V6.1-78	The compiler's detection of uninitialized variables was improved and expanded to include a new informational diagnostic USEBEFDEF if a variable is used before it is defined in a subroutine or function, suggesting a reliance on implicit SAVE semantics but also possibly indicating a programming error.
V6.1-79	The compiler now writes correct information to the debug symbol table about the start of a routine and the routine's prolog code. This corrects a problem where if a SET MODULE was not done in the debugger that EXAMINE/SOURCE of the routine name did not work. Also, if the module is SET, a routine breakpoint will now stop after all of the routine initialization prolog code instead of at the first instruction, thus allowing arguments to be examined without having to step through the prolog code.
V6.1-80	The /WARNINGS=TRUNCATED_SOURCE feature was restored after being inadvertently disabled for V6.1.
V6.1-81	The compiler no longer issues an inappropriate VARUNUSED diagnostic for a variable whose only use is as the loop control variable in an implied-DO loop in a DATA statement (or data-initialization clause).
V6.1-82	The compiler no longer fails with a bugcheck during the LOCAL phase for certain programs which declare a structure field which is a record array.
V6.1-83	The compiler now generates correct code when a pointee array is passed as an actual argument.
V6.1-84	The compiler no longer fails with a bugcheck during the LEXSYN phase if a structure field array is declared using an invalid dimension expression (eg. a CHARACTER or STRUCTURE name) and the field array is subsequently referenced.
V6.1-85	The compiler no longer fails with a bugcheck during the CODE phase if a REAL*16 or COMPLEX*16 function is called recursively.
V6.1-86	The compiler no longer fails with a bugcheck during the CODE phase if a pointee array is accessed in a program unit compiled with /CHECK=BOUNDS enabled.
V6.1-87	The compiler now silently ignores the CDEC$ OPTIONS and CDEC$ END OPTIONS directives which are recognized by DEC Fortran on other platforms.
V6.1-88	The compiler no longer fails with a bugcheck during the SPLIT phase for certain programs compiled with /OPTIMIZE=LEVEL=4.
V6.1-89	The compiler no longer issues erroneous VARUNUSED warnings when structure declarations appear in a BLOCK DATA subprogram.
V6.1-90	The compiler now suppresses VARUNUSED diagnostics for variables generated by the FORTRAN/DML preprocessor when an INVOKE statement is used.
V6.1-91	The compiler no longer generates incorrect code for certain unusual programs which contain nested loops performing an array copy operation in row-major order, a leftmost array bound of 1 and a particular combination of datatype and loop length.
V6.1-92	The compiler no longer fails with a bugcheck during SELECT for programs which erroneously contain an IMPLICIT NONE statement after a statement which appears to be an array declaration with 8 or more dimensions.
V6.1-93	The compiler now allows any 8-bit character other than codes X'00' through X'04' to appear in a source line. Previously, any non-printing character would result in an INVCHASOU warning and the offending character would be replaced by a blank. This brings the behavior of the VAX compiler more in line with that of other DEC Fortran implementations.

(continued on next page)

Table A–1 (Cont.) Compaq Fortran Maintenance Changes

Version	Description
V6.1-94	The compiler now treats a reference to an external routine in a USEROPEN specification as a call reference for the purposes of the SCA analysis data file. This allows SCA to find the routine as being (implicitly) called from the OPEN statement. Previously the reference was treated as an address reference.
V6.1-95	The compiler no longer improperly processes a statement label when a source line has a redundant continuation mark.
V6.1-96	Error messages relating to problems with source lines, for example REDCONMAR or INVSTALAB are now displayed in the listing file after the appropriate source line rather than before, have the correct line number indicated, and have more accurate LSE diagnostics emitted.
V6.1-97	The compiler's use of virtual memory for large compilations was reduced somewhat.
V6.1-98	The compiler no longer issues a spurious VARUNUSED diagnostic for a variable used as the loop control variable in an implied-DO-in-DATA in a BLOCK DATA subprogram.
V6.1-99	The compiler now emits correct SCA analysis data information for the use of a BLAS1-Extended routine which was not named in a type declaration statement.
V6.1-100	The compiler now accepts the '#' character in column 1 as a whole-line comment delimiter. This is intended to allow the use of C preprocessors with Fortran source and use of '#' as a general comment delimiter is not recommended.
V6.1-101	The compiler no longer generates incorrect code for certain pointee structure field references.
V6.1-102	The compiler no longer generates incorrect code for certain COMPLEX statement functions whose function value real part contains a dual negation.
V6.1-103	The compiler now correctly recognizes the IARGCOUNT, IARGPTR and LOC intrinsics when /MATH_LIBRARY=V5 is specified.
V6.1-104	The compiler no longer fails with a bugcheck during CREATE_FLOW for certain programs when compiled with /VECTOR.
V6.1-105	FORSYSDEF.TLB, when rebuilt with this version or later of DEC Fortran, defines constants (other than masks) which have negative values using decimal notation rather than hexadecimal. This allows such values to be assigned to INTEGER*2 variables and record fields without causing the compiler to issue an EXCDIGTRU diagnostic.
V6.1-106	The compiler no longer generates incorrect code which causes a register to be overwritten before its last use in certain uses of array elements in complex arithmetic.
V6.1-107	The compiler no longer generates incorrect code for certain cases involving the loading of arguments to complex intrinsics.
V6.2-108	A problem introduced in V6.1-97, which could cause the compiler to bugcheck or loop, was corrected.
V6.2-109	The compiler no longer issues spurious VARUNUSED diagnostics for a POINTER variable which is an actual argument or in COMMON if one of its pointees is used.
V6.2-110	The compiler's checking for uninitialized variables has been improved to include variables which have not been selected for full analysis (a maximum of 128 variables are selected) but which have no stores in the program unit.
V6.2-111	The compiler no longer allocates an unnecessary longword of storage when a record array declaration is seen.
V6.2-112	The compiler no longer improperly processes constants of the form B'xxx', O'xxx', X'xxx' and Z'xxx' when the constant string includes embedded blanks or tabs (for example: X'123 45'). Also, the compiler no longer issues a CHACONCONTD diagnostic if a constant of the form 'xxx'X, etc. is continued across multiple source lines.

(continued on next page)

Table A–1 (Cont.) Compaq Fortran Maintenance Changes

Version	Description
V6.2-113	The compiler no longer fails with a bugcheck during the LEXSYN phase when a logical-IF contains a statement which would, if it appeared on its own, cause an "unrecognized statement" error.
V6.2-114	The compiler no longer fails with a bugcheck during the PACK phase when the following combination occurs: 1) Use of the SIN, COS, TAN, ATAN, LOG, LOG10, EXP or SQRT intrinsics, 2) The argument to the intrinsic is of a COMPLEX type, 3) The compile command qualifiers /VECTOR and /MATH_LIBRARY=FAST were both specified. Note that the /MATH_LIBRARY=FAST qualifier is documented to have an effect for real datatypes only.
V6.2-115	The compiler no longer generates incorrect SCA analysis data information for a main program which begins with a DATA, DEFINE FILE, FORMAT or NAMELIST statement. In addition, the compiler now generates correct SCA typing information for variables whose first appearance is in a NAMELIST statement which occurs in the executable part of a program.
V6.2-116	The compiler no longer fails with a bugcheck during the LEXSYN phase under extremely rare and unpredictable circumstances while processing an actual argument list.
V6.2-117	The compiler no longer fails with a bugcheck during the LIFE phase for certain programs which have an array reference as a DO-loop bound.
V6.2-118	The compiler no longer emits bounds checking code for expanded BLAS routines, even if /CHECK=BOUNDS is in effect. The bounds checking code interfered with the common use of passing multiple-dimension arrays to certain BLAS routines (for example, DSET).
V6.2-119	The compiler no longer generates incorrect code for loading the argument to certain COMPLEX*16 intrinsic functions such that the address of the argument was overwritten by the real part of the argument.
V6.2-120	The compiler now correctly flags as an extension the use of the INTEGER*1, LOGICAL*2 and REAL*16 types in an INTRINSIC statement when /STANDARD=MIA is in effect.
V6.2-121	The compiler no longer fails with a bugcheck during the FINAL phase when a %LOC function or LOC intrinsic makes a recursive reference to the enclosing subroutine or function when the generated instruction for that reference is more than 128 bytes away from the routine entry point. Also the compiler now correctly issues the ROUREFREC diagnostic if such recursive reference occurs but the /RECURSIVE qualifier was not specified.
V6.2-122	The compiler no longer issues a USEUNIVAR for the use of a character variable which was partially defined by having had a substring passed as an actual argument.
V6.2-123	The compiler no longer disallows the lowercase letters 'a' through 'f' in prefix-style hexadecimal constants (eg. X'123abc').
V6.2-124	The compiler now gives the ASSDOVAR warning if a DO-loop control variable appears as an IOSTAT specifier in an I/O statement inside a DO loop.
V6.2-125	The compiler now properly sign-extends bit constants when used in contexts where they are treated as INTEGER*2 or BYTE types. For further details, see Section 2.3.5.
V6.2-126	The compiler no longer gives an INVLEXEME error when a record field array is used as the buffer in an ENCODE or DECODE statement.
V6.2-127	The compiler no longer fails with a bugcheck during the CODE phase for certain programs which use inlined BLAS intrinsics and are compiled /CHECK=BOUNDS.
V6.2-128	The compiler now emits the correct DEBUG symbol table information for untyped (non-Fortran-77) PARAMETER constants whose value is a bit constant longer than 32 bits so that these values may be evaluated during a DEBUG session.

(continued on next page)

Table A–1 (Cont.) Compaq Fortran Maintenance Changes

Version	Description
V6.2-129	The compiler now supports /DEBUG=PARAMETERS=NONE. If this is specified, no PARAMETER constants are written to the debug symbol table. Note that /DEBUG=NOSYMBOLS prevents any debug symbol information, including PARAMETER constants, from being written.
V6.2-130	The compiler now supports the command line keywords /WARNINGS=[NO]INFORMATIONAL and /ASSUME=[NO]SOURCE_INCLUDE.
V6.2-131	The compiler no longer fails with a bugcheck during the CODE phase when a pointee non-adjustable array is passed by descriptor.
V6.2-132	The compiler no longer issues a VARUNUSED diagnostic for pointees.
V6.2-133	The compiler no longer corrupts its internal memory structures for certain programs which contain COMPLEX variables. While in most cases this corruption seemed to be harmless, it has been known to cause the compiler to fail with an access violation during the SPLIT phase. Other symptoms are possible.
V6.2-134	The compiler no longer generates incorrect code which would return an improper result for %LOC of a pointee array.
V6.2-135	The compiler no longer fails with a bugcheck during the SELECT phase if a program contains more than 252 COMMON blocks and EQUIVALENCEs variables in some of those COMMON blocks.
V6.2-136	The compiler no longer generates incorrect code in some cases when an actual argument is used as an argument to a two-operand intrinsic routine such as SIGN.
V6.2-137	The compiler no longer gives a spurious FMTINVCHA error when a FORMAT statement has interspersed comment lines and the /DESIGN=COMMENTS compile command qualifier is in effect.
V6.2-138	The compiler no longer fails with a bugcheck during the LOCAL phase when an expression involving a mixture of a call to an external function declared REAL and a COMPLEX expression is the argument to certain intrinsic functions.
V6.2-139	The compiler no longer fails with a bugcheck during ALLOCATE when a RECORD statement declares a pointee as a record with a structure which contains data initialization. The new behavior is to ignore the data initialization for that record variable as is already done if the variable is a formal argument.
V6.2-140	The compiler no longer fails with a bugcheck during the PACK phase when an expression compares two pointee character record fields which are not at the base of the record structure.
V6.3-141	The compiler no longer fails with a bugcheck during the LIFE phase when an ENTRY statement includes in its formal argument list a name which has been previously declared as a pointee. This usage is invalid; the compiler will now issue a MULDECNAM error.
V6.3-142	The compiler no longer fails with a bugcheck during the PACK phase for certain programs with expressions containing character concatenation, substrings and the use of the ISHFT intrinsic function.
V6.3-143	The compiler no longer fails with a bugcheck during the LIFE phase when the argument to SIZEOF is a function reference.
V6.3-144	The compiler now checks local variables for misalignment due to EQUIVALENCE when /WARNINGS=ALIGNMENT is specified.
V6.3-145	The compiler no longer fails with a bugcheck during the SELECT phase for certain main programs containing an adjustable array declaration which is followed by an IMPLICIT NONE statement.

(continued on next page)

Table A–1 (Cont.) Compaq Fortran Maintenance Changes

Version	Description
V6.3-146	The compiler no longer generates bad code for certain programs which pass pointee record fields as actual arguments.
V6.3-147	The compiler no longer fails with a bugcheck during the SYNC phase when /OPTIMIZE=LEVEL=4 is specified for certain programs containing a comparison between a type conversion from a non-integer or logical type (eg. ICHAR) and .TRUE. or .FALSE..
V6.3-148	The compiler no longer generates incorrect DEC Source Code Analyzer information for certain references to BLAS intrinsics when the /ANALYSIS_DATA qualifier is specified.
V6.3-149	The compiler no longer fails with a bugcheck during the LEXSYN phase in rare circumstances for certain programs using the DICTIONARY statement. (This problem was reported against DEC Fortran for OpenVMS Alpha; the VAX compiler shares the problem even though it has never been reported on a VAX.)
V6.3-150	The compiler now correctly disallows in all cases the use of the %REF, %VAL and %DESCR built-in functions outside the context of an actual argument.
V6.3-151	The compiler no longer generates incorrect DEC Source Code Analyzer information for certain programs containing placeholders in declarations when the /DESIGN=PLACEHOLDERS and /ANALYSIS_DATA qualifiers are specified.
V6.3-152	The compiler no longer generates bad code or spurious USEUNIVAR warnings in unusual circumstances for certain programs in which a variable becomes defined by its appearance in the I/O list of a READ statement and where just prior to the read a subroutine was called with a record field as an actual argument.
V6.3-153	The compiler no longer generates bad code, where an array index register would get overwritten, for certain programs containing a use of the MAX or MIN intrinsic functions with array element arguments as an actual argument.
V6.3-154	The compiler's listing of PSECT attributes now uses the keywords OCTA and PAGE for alignments of 2**4 and 2**9.
V6.3-155	The compiler now emits Debug Symbol Table information for pointees.
V6.3-156	The compiler no longer emits incorrect code for certain cases of a COMPLEX*16 intrinsic function reference.
V6.3-157	The compiler now permits a RECORD variable whose corresponding STRUCTURE is DATA-initialized to be used as a pointee in a POINTER statement. The data-initialization attribute is ignored, as it is already if a DATA-initialized RECORD is used as a dummy argument. This allows the same STRUCTURE to be used for a pointee and a local variable.
V6.3-158	The compiler no longer emits initialization code to load the vector alignment fixup handler when /CHECK=ALIGN is specified for a BLOCK DATA subprogram.
V6.3-159	The compiler no longer fails with a bugcheck in the LOCAL phase for certain programs containing a reference to a COMPLEX array where the array index expression contains a function call and /CHECK=BOUNDS is specified.
V6.3-160	The compiler no longer fails with a bugcheck in the SPLIT phase for certain programs which use pointee arrays.
V6.3-161	If a the first executable statement is followed by a blank line or comment, the compiler now correctly emits DEBUG information indicating that the executable code begins at the first executable statement rather than the second.
V6.3-162	The compiler no longer passes the wrong address for an integer variable passed by descriptor to a subroutine when that variable is incremented in a loop in the called routine and /OPTIMIZE=LEVEL=4 is specified.

(continued on next page)

Table A–1 (Cont.) Compaq Fortran Maintenance Changes

Version	Description
V6.3-163	The compiler now emits the informational diagnostic STREMPTY if a STRUCTURE is declared with no fields.
V6.3-164	The compiler now assumes the VOLATILE attribute for all formal arguments for a subprogram in which IARGCOUNT or IARGPTR is used. See Section 1.4.2 for additional details.
V6.4-165	The DPROD intrinsic is now generic and accepts REAL*8 arguments, giving a REAL*16 result. See Section 2.2.4 for additional information.
V6.4-166	The compiler now produces more compact object modules and machine code listings when a variable is data initialized using a repeated character or Hollerith constant.
V6.4-168	The compiler now correctly implements the BLAS routines CSROT and ZDROT as having a real argument S.
V6.4-169	The listing file now more correctly reflects the effects of CDEC$ ALIAS.
V6.4-170	If an untyped bit or Hollerith constant is used in an aritmetic-IF, its type is assumed to be INTEGER*4.
V6.4-171	The compiler no longer gives a MULDECNAM error if, in a recursive subroutine, there is a call to a contained entry point passing an alternate return label.
V6.4-172	The compiler now correctly parses the CDEC$ OPTIONS /ALIGN=RECORD directive, and properly accounts for UNIONs when aligning.
V6.4-173	The compiler no longer fails with a bugcheck during CSE for certain programs using an array reference with a constant subscript as the upper bound of a DO loop.
V6.4-174	The compiler now disallows dummy arguments of ENTRYs in data initialization.
V6.4-176	The compiler now produces better diagnostics for invalid bit constants.
V6.4-177	The compiler now forces STRUCTUREs derived from DICTIONARY statements to be packed, even if /ALIGN=RECORDS=NATURAL is in effect. Use CDD/Repository alignment attributes to align DICTIONARY fields.
V6.4-178	The compiler no longer allows the recursive use of a subroutine's entry name as a function.
V6.4-179	The compiler now issues the YEAR2000 informational message if the IDATE or DATE intrinsics are used. This can be controlled with /WARNINGS=NOUSAGE.
V6.4-180	The compiler no longer generates incorrect code when passing a pointee to a non-adjustable character array as an actual argument.
V6.4-181	The compiler now issues at most one CHACONCONTD diagnostic while processing a character constant format.
V6.4-182	The compiler no longer fails with a bugcheck during CODE for certain programs using DCMPLX of a variable EQUIVALENCEd into COMMON when /NOOPT is in effect.
V6.4-183	The compiler no longer fails with a bugcheck during LOCAL when compiling certain functions that have both COMPLEX and REAL entry points.
V6.4-184	The compiler no longer fails with a bugcheck for when compiling certain functions that assign a pointee value to the function return value.
V6.4-185	If an INTEGER*4 constant value is converted to INTEGER*2, overflow is detected in all appropriate cases.
V6.4-187	If an untyped binary, octal or hexadecimal constant is used in an arithmetic operation with an integer constant, the untyped constant assumes the INTEGER*4 type, even if the integer constant's value implies that its type be INTEGER*2.
V6.5-188	The DATE_AND_TIME intrinsic routine is now supported.

(continued on next page)

Table A–1 (Cont.) Compaq Fortran Maintenance Changes

Version	Description
V6.5-189	Use of the LOC intrinsic on an untyped EXTERNAL name where IMPLICIT NONE is in effect no longer results in an INTFUNARG error.
V6.5-190	Implement /WARN=[NO]UNREACHABLE.
V6.5-191	The use of %DESCR, %REF or %VAL in an I/O list is now properly diagnosed with an INVPERUSE error.
V6.5-192	The compiler no longer bugchecks if OPTIONS /BLAS=keyword is used.
V6.5-193	The compiler no longer outputs invalid debug symbol table information for record structures when /ALIGN is in effect.
V6.5-194	An unformatted I/O statement with * as the unit specification now gives an INVIOSPEC error.
V6.5-195	The compiler no longer bugchecks during GRAPH when /OPT=LEVEL=4 is in effect for certain programs which repeatedly evaluate character substrings.
V6.5-196	The optimizer now considers COMMON variables to be possibly accessed during an OPEN statement (a USEROPEN may access COMMON). Also, the compiler now considers the number of routine calls in a program unit when deciding whether or not to keep a copy of a COMMON variable in a register, as it has to save and restore the variable around each call.
V6.5-197	A new informational diagnostic COMNAMCONF is issued if a COMMON name is found to be the same as some other global entity in the same program unit. This is a violation of the Fortran 77 standard and causes errors on other platforms. This message can be disabled with /WARNINGS=NOUSAGE.
V6.5-198	The compiler now accepts the syntax CHARACTER*(expr) FUNCTION where "expr" is an integer constant expression. This is Fortran 77 standard syntax which previously was not accepted.
V6.5-199	Correct a problem created by edit V6.5-198 that could cause routines with more than six arguments to fail to compile.
V6.6-201	Name change to Compaq Fortran 77, version 6.6.