OpenVMS Compatibility Between VAX and Alpha

Order Number: AA–PYQ4C–TE

May 1995

This manual compares and contrasts OpenVMS on VAX and Alpha computers, focusing on the features provided to end users, system managers, and programmers.

Revision/Update Information: This manual supersedes *OpenVMS Compatibility Between VAX and Alpha*, OpenVMS AXP Version 6.1 and OpenVMS VAX Version 6.1.

Software Version: OpenVMS Alpha Version 6.2
OpenVMS VAX Version 6.2

Digital Equipment Corporation
Maynard, Massachusetts

May 1995

This document was prepared using VAX DOCUMENT Version 2.1.

ZK6310

This document is available on CD–ROM.

Send Us Your Comments

We welcome your comments on this or any other OpenVMS manual. If you have suggestions for improving a particular section or find any errors, please indicate the title, order number, chapter, section, and page number (if available). We also welcome more general comments. Your input is valuable in improving future releases of our documentation.

You can send comments to us in the following ways:

- Internet electronic mail: openvmsdoc@zko.mts.dec.com
- Fax: 603-881-0120 Attn: OpenVMS Documentation, ZKO3-4/U08
- Online form

  Print or edit the online form SYS$HELP:OPENVMSDOC_COMMENTS.TXT. Send the completed online form by electronic mail to our Internet address, or send the completed hardcopy form by fax or through the postal service.

  Please send letters or the form to:

  ```
  Digital Equipment Corporation
  Information Design and Consulting
  OpenVMS Documentation
  110 Spit Brook Road, ZKO3-4/U08
  Nashua, NH  03062-2698
  USA
  ```

Thank you.

How To Order Additional Documentation

Use the following table to order additional documentation or information. If you need help deciding which documentation best meets your needs, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Location	Call	Fax	Write
U.S.A.	DECdirect 800.DIGITAL 800.344.4825	Fax: 800.234.2298	Digital Equipment Corporation P.O. Box CS2008 Nashua, NH 03061
Puerto Rico	809.781.0505	Fax: 809.749.8300	Digital Equipment Caribbean, Inc. 3 Digital Plaza, 1st Street, Suite 200 P.O. Box 11038 Metro Office Park San Juan, Puerto Rico 00910−2138
Canada	800.267.6215	Fax: 613.592.1946	Digital Equipment of Canada, Ltd. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6 Attn: DECdirect Sales
International	—	—	Local Digital subsidiary or approved distributor
Internal Orders	DTN: 264.3030 603.884.3030	Fax: 603.884.3960	U.S. Software Supply Business Digital Equipment Corporation 10 Cotton Road Nashua, NH 03063−1260

ZK−7654A−GE

Contents

2 Interoperability of OpenVMS VAX and OpenVMS Alpha

3 Migration When You're Ready

4 Ensuring the Portability of Applications

A DCL Differences

Index

Examples

Figures

Tables

Preface

Intended Audience

This manual is of primary interest to current OpenVMS VAX users who are considering the addition of an OpenVMS Alpha system to their computing environment and to users who are planning to migrate OpenVMS VAX applications to OpenVMS Alpha systems.

Document Structure

This manual consists of the following chapters and an index:

- Chapter 1 describes the similarities and differences between OpenVMS VAX and OpenVMS Alpha and some of the new features introduced for both systems.
- Chapter 2 describes how OpenVMS VAX and OpenVMS Alpha systems can interoperate in networks and VMSclusters.
- Chapter 3 lists the features available with the OpenVMS Alpha Version 6.2 operating system. It also lists some of the third-party applications that are available on OpenVMS Alpha systems and describes the migration products and services available from Digital.
- Chapter 4 describes how to assess the portability of an OpenVMS VAX application. It also discusses the differences between the OpenVMS VAX and the OpenVMS Alpha programming environments and presents guidelines for new program development on OpenVMS VAX.

Related Documents

To find out more about topics discussed in this manual, refer to the following table for the topic and related document.

Topic	Document
Debugger features that contribute to migration	*OpenVMS Debugger Manual*
DECnet for OpenVMS	*DECnet for OpenVMS Networking Manual*
DECnet for OpenVMS utilities	*DECnet for OpenVMS Network Management Utilities*
Delta/XDelta changes for OpenVMS Alpha	*OpenVMS Delta/XDelta Debugger Manual*

Device drivers, user-written, for OpenVMS Alpha	*Creating an OpenVMS AXP Step 2 Device Driver from a Step 1 Device Driver* *Creating an OpenVMS AXP Step 2 Device Driver from an OpenVMS VAX Device Driver* *OpenVMS AXP Device Support: Reference*
DPML (Digital Portable Mathematics Library)	*Digital Portable Mathematics Library*
Help Message Utility	*OpenVMS System Messages: Companion Guide for Help Message Users*
Linker changes for OpenVMS Alpha	*OpenVMS Linker Utility Manual*
MACRO–64 assembler	*MACRO–64 Assembler for OpenVMS AXP Systems Reference Manual*
PALcode	*Alpha Architecture Reference Manual*
Planning for migration	*Migrating to an OpenVMS AXP System: Planning for Migration*
Porting applications written in mid- and high-level languages	*Migrating to an OpenVMS AXP System: Recompiling and Relinking Applications*
Porting VAX MACRO applications	*Migrating to an OpenVMS AXP System: Porting VAX MACRO Code*
SDA commands for OpenVMS Alpha	*OpenVMS AXP System Dump Analyzer Utility Manual*
Security	*OpenVMS Guide to System Security*
Translating OpenVMS VAX images into OpenVMS Alpha images	*DECmigrate for OpenVMS AXP Systems Translating Images*
System management differences and similarities between OpenVMS Alpha and OpenVMS VAX	*A Comparison of System Management on OpenVMS AXP and OpenVMS VAX*
VAXcluster and VMScluster systems	*VMScluster Systems for OpenVMS* *Guidelines for VMScluster Configurations*
VMScluster system restrictions	*OpenVMS Version 6.2 Release Notes*

Conventions

The name of the OpenVMS AXP operating system has been changed to OpenVMS Alpha. Any references to OpenVMS AXP or AXP are synonymous with OpenVMS Alpha or Alpha.

The following conventions are used in this manual:

boldface text	Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason. Boldface text is also used to show user input in Bookreader versions of the manual.
italic text	Italic text indicates important information, complete titles of manuals, or variables. Variables include information that varies in system messages (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where *device-name* contains up to five alphanumeric characters).
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.

-	A hyphen in code examples indicates that additional arguments to the request are provided on the line that follows.
numbers	All numbers in text are assumed to be decimal, unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

1
Overview

This chapter describes:

- Lineage of OpenVMS Alpha
- Similarities and differences at the DCL level
- Similarities and differences between OpenVMS VAX and OpenVMS Alpha in the end-user, system management, and programming environments

1.1 Introduction

The purpose of this manual is to provide OpenVMS VAX users with the information they need to assess the impact of adding one or more OpenVMS Alpha systems to their computing environments. The manual focuses on the similarities and differences between OpenVMS Alpha Version 6.2 and OpenVMS VAX Version 6.2.

OpenVMS Alpha runs on Digital's Alpha computers, which are reduced instruction set computers (RISC). OpenVMS Alpha systems provide as much compatibility as possible with OpenVMS VAX systems without compromising the advantages offered by the Alpha architecture. OpenVMS VAX customers can use the RISC technology of OpenVMS Alpha with little change in their computing environment.

Experienced OpenVMS VAX system managers will find their knowledge and most of their practices transferable to the OpenVMS Alpha environment. Many OpenVMS Alpha system managers compare the degree of change to that introduced by the change from VAX VMS Version 4.*n* to VAX VMS Version 5.0.

Many Digital customers use OpenVMS VAX to run their applications that require high standards of availability, scalability, and data integrity. They depend on its reliability, robust engineering, and leadership features, such as VAXcluster systems. Digital anticipates that many of its customers will add OpenVMS Alpha systems to their computing environments. Digital recognizes that this is an evolutionary process and will differ from customer to customer. Digital is committed to providing a clear coexistence environment and migration path between OpenVMS VAX and OpenVMS Alpha.

1.2 Lineage of OpenVMS Alpha

In 1978, Digital Equipment Corporation released Version 1.0 of the VMS operating system. Each new release since Version 1.0 represents a balance between compatibility with earlier releases and the introduction of new features and new technology that enable users to do their work more cost-effectively. A further development in this evolution was the introduction of OpenVMS AXP in November 1992.

Since the release of OpenVMS AXP Version 1.0, which was based on VMS Version 5.4–2, additional OpenVMS VAX features have been added to each new release. OpenVMS AXP Version 1.0 was followed by OpenVMS AXP Version 1.5. The next release of OpenVMS AXP was numbered Version 6.1 to acknowledge its functional equivalence with OpenVMS VAX Version 6.1. For Version 6.2, Digital changed the name of OpenVMS AXP to OpenVMS Alpha.

For the purposes of this manual, functional equivalence between OpenVMS VAX and OpenVMS Alpha is defined as the same type of functionality with possible slight variations in the implementation or the user interface.

When OpenVMS VAX Version 6.1 and OpenVMS AXP Version 6.1 were released, some exceptions to functional equivalence existed. Most of these exceptions are eliminated with this release (see Table 3–1).

1.3 End-User's Environment

The end-user's environment on OpenVMS Alpha Version 6.2 is virtually the same as that on OpenVMS VAX Version 6.2, as described in the following sections.

1.3.1 DIGITAL Command Language (DCL)

The DIGITAL Command Language (DCL), the standard user interface to OpenVMS, remains essentially unchanged with OpenVMS Alpha. All commands and qualifiers available on OpenVMS VAX are also available on OpenVMS Alpha, except for a few, as shown in Appendix A. In addition, a few qualifiers are available only on OpenVMS Alpha, as shown in Appendix A.

1.3.2 DCL Help

DCL help is available on OpenVMS Alpha. Most of the DCL help text is common to both OpenVMS Alpha and OpenVMS VAX systems. For a few topics, information that is specific to one system is included in the display for both systems. System-specific information is identified by the phases *On VAX* and *On Alpha*.

1.3.3 DCL Command Procedures

DCL command procedures that use commands, qualifiers, and lexical functions available on OpenVMS VAX will continue to work on OpenVMS Alpha systems without change, except for command procedures that contain those few qualifiers not available on OpenVMS Alpha.

1.3.4 Databases

Standard databases, such as Oracle Rdb, function the same on OpenVMS VAX and OpenVMS Alpha systems. Most third-party databases that are available for OpenVMS VAX are also available for OpenVMS Alpha. Many of them are shown in Table 3–2.

1.3.5 DECforms

The DECforms interface is unchanged. Applications that use the predecessor to DECforms, TDMS, can be moved to OpenVMS Alpha with the aid of a TDMS emulator or a TDMS converter. The VAX TDMS Emulator for OpenVMS and the VAX TDMS to DECforms Converter for OpenVMS VAX are produced by Praxa Limited of Melbourne, Australia and available through Digital.

1.3.6 DECwindows Motif

DECwindows Motif for OpenVMS VAX and DECwindows Motif for OpenVMS Alpha contain virtually identical functionality. The X11R5 display server that ships with OpenVMS Alpha does, however, contain additional extensions (for example, Shape, which gives the ability to create non-rectangular windows, and scalable fonts) that are not available on the OpenVMS VAX platform.
For more information about these X11R5 features and extensions, see *Managing DECwindows Motif for OpenVMS Systems*.

1.3.7 Editors and Formatter

The EVE and EDT editors are unchanged. EVE is the default editor for OpenVMS VAX and for OpenVMS Alpha. EDT was the default editor for OpenVMS VAX prior to Version 6.0.

The TECO editor and the DSR formatter are also provided with OpenVMS Alpha. They, too, are unchanged.

1.3.8 Help Message Utility

The Help Message utility is available on both OpenVMS VAX and OpenVMS Alpha systems. It enables users to access online descriptions of system messages on a character-cell terminal (including DECterm windows).

Help Message operates through a DCL interface that accesses message descriptions in a text file. This text file is derived from the latest version of the OpenVMS system messages documentation and, optionally, from other source files, including user-supplied message documentation. For more information about Help Message, see *OpenVMS System Messages: Companion Guide for Help Message Users*.

1.3.9 Password Generator

Both OpenVMS VAX and OpenVMS Alpha have random password generators, which can be used in place of passwords created by users. However, the random password generator on OpenVMS Alpha systems has a new mechanism for generating passwords. This new mechanism produces nonsense passwords such as *dramnock*, *inchworn*, *mycousia*, and *pestrals* that seem "natural" to users. It makes possible the future development of language-specific random passwords. The list of possible passwords presented by an OpenVMS Alpha system is not hyphenated.

1.4 System Manager's Environment

Most of the OpenVMS VAX Version 6.2 system management utilities, command formats, and tasks are identical in the OpenVMS Alpha environment.

Some components that were available only on OpenVMS VAX Version 6.1 or only on OpenVMS AXP Version 6.1 are now available on both systems, as shown in Table 1–1.

Table 1–1 Components That Achieved Functional Equivalence in OpenVMS Version 6.2

Component	Previously Available
Cluster event notification system services	Alpha only
DECamds Data Analyzer	VAX only
Dynamic device recognition	Alpha only
Full name support, including support by DECdtm	VAX only
LAT selected features	Alpha only
New proxy service and new security server	VAX only
RECALL command, additional support	Alpha only
SCSI–2 device support for Tagged Command Queuing	VAX only

However, there are some system management differences that must be considered to properly set up, maintain, secure, and optimize OpenVMS Alpha systems and to establish proper network connections. These differences fall into two categories:

- Implementation differences for common components
- System management components not available on both OpenVMS VAX and OpenVMS Alpha

The differences are briefly described in this section and in greater detail in *A Comparison of System Management on OpenVMS AXP and OpenVMS VAX* and in the *OpenVMS System Manager's Manual*.

1.4.1 Common Components With Implementation Differences

The components described in this section exist on both OpenVMS VAX and OpenVMS Alpha systems, but the implementations on VAX and Alpha differ, or the OpenVMS VAX implementation differs from earlier versions of OpenVMS VAX.

1.4.1.1 Disk Quotas

You might need to increase disk quotas on OpenVMS Alpha disks that store translated OpenVMS VAX images and native OpenVMS Alpha images. Translated images are OpenVMS Alpha executable images produced by the VAX Environment Software Translator (VEST), the major component of DECmigrate (described in Section 4.2.3). The reasons why increased disk quotas are usually needed are listed in Table 1–2.

Table 1–2 Larger Disk Quotas Needed on OpenVMS Alpha

Condition	Explanation
Translated images	Require more disk space because each image includes both Alpha code and the original VAX code.
Native OpenVMS Alpha images	Require more disk space because RISC images typically contain more instructions and code to establish the linkage between procedure calls. The default values for related memory quotas have been adjusted on OpenVMS Alpha systems.

1.4.1.2 I/O Subsystem Configuration Commands

On OpenVMS VAX systems, the System Generation utility (SYSGEN) is used to configure the I/O subsystem. On OpenVMS Alpha systems, SYSGEN is used for some configuration tasks, and the System Management utility (SYSMAN) and the AUTOGEN command procedure are used for others, as shown in Table 1–3.

Table 1–3 A Comparison of I/O Subsystem Configurations

	OpenVMS VAX	OpenVMS Alpha
SYSGEN	Modify system parameters[1] Load page and swap files Create additional page files Create additional swap files Load device drivers	Modify system parameters[1] Load page and swap files Create additional page files Create additional swap files
SYSMAN	Not used	Load device drivers

[1]Although SYSGEN is available for modifying system parameters, Digital recommends that you use AUTOGEN and its data files instead, or that you use SYSMAN between boots, for dynamic parameters.

OpenVMS VAX command procedures that use commands such as SYSGEN AUTOCONFIGURE ALL must be modified if they are copied to OpenVMS Alpha systems as part of your migration effort.

1.4.1.3 Menu-Driven Maintenance Procedure

A replacement for the standalone BACKUP utility was introduced with the OpenVMS AXP Version 6.1 distribution CD–ROM. It was also provided on the OpenVMS VAX Version 6.1 distribution CD–ROM as an alternative to the standalone BACKUP utility (which is still currently supported on OpenVMS VAX systems).

It is a menu-driven procedure which enables you to enter a DCL environment, from which you can perform backup and restore operations on the system disk (instead of using standalone BACKUP).

For more detailed information about using the menu-driven procedure, see the following documentation:

- On OpenVMS VAX, see the *OpenVMS System Manager's Manual*
- On OpenVMS Alpha, see the *OpenVMS Alpha Version 6.2 Upgrade and Installation Manual*

1.4.1.4 MONITOR POOL Command

The Adaptive Pool Management feature of OpenVMS VAX and OpenVMS Alpha has made the DCL command MONITOR POOL obsolete. Adaptive Pool Management automatically manages the creation and sizing of nonpaged pool lookaside lists.

You can obtain information about nonpaged and paged dynamic pool on OpenVMS VAX through the use of the DCL command SHOW MEMORY and the System Dump Analyzer (SDA) command SHOW POOL. You can obtain the same information on OpenVMS Alpha with the SDA command SHOW POOL and its qualifiers /RING_BUFFER and /STATISTICS.

1.4.1.5 Name Changes for Files Supplied with the Operating Systems

The name changes for certain files supplied with the operating system are shown in Table 1–4.

Table 1–4 Operating System File Name Changes

OpenVMS VAX Version 5.5	OpenVMS VAX Version 6.0 and later	OpenVMS AXP Version 1.0 and later
SYSTARTUP_V5.COM	SYSTARTUP_VMS.COM	SYSTARTUP_VMS.COM
VAXVMSSYS.PAR	VAXVMSSYS.PAR	ALPHAVMSSYS.PAR

1.4.1.6 Page Size

OpenVMS VAX and OpenVMS Alpha systems allocate and deallocate memory for processes in units called **pages**. On OpenVMS VAX systems, a page is 512 bytes. On OpenVMS Alpha systems, a page will be one of four values, 8K, 16K, 32K, or 64K bytes. An OpenVMS Alpha system implements only one of the four page sizes; the initial set of Alpha computers use an 8K byte (8192 bytes) page.

This difference in page size is significant to OpenVMS system managers in two ways:

- Process quotas and limits, and system parameters might require adjustment to account for the additional resources (especially memory resources) users might require. For example, higher values might be necessary for the PGFLQUOTA process quota and the GBLPAGES system parameter.
- In a number of cases, OpenVMS Alpha interactive utilities present to and accept from users units of memory in a 512-byte quantity called a **pagelet**. Thus, one Alpha pagelet is the same size as one VAX page. On an Alpha computer with 8K byte pages, 16 Alpha pagelets equal 1 Alpha page.

In your OpenVMS Alpha environment, you will need to notice when page or pagelet values are being shown in memory displays. If a memory value represents a page on an Alpha system, the documentation might refer to "CPU-specific pages." This convention indicates possible significant differences in the size of the memory being represented by the page unit, depending on the Alpha computer in use (8K, 16K, 32K, or 64K byte pages). In general, OpenVMS Alpha utilities display CPU-specific page values when the data represents physical memory.

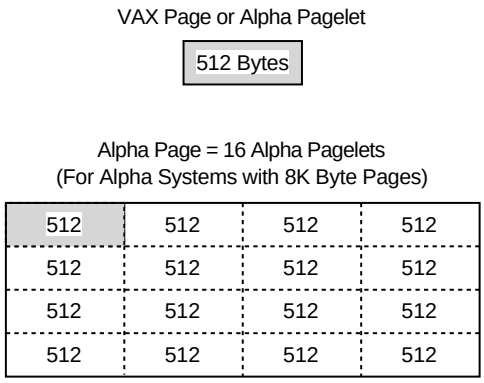
Internally, for the purposes of memory allocation, deletion, and protection, OpenVMS Alpha will round up (if necessary) the value you supply in pagelets to a number of CPU-specific pages.

The use of pagelets provides compatibility for OpenVMS VAX system managers and application programmers who are accustomed to thinking about memory values in 512-byte units. In a VMScluster system with OpenVMS VAX and OpenVMS Alpha nodes, it is helpful to know that a VAX page and an Alpha pagelet represent a common unit of 512 bytes. Also, existing OpenVMS VAX applications do not need to change parameters to the memory management system services when the applications are ported to OpenVMS Alpha.

OpenVMS Alpha does not allocate or deallocate a portion of a page. The user-interface quantity called a pagelet is not used internally by the operating system. Pagelets are accepted and displayed by utilities so that users and applications operate with the information that each VAX page value and each Alpha pagelet value equal a common 512-byte quantity.

Figure 1–1 illustrates the relative sizes of a VAX page, an Alpha 8K byte page, and an Alpha pagelet.

Figure 1–1 Comparison of VAX and Alpha Page Size



1.4.1.7 Security

On Alpha systems, OpenVMS contains all the security features of OpenVMS VAX with the exception of DECnet connection auditing. OpenVMS VAX Version 6.0 was successfully evaluated at the C2 level in September 1993. A Security RAMP (Rating Maintenance Phase) is currently underway for OpenVMS VAX Version 6.1, and the C2 rating is expected in the very near future. A Security RAMP is also currently underway for OpenVMS Alpha Version 6.1. OpenVMS VAX Version 6.2 and OpenVMS Alpha Version 6.2, and all future releases, will also be entered in the RAMP process in order to maintain the C2 Security rating.

1.4.1.8 VMScluster Systems

A VMScluster system can consist entirely of OpenVMS Alpha nodes or a combination of one or more OpenVMS VAX nodes and one or more OpenVMS Alpha nodes. The system management of a VMScluster system is essentially the same as that of a VAXcluster system.

For more information on VMScluster systems, see Section 2.4.

1.4.2 System Management Features Not Available on Both Systems

The features described in this section are not available on both OpenVMS VAX and OpenVMS Alpha systems. Components that exist only on OpenVMS VAX or only on OpenVMS Alpha are either planned, under investigation, or not planned for the other system, as shown in Table 1–5.

Table 1–5 System Management Features Not Available on Both Systems

Feature	On VAX or Alpha	Status
Card reader and input symbiont	VAX	Not planned for Alpha
DECevent utility	Alpha	Under investigation for VAX
DECnet connection auditing	VAX	Planned for Alpha
DECnet DDCMP support	VAX	Not planned for Alpha
DECnet host-based routing	VAX	Not planned for Alpha
Dump file off the system disk	VAX	Planned for Alpha
Installation of the operating system with PCSI	Alpha	Planned for VAX
MSCP dynamic load balancing	VAX	Planned for Alpha
Patch utility	VAX	Limited functionality under investigation for Alpha
SCSI VMSclusters	Alpha	Under investigation for VAX (see Section 2.4)
Selected optional software products	VAX or Alpha	See Section 1.4.2.4, Section 3.2, and Section 3.3
Shadowing dump file failover	VAX	Under investigation for Alpha
Snapshot facility	VAX	Not planned for Alpha
SYSMAN I/O function for loading device drivers[1]	Alpha	Under investigation for VAX

[1] For an example of the SYSMAN I/O function for loading device drivers, see Table 1–3.

1.4.2.1 DECevent Event Management Utility

OpenVMS AXP Version 6.1 includes the DECevent utility (DECevent), which provides the interface between a system user and the system's event log files. DECevent allows system users to produce ASCII reports derived from system event entries. DECevent uses the system event log file, SYS$ERRORLOG:ERRLOG.SYS, as the default input file for event reporting unless another file is specified.

1.4.2.2 MSCP Dynamic Load Balancing

MSCP dynamic load balancing is used by VMScluster systems to balance the I/O load efficiently among systems within a VMScluster. Dynamic load balancing automatically checks server activity every five seconds. If activity to any server is excessive, the serving load automatically shifts to other servers in the cluster.

1.4.2.3 Installation of the Operating System with PCSI

The POLYCENTER Software Installation (PCSI) utility is provided with OpenVMS VAX and OpenVMS Alpha. On VAX, the use of PCSI is limited to installing layered products. On Alpha, PCSI is also used to install OpenVMS.

1.4.2.4 Optional Software Products Not Supported

Some optional Digital software products that are supported on OpenVMS VAX are not yet supported on OpenVMS Alpha Version 6.2. If you copy existing startup procedures from one of your OpenVMS VAX computers to an OpenVMS Alpha computer, you must comment out the calls to the startup procedures of currently unsupported optional software products. The *Alpha Applications Catalog* lists all the available Digital optional software products and third-party applications (to obtain a copy, see Section 3.2).

1.4.2.5 Patch Utility

The Patch utility is not offered on OpenVMS Alpha. However, investigation is underway regarding support for the PATCH/ABSOLUTE function on OpenVMS Alpha.

1.4.2.6 Snapshot Facility

The Snapshot facility (Snapshot), sometimes referred to as Fastboot, lets you reduce system startup time by booting OpenVMS VAX from a saved system image disk file. Snapshot can be used only for a standalone system (that is, a system that is not in a VAXcluster environment).

For more information, see the *OpenVMS System Manager's Manual*.

1.5 Programming Environment

The similarities and differences in the programmer's environment between OpenVMS VAX Version 6.2 and OpenVMS Alpha Version 6.2 are outlined in this section and described in more detail in Chapter 4, except where noted otherwise. Chapter 4 also provides guidelines for developing applications that will run on both OpenVMS VAX and OpenVMS Alpha and additional guidelines for developing applications that will run in a mixed-architecture VMScluster.

The same types of program development tools that you are accustomed to using on OpenVMS VAX are available on OpenVMS Alpha systems including the Linker utility, the Librarian utility, the OpenVMS Debugger (also known as the symbolic debugger), the Delta/XDelta Debugger, and run-time libraries, including the parallel processing run-time library (PPL RTL).

1.5.1 RISC Architecture of Alpha

The RISC architecture of Alpha computers differs from the complex instruction set computer (CISC) architecture of VAX computers. The differences are particularly apparent when multiple processes or multiple execution threads in the same process access the same region of memory. An asynchronous system trap (AST) routine whose execution preempts the processing of a main routine is one example of concurrent threads within a single process.

The VAX architecture, through its microcode, provides instruction semantic guarantees that the Alpha architecture does not. Complex atomic operations and synchronization guarantees incur overhead that RISC architectures are designed to avoid.

For example, on a VAX computer, you can perform complex memory operations atomically and can access data at byte and word memory locations. If your code contains such VAX architectural dependencies, you likely will need to either use a compiler qualifier that mitigates the dependencies or make changes to your code.

1.5.2 User-Written Device Drivers

Both OpenVMS VAX and OpenVMS Alpha support user-written device drivers. The overall structure of device drivers is the same on each architecture but source changes are required to migrate a device driver from one architecture to the other. In addition, on OpenVMS Alpha, device drivers can be written in C.

For more information, see Section 4.3.3.

1.5.3 Compilers

Most DEC compilers (and the MACRO–64 assembler) are available on OpenVMS Alpha Version 6.2 as shown in the following list:

- DEC Ada
- DEC BASIC
- DEC C
- DEC C++
- DEC COBOL
- DEC Fortran
- DEC Pascal
- MACRO–32
- DEC OPS5
- DEC PL/I

A Bliss compiler is also available for OpenVMS Alpha, although it is unsupported. It is included on the OpenVMS Freeware CD that ships with OpenVMS VAX Version 6.2 and with OpenVMS Alpha Version 6.2.

These compilers are also available on OpenVMS VAX Version 6.2, except for the MACRO–32 compiler, which converts VAX MACRO code into Alpha machine code. The MACRO–32 compiler is bundled with OpenVMS Alpha to ease migration.

The differences between the compilers on VAX and Alpha may require some minor changes to your code. For more information, see Chapter 4.

1.5.4 Native Assembler

The native assembler, MACRO–64, is not bundled with the OpenVMS Alpha operating system. Instead, it is available as an optional software product. For more information, see Section 4.3.2.

1.5.5 DECmigrate for OpenVMS AXP

DECmigrate for OpenVMS AXP is an optional Digital software product that translates OpenVMS VAX images into OpenVMS Alpha images. If the native compiler for your application is not available, you can translate it.

DECmigrate for OpenVMS AXP can also be used to analyze code to determine how easy or difficult it would be to migrate it. For more information about DECmigrate for OpenVMS AXP, see Section 4.2.3.

1.5.6 Linker

The way certain linking tasks, such as creating shareable images, are performed is different on OpenVMS Alpha systems. You may need to modify the LINK command used to build your application. For example, instead of creating a transfer vector file for a shareable image, you must create a linker options file and declare universal symbols by specifying the SYMBOL_VECTOR= option. For more information, see Section 4.3.1.

1.5.7 Librarian

The Librarian utility provides a new qualifier, /VAX. The /VAX qualifier directs the Librarian utility to create an OpenVMS VAX object module library when used with the /CREATE and /OBJECT qualifiers or an OpenVMS VAX shareable image library when used with the /SHARE qualifier. OpenVMS Alpha libraries are the default on OpenVMS Alpha systems. For more information, see the *OpenVMS Command Definition, Librarian, and Message Utilities Manual*.

1.5.8 Debuggers

The OpenVMS Debugger provides several features that facilitate debugging OpenVMS Alpha code. These features address the architectural differences that exist between VAX and Alpha computers. For example, the /UNALIGNED_DATA qualifier used with the SET command enables you to detect unaligned data. On VAX, the OpenVMS Debugger offers a new facility, the Heap Analyzer, that represents graphically, in real time, the utilization of dynamic memory (heap). This can be used for user-mode utility or application code and is only available on OpenVMS VAX at this time. For more information about the OpenVMS Debugger, see Section 4.3.4.

The Delta/XDelta Debugger provides several new commands and changes to existing commands for debugging OpenVMS Alpha programs. For more information, see Section 4.3.5.

The OpenVMS Alpha System-Code Debugger lets you use the familiar OpenVMS Debugger interface to observe and manipulate system code interactively as it executes. This debugger is not available on OpenVMS VAX. For more information, see Section 4.3.6.

1.5.9 System Dump Analyzer

The System Dump Analyzer on OpenVMS Alpha systems is almost identical to the utility provided on OpenVMS VAX systems. Most commands, qualifiers, and displays are the same. For more information, see Section 4.3.7.

The Crash Log Utility Extractor (CLUE) is available on both OpenVMS VAX and OpenVMS Alpha. It was created to make the information for debugging crash dumps more accessible and more useful. For more information, see Section 4.3.8.

1.5.10 Programming Components Not Available on Both Systems

Some programming components are not available on both OpenVMS VAX and OpenVMS Alpha. They are shown in Table 1–6 and described in this section.

Table 1–6 Programming Components Not Available on Both Systems

Component	On VAX or Alpha	Status
H_float and D_float floating-point data types	VAX	Not planned for Alpha (see Section 1.5.10.1)
Heap Analyzer in Debugger	VAX	Under investigation for Alpha (see Section 1.5.8)
OpenVMS Alpha System-Code Debugger	Alpha	Not planned for VAX (see Section 1.5.8)
Support for device drivers written in C	Alpha	Not planned for VAX (see Section 4.3.3)

Before moving any OpenVMS VAX applications to an OpenVMS Alpha system, Digital recommends that you become familiar with the differences in the OpenVMS Alpha program development environment as described in this section and in Chapter 4.

1.5.10.1 Floating-Point Data Types

Support for the H_float floating-point and full-precision D_float floating-point data types has been eliminated from the hardware to improve overall system performance.

Alpha hardware converts D_float floating-point data to G_float floating-point data for processing. On VAX computers, the D_float floating-point data type has 56 fraction bits (D56) and 16 decimal digits of precision.

The H_float floating-point and D_float floating-point data types can usually be replaced by the G_float floating-point data type or one of the IEEE formats. However, if you require the H_float floating-point data type or the extra precision of the D_float floating-point data type, you may have to translate part of your application with DECmigrate for OpenVMS AXP.

1.5.10.2 Vector Processing

Vector processors were an option for VAX 6500 and VAX 9000 computers to provide higher performance for numerically intensive applications. Alpha computers and later versions of VAX computers do not provide this option because their basic designs provide high-speed calculations.

If you used this option for Fortran applications, you do not have to make any changes to your code for it to run on Alpha computers or later models of VAX computers. You only have to recompile it with the DEC Fortran for OpenVMS Alpha compiler. If you used vector-specific support in VAX MACRO applications, you will need to make changes to your code before recompiling it with the MACRO–32 compiler for OpenVMS Alpha.

Image files whose source files included vector instructions cannot be translated. The VAX Environment Software Translator (VEST) component of DECmigrate does not support them.

2

Interoperability of OpenVMS VAX and OpenVMS Alpha

This chapter describes the similarities and differences, where they exist, of the following topics:

- Interoperability of OpenVMS VAX and OpenVMS Alpha on a DECnet network and on a TCP/IP network
- DECnet (Phase IV) network features and management
- DECnet/OSI network features
- Interoperability of OpenVMS VAX and OpenVMS Alpha in a VMScluster

2.1 Interoperability on a Network

OpenVMS Alpha and OpenVMS VAX systems can interoperate in a network, sharing system resources and enabling communication with local and remote nodes, in the same way that OpenVMS VAX systems interoperate. Network management of OpenVMS Alpha nodes is also similar to network management of OpenVMS VAX nodes. However, some differences exist due to architectural and implementation differences.

2.1.1 Interoperability Using DECnet for OpenVMS and DECnet/OSI

DECnet for OpenVMS and DECnet/OSI are used to establish networking connections with other OpenVMS VAX and OpenVMS Alpha nodes. Both products are available for OpenVMS VAX systems and for OpenVMS Alpha systems.

DECnet for OpenVMS, formerly known as DECnet–VAX, implements Phase IV of DNA (Digital Network Architecture). The features of DECnet for OpenVMS AXP are similar to those of the DECnet–VAX software that is part of VMS Version 5.4–3, with a few exceptions, as noted in Section 2.2. This product is included with the OpenVMS operating system and ships on the OpenVMS CD.

DECnet/OSI for OpenVMS, which implements Phase V of DNA, is an ISO-compliant product. DECnet/OSI for OpenVMS conforms to the Open Systems Interconnection (OSI) networking standards defined by the International Organization for Standardization (ISO). DECnet/OSI provides all the features of DECnet Phase IV as well as OSI features that enable participation in open standards based networks. DECnet/OSI is a layered product on OpenVMS and ships on the layered product CD.

Full names support for DECnet/OSI was introduced in OpenVMS VAX Version 6.1 and is now available on both OpenVMS VAX Version 6.2 and OpenVMS Alpha Version 6.2.

File transfers over the DECnet network, including copying and printing, can be done between OpenVMS VAX and OpenVMS Alpha systems running either DECnet for OpenVMS or DECnet/OSI.

The SET HOST command enables full interoperability for DECnet remote login between OpenVMS VAX and OpenVMS Alpha nodes running DECnet for OpenVMS or DECnet/OSI.

2.1.2 Interoperability Using TCP/IP Networking on OpenVMS Systems

TCP/IP software supports communication between computer systems of similar or different design as well as interconnection of various physical networks to form larger networks. There are several vendors of TCP/IP software for OpenVMS (see Appendix A of *TCP/IP Networking on OpenVMS Systems*). Users on an OpenVMS system running TCP/IP software can execute the basic applications shown in Table 2–1.

Table 2–1 Applications of TCP/IP Software for OpenVMS

Application Type	Vendor Specific[1]	Vendor Common[2]
Virtual terminal service	RLOGIN	SET HOST/RLOGIN
	TELNET	SET HOST/TELNET
File access	FTP	COPY/FTP, DIR/FTP
	RCP	COPY/RCP
Disk service	NFS[3]	

[1] Same commands available but qualifiers and parameters may differ

[2] Commands, qualifiers, and parameters identical for all named vendors

[3] Not all implementations support the Network File System (NFS)

The TCP/IP software can also be used to connect to and access the global Internet. For more information about TCP/IP networking on OpenVMS, see *TCP/IP Networking on OpenVMS Systems*.

2.1.3 Network Interfaces

Most of the network protocols, buses, and interconnects that are supported on OpenVMS VAX systems are also supported on OpenVMS Alpha systems, as shown in the following tables.

2.1.3.1 Network Protocols

The network protocols supported on OpenVMS VAX and OpenVMS Alpha systems are shown in Table 2–2.

Table 2–2 Network Protocol Support

Protocol	OpenVMS VAX Versions V5.5-2 through 6.2	OpenVMS Alpha Versions 1.5 through 6.2
DECnet (Phase IV)	Yes	Yes
DECnet/OSI (Phase V)	Yes	Yes
LAD/LAST	Yes	Yes
LAT	Yes	Yes
LAVC	Yes	Yes
TCP/IP[1]	Yes	Yes
X.25	Yes[2]	Yes[3]

[1]Provided by one of the TCP/IP for OpenVMS vendors listed in Appendix A of *TCP/IP Networking on OpenVMS Systems*.

[2]For OpenVMS VAX Version V5.5-2, provided by VAX P.S.I.; for OpenVMS VAX Version 6.0 and later, provided by DECnet/OSI.

[3]For OpenVMS AXP Version 1.5 and Version 1.5-1H1, provided by DEC X.25 Client for OpenVMS AXP software. Running this software, a node can connect to an X.25 network via an X.25 gateway. For systems running DECnet/OSI, X.25 support is provided by X.25 for OpenVMS AXP, which includes both client and native X.25 functionality.

2.1.3.2 Buses

The buses supported on OpenVMS VAX and OpenVMS Alpha systems are shown in Table 2–3. Support is also dependent on the computer model.

Table 2–3 Bus Support

Bus	OpenVMS VAX Versions V5.5-2 through 6.2	OpenVMS Alpha Versions 1.5 through 6.2
BI-bus	Yes	No[1]
DSSI	Yes	Yes
EISA bus	No	Yes[2]
Futurebus+	No	Yes[2]
ISA	No	Yes[3]
PCI	No	Yes[4]
Q–bus	Yes	No[1]
SCSI	Yes	Yes
TURBOchannel	Yes	Yes
UNIBUS	Yes	No[1]
VME	Yes	No[5]
XMI	Yes	Yes

[1]Support not planned.

[2]Except for OpenVMS AXP Version 1.5.

[3]OpenVMS AXP Version 6.1-1H1 and later.

[4]OpenVMS AXP Version 6.1 and later.

[5]A custom Digital offering is available.

2.1.3.3 Interconnects

The interconnects (including their respective protocols and drivers) that are supported on OpenVMS VAX and OpenVMS Alpha systems are shown in Table 2–4.

Table 2–4 Interconnect Support

Interconnect	DECnet[1]	TCP/IP[2]	LAT	Cluster: Computer to Computer	Cluster: Computer to Tape or Disk
Asynchronous line	V	–	–	–	–
ATM[3]	A	A	A	A	–
CI	V	–	–	A,V	A,V
DSSI	–	–	–	A,V	A,V
Ethernet	A,V	A,V	A,V	A,V	–
FDDI[4]	A,V	A,V	A,V	A[5],V	–
SCSI	–	–	–	–	A[6],V[7]
Synchronous line	V	–	–	–	–
Token ring	–	A[8]	–	–	–

[1]Provided by DECnet for OpenVMS and DECnet/OSI.

[2]Provided by one of the TCP/IP for OpenVMS vendors listed in Appendix A of *TCP/IP Networking on OpenVMS Systems*.

[3]Available on TurboChannel only, in point-to-point configurations.

[4]Boot support is not available for all computer models (see Table 2–5).

[5]OpenVMS AXP Versions 1.5 and 1.5-1H1 do not use an FDDI adapter for cluster communications, but Ethernet bridging to FDDI backbones can be used. OpenVMS Alpha, starting with Version 6.1 can use an FDDI adapter for cluster communications.

[6]One computer per SCSI bus, serving tape and disk. Two computers on the SCSI bus, with disks only, starting with OpenVMS Alpha Version 6.2.

[7]One computer per SCSI bus, serving tape and disk.

[8]Available only on OpenVMS Alpha, starting with Version 6.1.

Key

A = OpenVMS Alpha Version 1.5 through Version 6.2
V = OpenVMS VAX Version V5.5-2 through Version 6.2
– = Neither

2.1.3.4 FDDI Boot Support on OpenVMS Alpha

While FDDI is supported as an interconnect, it does not provide boot support for any VAX computer models. However, it does provide boot support for several Alpha computer models, as shown in Table 2–5.

Note

Configurations without FDDI boot support must also be wired to an Ethernet circuit if either of the following conditions exists:

- Operating system software installations or updates are to be performed from an InfoServer
- System is to be a VMScluster satellite (unlikely for larger systems)

Table 2–5 FDDI Boot Support for OpenVMS Alpha Version 6.2

Interconnect	Adapter	Alpha Computer	Boot Support
EISA			
	DEFEA	1000	Yes
	DEFEA	2000	Yes (Except DEC 2000)
	DEFEA	2100	Yes
XMI			
	DEMFA	7000	Yes
	DEMFA	8200	Yes
	DEMFA	8400	Yes
Futurebus+			
	DEFAA	4000	No
	DEFAA	7000	No
TURBOchannel			
	DEFZA	3000	No
	DEFTA	3000	Yes
PCI			
	DEFPA	200	No
	DEFPA	250	No
	DEFPA	400	No
	DEFPA	600	No
	DEFPA	1000	No
	DEFPA	2000	No
	DEFPA	2100	No
	DEFPA	8200	No
	DEFPA	8400	No

2.2 DECnet (Phase IV) Network Features and Management

The similarities and differences in the network features and management for DECnet for OpenVMS AXP (Phase IV) and for DECnet–VAX for VMS Version 5.4–3 are described in this section.

2.2.1 Similarities

The features and management of DECnet for OpenVMS AXP (Phase IV) are similar to those of DECnet–VAX for VMS Version 5.4–3, with some exceptions. The following list shows the features and management tasks that are identical:

- DECnet objects
- DECnet Test Sender/DECnet Test Receiver utility (DTS/DTR)
- Downline load and upline dump operations
- Event logging
- Ethernet monitor (NICONFIG)
- File access listener (FAL)

  (Identical between Alpha nodes and between Alpha and VAX nodes.)
- Loopback mirror testing
- NETCONFIG_UPDATE.COM procedure
- Node name rules
- Product Authorization Key (PAK) name for end-node license (DVNETEND)
- SET HOST capabilities

  (Identical between Alpha nodes and between Alpha and VAX nodes.)
- Size of network
- Task-to-task communication

2.2.2 Differences

The features and management tasks of DECnet for OpenVMS AXP (Phase IV) are similar to those of DECnet–VAX for VMS Version 5.4–3, with some exceptions, as shown in Table 2–6.

Table 2–6 Differences of DECnet Features and Management Tasks

Feature or Task	OpenVMS VAX	OpenVMS Alpha
Cluster alias	Level 1 and Level 2 routing supported on nodes acting as routers for a cluster alias.	Only Level 1 routing supported on nodes acting as routers for a cluster alias.

(continued on next page)

Table 2–6 (Cont.) Differences of DECnet Features and Management Tasks

Feature or Task	OpenVMS VAX	OpenVMS Alpha
Configuring DECnet databases and starting OpenVMS Alpha computer's access to the network		Process for both is the same but subject to the routing limitations and the lack of support for CI, DDCMP, the Distributed Name Service (DNS) node name interface, VAX P.S.I., and certain Network Control Program (NCP) utility command parameters. The functions of the SYS$MANAGER:STARTNET.COM procedure are similar.
Lines supported	CI, asynch (DDCMP), Ethernet, and FDDI	Ethernet and FDDI
NETCONFIG.COM procedure (part for specifying a router)	NETCONFIG.COM prompts you, "Do you want to operate as a router?"	NETCONFIG.COM does not prompt you. You have to enable Level 1 routing manually. Otherwise, NETCONFIG.COM is the same on OpenVMS Alpha systems.
Network management via Network Control Program (NCP) utility and the network management listener (NML) object		In many cases, the NCP commands and parameters are identical. However, the NCP command parameters for SET and SHOW operations for DDCMP, full host-based routing, the Distributed Name Service (DNS), and VAX P.S.I. have no effect on OpenVMS Alpha.
Product Authorization Key (PAK) name for cluster alias routing support	DVNETRTG	DVNETEXT
Routing	Level 1 and Level 2 routing are supported.	Level 1 routing is supported only on nodes acting as routers for a cluster alias. Level 2 routing is not supported.
VAX P.S.I.	Supported (except DEC 2000)	Not supported. For DECnet for OpenVMS AXP Version 1.5, DEC X.25 Client for OpenVMS AXP replaces VAX P.S.I. For DECnet/OSI, X.25 for OpenVMS AXP provides both client and native X.25 functionality.

For more information about the system management differences between DECnet for OpenVMS (Phase IV) on OpenVMS VAX and OpenVMS Alpha systems, see *A Comparison of System Management on OpenVMS AXP and OpenVMS VAX*.

2.3 DECnet/OSI Network Features

Digital's open DECnet/OSI network is a family of hardware and software products that allows Digital operating systems to communicate with each other and with systems produced by other vendors.

The DECnet/OSI network includes the following features:

- Remote system communication
- Resource sharing
- Support for distributed processing

- Distributed network management
- Optional use of distributed system services for networkwide names and synchronized time

Network users can access resources on any system in the network and the resources of other vendors' systems on multivendor networks.

DECnet/OSI for OpenVMS is Digital's implementation for OpenVMS systems of:

- The Open Systems Interconnection (OSI) communications specifications, as defined by the International Standards Organization (ISO).
- Digital's communications architecture, Digital Network Architecture (DNA) Phase V, which is also backward compatible with the Phase IV architecture.

Phase V integrates the DNA and OSI layers. The DNA Phase V Reference Model is the architectural model on which DECnet/OSI networking implementations are based.

Table 2–7 shows the changes that have evolved with each new phase.

Table 2–7 DNA Phases

Phase I	Limited to two nodes
Phase II	Up to 32 nodes: file transfer, remote file access, task-to-task programming interfaces, network management
Phase III	Up to 255 nodes: adaptive routing, downline loading, record access
Phase IV	Up to 64,449 nodes: Ethernet local area networks, area routing, host services, VMScluster support
Phase V	Virtually unlimited number of systems: OSI protocol support, transparent transport level links to TCP/IP, multivendor networking, local or distributed name service, distributed network management

DECnet/OSI for OpenVMS provides the integration of DECnet and OSI network protocols that continues support for DECnet applications and enables support for OSI applications on OpenVMS. With a separate TCP/IP running on the same system, DECnet/OSI supports a multivendor, multiprotocol network environment. DECnet applications can be run over NSP, CLNP, or TCP/IP transports. OSI applications can be run over CLNP or TCP/IP transports.

A full implementation of the Digital Network Architecture (DNA) Phase V for OpenVMS systems, the OSI component of the DECnet/OSI software is implemented and tested in accordance with the current U.S. and UK GOSIP requirements. GOSIP is the Government OSI Profile that defines OSI capabilities required by government procurement.

For more information about DECnet/OSI, see the DECnet/OSI documentation.

2.4 Interoperability in a VMScluster System

VMScluster systems for OpenVMS Alpha offer all the software features of VAXcluster systems. All the related VAXcluster software products, including the Business Recovery Server (BRS), are available for OpenVMS Alpha nodes. BRS, formerly named the Multi-Datacenter Facility, was released in June 1994.

In addition, VMScluster systems running OpenVMS Alpha Version 6.2 support the Small Computer System Interface (SCSI) interconnect as a multihost shared storage interconnect, a feature not available on OpenVMS VAX.

For configurations with VAX and Alpha nodes running the same version or different versions, additional capabilities and restrictions exist, as described in this section.

2.4.1 Booting in a Mixed-Architecture VMScluster System

The capabilities and restrictions of booting in a mixed-architecture VMScluster system are shown in Table 2–8.

Table 2–8 Booting in a Mixed-Architecture VMScluster System

Function	Description
System disk	Separate system disk required for VAX and Alpha
Cross-architecture satellite booting (VAX boot node provides booting services to Alpha satellites and Alpha boot node provides booting services to VAX satellites)	Enabled for systems running Version 6.1 or later of OpenVMS and DECnet for OpenVMS (Phase IV). Satellite booting across architectures is under investigation for nodes running DECnet/OSI.

2.4.2 Upgrades in a Mixed-Architecture VMScluster System

Rolling upgrades in a mixed-architecture VMScluster system are performed in the same way that rolling upgrades in a single-architecture VMScluster or VAXcluster system are performed. Cross-architecture upgrading is not enabled.

2.4.3 Restrictions of Selected Features in Mixed-Version VMScluster Systems

New features or improvements to existing features are introduced with each new release. Even though a feature or improvement is available on both platforms, its use may be restricted in a mixed-version or mixed-architecture VMScluster, if it was not available in earlier versions.

2.4.3.1 Process Identifiers Limit

Starting with Version 6.1, OpenVMS Alpha and OpenVMS VAX support up to 1024 identifiers. Previous versions of OpenVMS Alpha and OpenVMS VAX support up to 256 identifiers. If any nodes in a VMScluster system are running versions prior to Version 6.1, the limit of 256 identifiers is in effect.

2.4.3.2 Virtual I/O Cache

Virtual I/O cache became available in OpenVMS Alpha Version 1.5 running standalone and OpenVMS VAX Version 6.0. If any nodes in a mixed-version or mixed-architecture VMScluster system are running OpenVMS VAX Version 5.5–2 or OpenVMS Alpha Version 1.5, the virtual I/O cache disables itself.

2.4.3.3 Remote Monitoring

Remote monitoring is a feature of the Monitor utility (MONITOR) that enables you to monitor any node in a VMScluster system. You can do this either by issuing the MONITOR CLUSTER command or by adding the /NODE qualifier to any interactive MONITOR request.

Remote monitoring is limited across nodes running different versions of OpenVMS. Table 2–9 shows which versions enable this feature and which do not.

In addition, a second difference exists when you are monitoring remote nodes in a VMScluster. The limit on the number of disks that can be monitored remotely on OpenVMS VAX Version 6.2 and OpenVMS Alpha Version 6.2 was raised from 799 to 909 for record output and from 799 to 1817 for display and summary outputs. If you are monitoring a remote node running OpenVMS Version 6.2 from a system running an earlier version of OpenVMS, its limit of 799 is in effect.

Table 2–9 Remote Monitoring Compatibility in a VMScluster

	OpenVMS VAX Version 6.*n*	OpenVMS Alpha Version 6.*n*	OpenVMS Alpha Version 1.5 & VAX Version 5.*n*
OpenVMS VAX Version 6.*n*	Yes	Yes	No
OpenVMS Alpha Version 6.*n*	Yes	Yes	No
OpenVMS Alpha Version 1.5 and VAX Version 5.*n*	No	No	Yes

If you attempt to monitor a remote node that is incompatible, the following message is displayed:

```
%MONITOR-E-SRVMISMATCH, MONITOR server on remote node is an incompatible
version
```

If you receive this error message, you can still obtain data about the remote node with MONITOR. You do this by recording the data on the remote node and then using the MONITOR playback feature to examine it on the local node.

For more information on MONITOR, see the *OpenVMS System Management Utilities Reference Manual*.

2.4.4 ANALYZE/ERROR and ANALYZE/IMAGE in a Mixed-Architecture VMScluster System

The ANALYZE/ERROR (ERF) utility is architecture-specific. In other words, in a mixed-architecture VMScluster, a user on a VAX node cannot use ANALYZE /ERROR to analyze the ERRLOG.SYS of an Alpha node, nor can a user on an Alpha node use ANALYZE/ERROR to analyze the ERRLOG.SYS of a VAX node.

Support for using ANALYZE/ERROR across architectures is under investigation.

On Alpha, using ANALYZE/IMAGE, you can analyze both Alpha and VAX images. On VAX, you can only analyze VAX images. If you attempt to analyze an Alpha image from an OpenVMS VAX system, an unclear message is displayed.

A correction to the message is planned for a future release of OpenVMS VAX. A change to the OpenVMS VAX functionality is under investigation.

2.4.5 VMScluster Configuration Support

OpenVMS Alpha Version 6.2 and OpenVMS VAX Version 6.2 provide two levels of support for mixed-version and mixed-architecture VMSclusters: warranted support and migration support.

Warranted support means that Digital has fully qualified the two versions to coexist in a VMScluster, and will answer all problems identified by customers using these configurations.

Migration support is a superset of the rolling upgrade support provided in earlier releases of OpenVMS and is available for mixes that are not warranted. Migration support means that Digital has qualified the versions for use together in configurations that are migrating in a staged fashion to a newer version of OpenVMS VAX or to OpenVMS Alpha. Digital will answer problem reports submitted against these configurations. However, in exceptional cases, Digital may request that you move to a warranted configuration as part of answering the problem. Migration support will help customers move to warranted VMScluster version mixes with minimal impact on their cluster environments.

Figure 2–1 shows which level of support is provided for all possible version pairings.

Figure 2–1 VMScluster Version Pairings

	AXP Version 6.1	*VAX Version 6.1*	*VAX Version 6.0*	*AXP Version 1.5*	*VAX Version 5.5–2*	*Alpha Version 6.2*
VAX Version 6.2	Migration	Migration	Migration	Migration	Migration	WARRANTED
Alpha Version 6.2	Migration	Migration	Migration	Migration	Migration	

ZK–7496A–GE

Note that Digital does not support the use of more than two versions in a VMScluster at a time. In many cases, more than two versions will successfully operate, but Digital cannot commit to resolving problems experienced with such configurations.

For more VMScluster information, see the documentation listed in Table 2–10.

Table 2–10 VMScluster Documentation Sources

Topic	Documentation
Configuration rules and restrictions	OpenVMS Cluster Software Product Description
Configuration guidelines	*Guidelines for VMScluster Configurations*
Differences between managing VAXclusters and VMSclusters	*A Comparison of System Management on OpenVMS AXP and OpenVMS VAX*
SCSI VMSclusters	*OpenVMS Version 6.2 New Features Manual*
SCSI tape support	*OpenVMS Version 6.2 New Features Manual*
VMScluster systems that span multiple sites	*OpenVMS Version 6.2 New Features Manual*
All other VMScluster information	*VMScluster Systems for OpenVMS* and the *OpenVMS Version 6.2 Release Notes*

3
Migration When You're Ready

This chapter provides information that will help you decide when to include one or more OpenVMS Alpha systems in your computing environment and which migration resources to use. The following topics are discussed:

- OpenVMS Alpha base operating system features
- Digital optional software products for OpenVMS Alpha
- Third-party applications for OpenVMS Alpha
- Application migration paths
- Hardware and software investment protection program
- Migration services, training, software, and documentation
- Alpha systems on the Internet

Most of the OpenVMS VAX features and many of the Digital optional software products and third-party applications are already available on OpenVMS Alpha.

To help protect your investment in Digital products, Digital offers a single uniform upgrade program known as the ADVANTAGE-UPGRADE program. Digital also provides a full range of migration resources so that you can select what is appropriate for your organization. The offerings include an array of migration services, training, software, documentation, and access to Alpha systems on the Internet.

3.1 OpenVMS Alpha Base Operating System Features

Table 3–1 shows the availability, on OpenVMS Alpha, of selected features of the OpenVMS VAX base operating system. The list is not complete; it represents the features that are most important to Digital customers. For more information, contact your Digital account representative or authorized reseller.

Table 3–1 Status of Selected OpenVMS VAX Base Operating System Features on OpenVMS Alpha

Feature	OpenVMS Alpha Version 6.2
Adaptive pool management[1]	Yes
Batch and print queuing system[2]	Yes
Class scheduler system services[1]	Yes

[1]OpenVMS VAX Version 6.0 feature.

[2]OpenVMS VAX Version 5.5 feature.

(continued on next page)

Table 3–1 (Cont.) Status of Selected OpenVMS VAX Base Operating System Features on OpenVMS Alpha

Feature	OpenVMS Alpha Version 6.2
C2 Security[1]	Yes[3]
DECdtm full names support	Yes
DECthreads[2]	Yes
Debugger	Yes
EMA base system support	Yes
Extended LAT integration[2]	Yes
DECnet/OSI full names support[4]	Yes
Heap Analyzer in the OpenVMS Debugger[4]	No[5]
Help Message[1]	Yes
InfoServer booting	Yes
ISO 9660 support[1]	Yes
LMF V1.1[2]	Yes
Media management enhancements (MME)[4]	Yes
MSCP dynamic load balancing[1]	No[6]
MoveFile function with enhancements[2]	Yes
Multiple queue manager support[1]	Yes
POLYCENTER Software Installation (PCSI) utility	Yes
SCSI-2 Tagged Command Queuing (TCQ)[7]	Yes
Snapshot facility[1]	No[8]
Symmetric multiprocessing (SMP)	Yes
TPU and EVE	Yes
Virtual I/O cache for standalone machines	Yes
Virtual I/O cache clusterwide	Yes
VMScluster support	Yes
VMScluster support: FDDI interconnect of Alpha systems	Yes
X-terminal support	Yes
User-written device driver support	Yes[9]

[1]OpenVMS VAX Version 6.0 feature.

[2]OpenVMS VAX Version 5.5 feature.

[3]C2 security features available but not yet evaluated by the U.S. government. Auditing DECnet connections is not yet supported on Alpha.

[4]OpenVMS VAX Version 6.1 feature.

[5]Under investigation for a future release.

[6]Planned for a future release.

[7]OpenVMS VAX Version 5.5-2H4 feature.

[8]Not planned for a future release.

[9]Step 2 drivers.

3.2 Digital Optional Software for OpenVMS Alpha

Many Digital optional software products were available on earlier versions of OpenVMS Alpha and many more are available now. The optional software product releases are cumulative. For example, the CD–ROM for Digital optional software products released in March 1995 contains all the optional products that have been released for OpenVMS Alpha.

Note

Optional software products that run on OpenVMS AXP Version 1.5 or earlier that contain device drivers will not run on OpenVMS AXP Versions 6.1 or OpenVMS Alpha 6.2, because the device driver interface changed significantly. The device drivers in such applications must be revised. Although many OpenVMS Alpha products are available, the versions that can run on OpenVMS AXP Version 6.1 and OpenVMS Alpha Version 6.2 may not be available yet.

A Digital layered product CD–ROM is released concurrently with each new version of OpenVMS Alpha. All layered products on this CD–ROM run on the version of OpenVMS Alpha that is released at the same time. The CD–ROM includes all Digital OpenVMS Alpha software products (including products previously released that were upgraded to run on the latest version of OpenVMS Alpha).

Digital software products are released quarterly. All Digital optional software products that run on OpenVMS Alpha Version 6.2 and on OpenVMS VAX Version 6.2 are listed in the *OpenVMS Version 6.2 Release Notes*.

The *Alpha Applications Catalog* lists all available applications: Digital optional software and third-party applications. The catalog is updated regularly. To obtain this catalog in the United States and Canada, call 1-800-DIGITAL (1-800-344-4825). In other locations, contact your Digital account representative or authorized reseller.

To find out when products not listed in the catalog will be available, contact your Digital account representative or authorized reseller.

3.3 Third-Party Applications for OpenVMS Alpha

More than 2000 third-party applications are currently available for OpenVMS Alpha. Some of them are shown in Table 3–2.

The *Alpha Applications Catalog*, which is updated regularly, lists all the available third-party applications and Digital optional software products. The catalog is available in printed form and also online on the World-Wide Web (WWW).

For a printed copy, in the United States and Canada, call 1-800-DIGITAL (1-800-344-4825). In other locations, you can obtain the catalog from your Digital account representative or authorized reseller. To view it on the World-Wide Web, use the following command:

```
http://www.digital.com/cgi-bin/www-swdev/PRODUCTS/CATALOG/catalog
```

To find out when products not listed in the catalog will be available, contact your Digital account representative or authorized reseller.

Table 3–2 Sampling of Third-Party Applications Available as of June 1994

Application	Company
ACUMATE	Kenan Technologies
ADABAS	Software AG of North America, Inc.
ade EKO	Adedata AB.
ade INV	Adedata AB.
ade T/D	Adedata AB.
ADINA	ADINA R&D, INC.
ALK-GIAP	AED Graphics GmbH
Anvil 5000	Manufacturing and Consulting Services, Inc.
Application Browser	Hypersoft Corporation
ARC/INFO	Environmental Systems Research Institute, Inc.
ASPEN PLUS	Aspen Technology, Inc.
Auto. Library System	Dynix, Inc.
BEV-PAK	Turn-Key Distribution Systems, Inc.
Blood Bank Systems	Antrim Corporation
BLAST	BLAST, Inc.
BROKERMAX	Citymax Integrated Information System Ltd.
Business Intelligence Network (BIN)	Henco Software, Inc.
Cadim/EDB	EIGNER + PARTNER GmbH
CADRA-III	ADRA Systems, Inc.
Client Server Interfaces	Sybase, Inc.
DADisp	DSP Development Corporation
DL Pager	Datalogics, Inc.
ECLIPSE	Intera Information Technologies
Financial Systems	Antrim Corporation
FlexiLab System	Sunquest Information Systems, Inc.
FlexiRad	Sunquest Information Systems, Inc.
FOCUS for Alpha	Information Builders, Inc.
Gembase Open Systems 4GL	Ross Systems, Inc.
GENSTAT 5	Numerical Algorithms Group, Ltd.
GLIM	Numerical Algorithms Group, Ltd.
GRAFkit	Centera Information Systems, Inc.
Graphics Language Interpreter	Centera Information Systems, Inc.
IAS	CODA, Incorporated
IBS-90	Winter Partners
IMPALA	EEC SYSTEMS, INC.

(continued on next page)

Table 3–2 (Cont.) Sampling of Third-Party Applications Available as of June 1994

Application	Company
Information Support Systems	Antrim Corporation
"Integra" Application Management Environment	G.C. McKeown & Co. (UK) Ltd.
Integrated Graphics System	Datalogics, Inc.
Integration Services	Antrim Corporation
IPS	CODA, Incorporated
MANMAN (including Process)	ASK Computer Services, Inc.
MANTIS	Cincom Systems, Inc.
MAPS	Logica Industry Limited
MARGO	SEMA GROUP-PROGICIELS
Mathematica	Wolfram Research, Inc.
MESSIDOR	SEMA GROUP-PROGICIELS
Mobile	KINGSTON_SCL LTD.
Multinet	TGV, Inc.
NAG FORTRAN Library	Numerical Algorithms Group, Ltd.
NAG Graphics Library	Numerical Algorithms Group, Ltd.
NAGWave Fortran 90 Compiler	Numerical Algorithms Group, Ltd.
NATURAL	Software AG of North America, Inc.
NETRON/CAP	NETRON, Inc.
NETRON/Client	NETRON, Inc.
Oasys 680x0 Cross Tools	Oasys
ORACLE V7 (Developer's release)	Oracle Corporation
ORACLE V7 (Production release)	Oracle Corporation
Oracle Financials and Human Resources	Oracle Corporation
Oracle Manufacturing	Oracle Corporation
Page Station/X	Datalogics, Inc.
PixTex/EFS	Excalibur Technology
PLEIADES HOSPITAL ADMINISTRATION	SEMA GROUP-PROGICIELS
PLEIADES HRM/PRIVATE AND PUBLIC SECTORS	SEMA GROUP-PROGICIELS
PLEIADES LOCAL COMMUNITIES	SEMA GROUP-PROGICIELS
Polyserver	Uniface Int.
PowerHouse 4GL	Cognos, Inc.
PROGRESS 4GL/RDBMS	Progress Software Corporation
PROGRESS Application and Development Environment	Progress Software Corporation
PROGRESS Developer's Toolkit	Progress Software Corporation

(continued on next page)

Table 3–2 (Cont.) Sampling of Third-Party Applications Available as of June 1994

Application	Company
PROGRESS FAST TRACK	Progress Software Corporation
PROGRESS RESULTS Query/Reporting Tool	Progress Software Corporation
PROMIS	Promis Systems Corporation
Promix Distribution Series	Ross Systems, Inc.
Promix Manufacturing Series	Ross Systems, Inc.
Renaissance CS Financial Series	Ross Systems, Inc.
Renaissance CS Human Resource Series	Ross Systems, Inc.
RSC-PR Payroll	Resource Systems Corporation
SAP R/3 System	SAP of America, Inc.
SAS System	SAS Institute, Inc.
SmartStar Report Painter	SmartSystems (UK) Ltd.
SmartStar Vision	SmartStar Corporation
STADEN Package	Medical Research Council
STUDENTS+	J.H. Leskin Associates, Inc.
Supercache	EEC Systems, Inc.
SuperDisk	EEC Systems, Inc.
SUPRA	Cincom Systems, Inc.
Sybase Lifecycle Tools	Sybase, Inc.
Sybase SQL Server	Sybase, Inc.
Synchrony	Henco Software, Inc.
TCM-EMS Time Critical Manufacturing System V	Effective Management Systems, Inc. (EMS)
TGRAF-X	Grafpoint, Inc.
Timeserver	Pilot Software Ltd.
TROPOS	Strategic Systems International
Unidata RDBMS	Unidata, Inc.
Uniface Development Environment	Uniface and Uniface Int.
UNIGRAPHICS	Electronic Data Systems Corporation
VAX TDMS Emulator for OpenVMS	Praxa Limited
VAX TDMS to DECforms Converter for OpenVMS VAX	Praxa Limited
WIN/TCP	Wollongong
VIDEOTELEFAX	Monaco Telematique MC-TEL
VIDEONET	Monaco Telematique MC-TEL
Wisconsin Sequence Analysis Package	Genetics Computer Group
WITNESS (U.S.)	AT&T ISTEL

3.4 Application Migration Paths to OpenVMS Alpha

Table 3–3 shows several paths for application migration from OpenVMS VAX to OpenVMS Alpha systems.

Table 3–3 Application Migration Paths

	AXP V1.0	AXP V1.5	AXP V6.1	Alpha V6.2
VAX V5.4–2	Recommended	Recommended	Supported[1]	Supported[1]
VAX V5.4–3	Recommended	Recommended	Supported[1]	Supported[1]
VAX V5.5, V5.5–1	Supported[1]	Recommended	Supported[1]	Supported[1]
VAX V5.5–2	Supported[1]	Recommended	Recommended	Recommended
VAX V6.0	Not Recommended	Supported[1]	Recommended	Recommended
VAX V6.1	Not Recommended	Not Recommended	Recommended	Recommended
VAX V6.2	Not Recommended	Not Recommended	Not Recommended	Recommended

[1] Supported means that most applications will run, but applications that depend on features that are not available either in a particular version of OpenVMS Alpha or in an OpenVMS Alpha compiler will not run or will not run correctly.

3.5 Hardware and Software Investment Protection Programs

Digital offers hardware and software investment protection in a single, uniform upgrade program known as the ADVANTAGE-UPGRADE Program. By offering all of Digital's upgrade selections under a single program, selection of the proper and most cost-effective upgrade has become simpler.

While the names have changed, the upgrade features remain the same. For example, you can:

- Lock in a not-to-exceed price for upgrading to Alpha systems (within a specified period of time) when you purchase new VAX or MIPS computers.
- Purchase a simple Alpha upgrade from older VAX computers or use the trade-in credit from your existing VAX computers.
- Trade in the original operating system licenses for credit towards an alternative operating system that will run on the same computers. This gives you the flexibility of switching to another operating system at a later date.

For Digital layered software products, user-based licenses are valid across hardware architectures so no special program has been introduced. New clusterwide and capacity-based licenses are significantly discounted. For more information, contact your Digital account representative or authorized reseller.

3.6 Migration Services

The ISV (independent software vendor) Engineering and Technical Support Centers provide migration services worldwide, as shown in Table 3–4. Migration experts staff the centers and assist Digital's business partners with their porting efforts.

For detailed information on migration services, contact your Digital account representative or authorized reseller, or call 1-800-832-6277 or 1-603-884-8990.

Table 3–4 Locations of Engineering and Technical Support Centers

Europe	United States	Asia
Basingstoke	Marlboro MA	Hong Kong
Galway	Palo Alto CA	Tokyo
Munich		

3.7 Migration Training

Digital offers a two-day seminar, the Alpha AXP Planning seminar (EY-L570E-S0-W3), which presents issues in planning for systems based on the Alpha architecture. You can learn more about this seminar by contacting your Digital account representative or authorized reseller or by reading about it in the Digital Learning Services (DLS) Internet Catalog. The DLS Internet Catalog provides course descriptions, the course schedule, and directions for registering. To access the DLS Internet Catalog on the World-Wide Web, use the following command:

```
http://www.digital.com/.i/digest/htdocs/digest/html/dis.ht
```

3.8 Migration Software

Compilers are available on OpenVMS Alpha for almost all the languages supported on OpenVMS VAX. Most programs can be recompiled and relinked for execution, in native mode, on OpenVMS Alpha. For more information, see Chapter 4.

In addition to the OpenVMS Alpha compilers, Digital also offers DECmigrate for OpenVMS AXP, a Digital optional software product. DECmigrate is used for the following purposes:

- To analyze code to determine how easy or difficult it might be to translate it
- To translate images for which you have no sources or whose native compiler is not yet available on OpenVMS Alpha systems

For more information about DECmigrate, see Section 4.2.3.

3.8.1 Mixing Native Alpha and Translated Images

You can mix migration methods among the individual images that comprise an application, that is, you can recompile some modules with the native OpenVMS Alpha compilers and translate other modules with DECmigrate. You can also partially translate an application as one stage in a migration. This enables you to run and test an application on an Alpha computer before it is completely recompiled.

For more information about interoperability of native Alpha and translated VAX images within an application, see *Migrating to an OpenVMS AXP System: Recompiling and Relinking Applications*.

3.9 Migration Documentation

Digital offers several documents for migration from OpenVMS VAX to OpenVMS Alpha. Migration information is also provided in the language user's guides for the DEC compilers. The differences in the DEC compilers between OpenVMS Alpha and OpenVMS VAX systems are described in the DEC compilers' users' guides. In some guides, such as the *DEC C User's Guide for OpenVMS Systems*, the differences are described in the context of the description of a language element; in other guides, such as the *DEC COBOL User Manual*, the differences are described in separate appendixes.

The following list describes the migration manuals and includes their order numbers.

- *Migrating to an OpenVMS AXP System: Planning for Migration*

  Order number: AA-PV62A-TE

  This manual describes the general characteristics of RISC architectures, compares the Alpha architecture to the VAX architecture, and presents an overview of the migration process and a summary of migration tools provided by Digital. The information in this manual is intended to help you define the optimal migration strategy for your application.

- *Migrating to an OpenVMS AXP System: Recompiling and Relinking Applications*

  Order number: AA-PV63A-TE

  This manual provides detailed technical information for programmers who must migrate mid- and high-level language applications to OpenVMS Alpha systems. It describes how to set up a development environment to facilitate the migration of applications, helps programmers identify application dependencies on elements of the VAX architecture, and introduces compiler features that help resolve these dependencies. Individual sections of this manual discuss specific application dependencies on VAX architectural features, data porting issues (such as alignment concerns), and the process of migrating VAX shareable images.

- *Migrating to an OpenVMS AXP System: Porting VAX MACRO Code*

  Order number: AA-PV64A-TE

  This manual describes how to use the MACRO–32 compiler for OpenVMS Alpha to port VAX MACRO code to an OpenVMS Alpha system. It describes the features of the compiler, presents a methodology for porting VAX MACRO code, identifies nonportable coding practices, and recommends alternatives to such practices. The manual also provides detailed descriptions of the compiler qualifiers, directives, built-ins, and the system macros created for porting to an OpenVMS Alpha system.

- *A Comparison of System Management on OpenVMS AXP and OpenVMS VAX*

  Order number: AA-PV71B-TE

  This manual compares system management on OpenVMS Alpha and OpenVMS VAX systems. It is intended for experienced system managers who need to learn quickly how specific tasks differ or remain the same on OpenVMS Alpha and OpenVMS VAX.

- *DECmigrate for OpenVMS AXP Systems Translating Images*

  Order number: AA-PSGMB-TE

This manual describes the VAX Environment Software Translator (VEST) utility, discussed in Section 3.8. It describes how to use VEST to convert most user-mode OpenVMS VAX images to translated images that can run on OpenVMS Alpha systems; how to improve the run-time performance of translated images; how to use VEST to trace OpenVMS Alpha incompatibilities in an OpenVMS VAX image back to the original source files; and how to use VEST to support compatibility among native and translated run-time libraries.

- *Creating an OpenVMS AXP Step 2 Device Driver from a Step 1 Device Driver*

  Order number: AA-Q28TA-TE

  This manual describes how to convert a Step 1 OpenVMS AXP device driver, written in VAX MACRO, to a Step 2 OpenVMS Alpha device driver, also written in VAX MACRO.

- *Creating an OpenVMS AXP Step 2 Device Driver from an OpenVMS VAX Device Driver*

  Order number: AA-Q28UA-TE

  This manual describes how to convert an OpenVMS VAX device driver—written in VAX MACRO—to a Step 2 OpenVMS Alpha device driver, also written in VAX MACRO.

- *OpenVMS AXP Device Support: Reference*

  Order number: AA-Q28PA-TE

  This manual provides reference material for creating OpenVMS Alpha device drivers, and it describes the macros, system routines, and entry points used in converting OpenVMS VAX and Step 1 OpenVMS AXP device drivers to Step 2 OpenVMS Alpha device drivers.

3.9.1 Obtaining Migration Documentation

When you purchase an OpenVMS Alpha media kit, you receive the migration documentation in Bookreader format (DECW$BOOK) on the compact disc. When you purchase the printed full OpenVMS documentation set, the migration documentation is included. You can also order any of the manuals separately. *DECmigrate for OpenVMS AXP Systems Translating Images* in Bookreader format accompanies the optional product, DECmigrate for OpenVMS AXP. To obtain a printed copy, you must order it separately.

For instructions on ordering documentation, see *How to Order Additional Documentation* in the front of this manual.

3.10 Alpha Systems on the Internet

As a service to the Internet community, Digital has made available two DEC 4000 Alpha systems. These systems can be used for evaluating the Alpha architecture and for testing the features of the supporting OpenVMS Alpha and DEC OSF/1 for Alpha operating systems, compilers, tools, and utilities.

Application developers who have access to the Internet can use these systems to test, qualify, or port their software for the Alpha architecture. Other Internet users interested in Alpha computing are invited to log in and evaluate these systems.

One DEC 4000 Alpha system (Internet address: axpvms.pa.dec.com) has the OpenVMS operating system installed. The other DEC 4000 Alpha system (Internet address: axposf.pa.dec.com) is running the Digital UNIX (formerly named DEC OSF/1) operating system. These systems can be reached either via telnet or rlogin.

To register for an account, Internet users connect to the desired machine, log in as axpguest (no password), and answer the short qualifying questionnaire. Users are asked to read all information in the motd/login banner and comply with all rules for machine usage/restrictions.

Users with questions about their accounts should send mail to Internet address: axpvms-system@pa.dec.com for the OpenVMS Alpha system and to Internet address: axposf-root@pa.dec.com for the Digital UNIX system.

4

Ensuring the Portability of Applications

This chapter describes:

- How to assess the portability of an application
- Software support for portability
- Differences in OpenVMS Alpha programming
- Guidelines for developing applications for OpenVMS VAX and OpenVMS Alpha
- Guidelines for developing applications for mixed-architecture VMScluster systems

In general, if your application is written in a high-level programming language, you should be able to run it on an Alpha system with a minimum amount of effort. High-level languages insulate applications from dependence on the underlying machine architecture, and, for the most part, the programming environment on Alpha systems duplicates the programming environment on VAX systems. Using native Alpha versions of the language compilers and the Linker utility (linker), you can recompile and relink the source files that make up your application to produce a native Alpha image.

If your application is written in VAX MACRO, you may be able to run it on an Alpha system with a minimum amount of effort, although it is more likely to contain some dependencies on the underlying VAX architecture, some of which may require your intervention.

4.1 How to Assess the Portability of an Application

The portability of an application depends on the language in which it is written, the amount of nonstandard code it might contain, the number of architectural dependencies it might contain, and whether a compiler is available for the language in which the application is written. While it is possible to introduce architectural dependencies in applications written in high-level languages, they are more likely to occur in applications written in mid- and low-level languages.

Privileged applications, which run in inner modes or at elevated interrupt priority levels (IPLs), may require significant changes because of assumptions incorporated in the code about the internal operation of the operating system. Typically, such applications also require significant changes after a major release of the OpenVMS VAX operating system.

Recently, Digital introduced new versions of several compilers. It is likely that the applications that you might want to move to an OpenVMS Alpha system were compiled using the earlier VAX compilers.

To assess the portability of an application, consider the following:

- The application's dependencies on the VAX architecture

- The differences between the VAX and DEC language compilers

To help you in your assessment, you can use DECmigrate for OpenVMS AXP and the compiler features designed to identify potential porting problems.

You may also need to identify nonstandard coding practices. They are generally more common in code written in lower-level languages, such as VAX MACRO. For information about such practices for VAX MACRO, refer to *Migrating to an OpenVMS AXP System: Porting VAX MACRO Code.*

4.1.1 Identifying Dependencies on the VAX Architecture in Your Application

Even if your application recompiles successfully with a compiler that generates native Alpha code, it may still contain subtle dependencies on the VAX architecture. The OpenVMS Alpha operating system has been designed to provide a high degree of compatibility with OpenVMS VAX; however, the fundamental differences between the VAX and Alpha architectures can create problems for applications that depend on certain VAX architectural features. The following list highlights those areas of your application you should examine.

- Check the data declarations contained in your application.

  The high-level language data types you selected to represent data items on an OpenVMS VAX system may not be the best choice on an OpenVMS Alpha system. In particular, consider the following:

  - **Data packing**—Applications on VAX systems typically use the smallest available data type to represent a data item to achieve efficient use of memory resources. For various reasons, using larger data types may be more efficient on OpenVMS Alpha systems. For example, unaligned data can take 100 times longer to process than aligned.
  - **Data-type selection**—The Alpha architecture supports most of the VAX native data types; however, certain VAX data types, such as the H_float floating-point data type, are not supported (see Table 4–1). Check to see if your application depends on the size or bit representation of an underlying native data type.
  - **Shared access to data**—Check any writable data item that is accessed by multiple threads of execution. The VAX architecture includes instructions that can perform certain complex operations, such as incrementing a variable, that appear as a single, noninterruptable operation to other threads of execution. The Alpha architecture is a load-store architecture that does not support atomic memory-to-memory modifications so different program constructs may be required.

    In addition, the VAX architecture supports instructions that can manipulate byte- and word-sized data in a single noninterruptable operation. The Alpha architecture supports noninterruptable access only to aligned longword- or aligned quadword-sized data.
  - **Buffer size**—Your application may determine the size of certain data buffers based on the VAX page size. Different implementations of the Alpha architecture can support 8K, 16K, 32K, or 64K byte pages. Search your application for the text strings “512” and “511” (or the hexadecimal equivalents, “200” and “1FF”) to find dependencies on the VAX page size.

- Check any condition handlers your application may include.

While the condition handling facility on OpenVMS Alpha systems is functionally equivalent to the VAX condition handling facility, certain aspects of the facility have changed, such as the format of the mechanism array. In addition, the way in which arithmetic exceptions are reported has changed.

- Check for dependence on the AST parameter list.

  While the AST parameter list on OpenVMS Alpha systems has the same format as on VAX systems, only the AST parameter field can be used. The other fields in the AST parameter list (contents of R0, R1, program counter [PC], and processor status [PS]) are provided for compatibility only and have no subsequent use after the AST procedure exits. For example, on OpenVMS VAX systems, some user-written AST procedures are designed to change one or more of the values in the other fields in the AST parameter list so that these new values take effect upon completion of the AST procedure. Because ASTs are handled differently on OpenVMS Alpha systems, such changes by the AST procedure to the other fields in the AST parameter list have no effect. Anything an AST procedure writes to the last four parameters on an Alpha computer is lost when the AST procedure exits.

Table 4–1 Floating-Point Data Type Support

Data Type	On VAX	On Alpha
D53_float (G_ float) (Default double-precision format)	Not supported.	Supported. Using D53_float instead of D56_float drops three bits of precision and yields slightly different results.
D56_ float (Default double-precision format)	Supported.	Not supported. You can obtain full support by translating your code with DECmigrate. Alternatively, you can substitute D53_float for D56_float, if your application does not require the extra three bits of precision.
F_float	Supported.	Supported.
G_float	Supported.	Supported.
H_float (128-bit floating-point)	Supported.	Not supported. You can obtain full H_float support with DECmigrate. You can use it to translate the code module that contains H_float structures, or you can recode your application, using a supported data type.
S_float (IEEE)	Not supported.	Supported.
T_float (IEEE)	Not supported.	Supported.
X_float (128-bit floating-point (IEEE))	Not supported.	Supported by DEC Fortran Version 6.2 and by DEC C Version 4.0. The X_float data format is not identical to H_float, but both cover a similar range of values. For Fortran applications, automatic conversion between X_float memory format and H_float on-disk is possible by use of the /CONVERT compiler qualifier, or the CONVERT= option on OPEN statements.

For more information about dependencies on the VAX architecture, see *Migrating to an OpenVMS AXP System: Recompiling and Relinking Applications* and the user's guides for the particular language you are using. For applications written in VAX MACRO, see *Migrating to an OpenVMS AXP System: Porting VAX MACRO Code.*

4.1.2 Compiler Differences

Compiler differences can exist for two reasons: differences between earlier and current versions of compilers running on OpenVMS VAX and differences between the DEC versions running on the VAX and Alpha computers. The OpenVMS Alpha compilers are intended to be compatible with their OpenVMS VAX counterparts. They include several qualifiers that contribute to compatibility, as described in Section 4.2.2.

The languages conform to language standards and include support for most OpenVMS VAX language extensions. The compilers produce output files with the same default file types as they do on OpenVMS VAX systems, such as .OBJ for an object module.

Note, however, that some features supported by the compilers on OpenVMS VAX systems may not be available OpenVMS Alpha systems. For example, the OpenVMS Alpha run-time libraries (RTLs) do not contain the routine LIB$ESTABLISH, which the OpenVMS VAX RTLs contain. Due to the nature of the OpenVMS Alpha calling standard, setting up condition handlers is done by compilers.

For those programs that need to dynamically establish condition handlers, some Alpha languages give special treatment for apparent calls to LIB$ESTABLISH and generate the appropriate code without actually calling an RTL routine. The following languages support LIB$ESTABLISH semantics in a compatible fashion with the corresponding VAX language:

- DEC C and DEC C++

  Although DEC C and DEC C++ for OpenVMS Alpha systems treat LIB$ESTABLISH as a built-in function, the use of LIB$ESTABLISH is not recommended on OpenVMS VAX or OpenVMS Alpha systems. C and C++ programmers should call VAXC$ESTABLISH instead of LIB$ESTABLISH (VAXC$ESTABLISH is a built-in function on DEC C and DEC C++ for OpenVMS Alpha systems).

- DEC Fortran

  DEC Fortran allows declarations to LIB$ESTABLISH and converts them to DEC Fortran RTL specific entry points.

- DEC Pascal

  DEC Pascal provides the built-in routines, ESTABLISH and REVERT, to use in place of LIB$ESTABLISH. If you declare and try to use LIB$ESTABLISH, you will get a compile-time warning.

- MACRO–32

  The MACRO–32 compiler will attempt to call LIB$ESTABLISH if it is contained in the source code.

  If MACRO–32 programs establish dynamic handlers by storing a routine address at 0(FP), they will work correctly when compiled on an OpenVMS Alpha system. However, you cannot set the condition handler address from within a JSB (Jump to Subroutine) routine, only from within a CALL_ENTRY routine.

If you are recompiling VAX C code, either an entire application or one or more modules, you will want to pay particular attention to any external symbols that it contains. Unlike the VAX C compiler which supports one external symbol model, the DEC C compiler supports four models. The default external symbol that is produced by the DEC C compiler is not the same as the single VAX C external symbol.

Furthermore, when you link such code, due to changes in the linker, if you did not specify the /SHARE qualifier when you recompiled the C code module, you will need to specify a related linker qualifier.

For more information about compiler differences between OpenVMS VAX and OpenVMS Alpha, refer to *Migrating to an OpenVMS AXP System: Recompiling and Relinking Applications*. For more information about the compiler differences for each language, refer to its documentation, especially the user's guides and the release notes. For more information about the linker, refer to Section 4.3.1.

4.2 Software Support for Portability

The OpenVMS Alpha operating system and many of the optional products it supports, such as the DEC compilers and DECmigrate for OpenVMS AXP, include features that contribute to portability. Some of the features are primarily diagnostic while others compensate for architectural dependencies.

4.2.1 VAX MACRO–32 Compiler for OpenVMS Alpha

The VAX MACRO–32 Compiler for OpenVMS Alpha is used to convert existing VAX MACRO code into machine code that runs on OpenVMS Alpha systems. It is included with OpenVMS Alpha for that purpose.

While some VAX MACRO code can be compiled without any changes, most code modules will require the addition of entry point directives. Many code modules will require other changes as well.

The compiler generates code that is optimized for OpenVMS Alpha systems, but many features of the VAX MACRO language that provide the programmer with a high level of control make it more difficult to generate optimal code for OpenVMS Alpha systems. For new program development for OpenVMS Alpha, Digital recommends the use of mid- and high-level languages. For more information on the MACRO–32 compiler, see *Migrating to an OpenVMS AXP System: Porting VAX MACRO Code*.

4.2.2 Compiler Support

The DEC compilers can produce messages that are very useful for identifying potential porting problems. For example, the MACRO–32 compiler provides the /FLAG qualifier with 10 options. Depending on which options you include, the compiler reports all unaligned stack and memory references, any run-time code generation (such as self-modifying code), branches between routines, or several other conditions.

The DEC Fortran compiler qualifier, /CHECK, produces messages about any of the various options you specify.

Some compilers on OpenVMS Alpha systems support new features not supported by their counterparts on OpenVMS VAX systems. To provide compatibility, some compilers support compatibility modes. For example, the DEC C compiler for OpenVMS Alpha systems supports a VAX C compatibility mode that is invoked by specifying the /STANDARD=VAXC qualifier.

In some cases, the compatibility is limited. For example, VAX C implements built-in functions that allow access to special VAX hardware features. Since the hardware architecture of VAX computers differs from Alpha computers, these built-ins are not available in DEC C for OpenVMS Alpha systems even when the /STANDARD=VAXC qualifier is used.

The compilers can also compensate for some architectural dependencies that may exist in your code. For example, the MACRO–32 compiler provides the /PRESERVE qualifier that can preserve granularity or atomicity or both.

The DEC C compiler provides a header file that defines macros for each data type. These macros map a generic data-type name, such as int64, to the machine-specific data type, such as *-64*. For example, if you must have a data type that is 64 bits long, use the int64 macro.

Review the documentation for your compiler to become familiar with all its features that support portability.

4.2.3 DECmigrate for OpenVMS AXP

DECmigrate for OpenVMS AXP is used to translate images for which the source code is not available. The VAX Environment Software Translator (VEST) component of DECmigrate translates the VAX binary image file into a native Alpha image. The translated image runs under the Translated Image Environment (TIE) on an Alpha computer. (TIE is a shareable image that is included with the OpenVMS Alpha operating system.) Translation does not involve running an OpenVMS VAX image under emulation or interpretation (with certain limited exceptions). Instead, the new OpenVMS Alpha image contains Alpha instructions that perform operations identical to those performed by the instructions in the original OpenVMS VAX image.

A translated image generally runs as fast on an Alpha computer as the original image runs on a VAX computer. However, a translated image does not benefit from the optimizing compilers that take full advantage of the Alpha architecture. Therefore, a translated image typically runs about 25% to 40% as fast as a native OpenVMS Alpha image. The primary causes for this reduced performance are unaligned data and the extensive use of complex VAX instructions.

DECmigrate translation support is limited to the language features, system services, and run-time library entry points that existed on OpenVMS VAX Version 5.5-2. This limitation and a method for overcoming it (in case your application uses features introduced after the OpenVMS VAX Version 5.5-2 release) are described in the *OpenVMS Version 6.2 Release Notes*.

A second function of DECmigrate is to analyze images to identify specific incompatibilities for an Alpha computer. Depending on the type of incompatibility, you can choose to specify a compiler qualifier that will compensate for the problem or make changes to the code.

For more information on image translation and VEST, see *DECmigrate for OpenVMS AXP Systems Translating Images*.

4.2.4 PALcode

The Alpha architecture does not favor a particular operating system. To accommodate different operating systems, it enables the creation of privileged architecture library code (PALcode).

Furthermore, certain OpenVMS Alpha compilers, such as C and the MACRO–32 compiler, provide PALcode built-ins that supplement the instructions available in the Alpha instruction set. For example, the MACRO–32 compiler provides built-ins that emulate those VAX instructions for which there are no Alpha equivalents and a built-in that enables you to write your own PALcode.

PALcode can be used to access internal hardware registers and physical memory. PALcode can provide direct correspondence of physical and virtual memory. For more information about PALcode, see the *Alpha Architecture Reference Manual*.

4.3 Differences in OpenVMS Alpha Programming

Differences in OpenVMS Alpha programming are discussed in this section.

4.3.1 Linker

Once you successfully recompile your source files, you must relink your application to create a native Alpha image. The linker produces output files with the same file types as on current VAX systems. For example, by default, the linker uses the file type .EXE to identify image files.

Because the way in which you perform certain linking tasks is different on OpenVMS Alpha systems, you will probably need to modify the LINK command used to build your application. The following list describes some of these linker changes that may affect your application's build procedure. See the *OpenVMS Linker Utility Manual* for more information.

- **Declaring universal symbols in shareable images**—If your application creates shareable images, your application build procedure probably includes a transfer vector file, written in VAX MACRO, in which you declare the universal symbols in the shareable image. On OpenVMS Alpha systems, instead of creating a transfer vector file, you must declare universal symbols in a linker options file by specifying the SYMBOL_VECTOR= option.
- **Linking against the OpenVMS executive**—On OpenVMS VAX systems, you link against the OpenVMS executive by including the system symbol table file (SYS.STB) in your build procedure. On OpenVMS Alpha systems, you link against the OpenVMS executive by specifying the /SYSEXE qualifier.
- **Optimizing the performance of images**—On OpenVMS Alpha systems, the linker can perform certain optimizations that can improve the performance of the images it creates. In addition, the linker can create shareable images that can be installed as resident images, which enhances performance.
- **Processing shareable images implicitly**—On OpenVMS VAX systems, when you specify a shareable image in a link operation, the linker also processes all the shareable images to which that shareable image was linked. On OpenVMS Alpha systems, to include these shareable images in your build procedure, you must explicitly specify them.

The linker supports several qualifiers and options, listed in Table 4–2, that are specific to OpenVMS Alpha systems. Some linker options, listed in Table 4–3, are not supported on OpenVMS Alpha systems.

Table 4–2 Linker Qualifiers and Options Specific to OpenVMS Alpha Systems

Qualifiers	Description
/DEMAND_ZERO	Controls how the linker creates demand-zero image sections.
/DSF	Directs the linker to create a file called a debug symbol file (DSF) for use by the OpenVMS Alpha System-Code Debugger.
/GST	Directs the linker to create a global symbol table (GST) for a shareable image (the default). More typically specified as /NOGST when used to create shareable images for run-time kits.
/INFORMATIONALS	Directs the linker to output informational messages during a link operation (the default). More typically specified as /NOINFORMATIONALS to suppress these messages.
/NATIVE_ONLY	Directs the linker to *not* pass along the procedure signature block (PSB) information, created by the compilers, in the image it is creating (the default). If /NONATIVE_ONLY is specified during linking, the image activator uses the PSB information, if any, provided in the object modules specified as input files to the link operation to invoke jacket routines. Jacket routines are necessary to allow native Alpha images to work with translated VAX images.
/REPLACE	Directs the linker to perform certain optimizations that can improve the performance of the image it is creating, when requested to do so by the compilers (the default).
/SECTION_BINDING	Directs the linker to create a shareable image that can be installed as a resident image.
/SYSEXE	Directs the linker to process the OpenVMS executive image (SYS$BASE_IMAGE.EXE) to resolve symbols left unresolved in a link operation.

Options	Description
SYMBOL_TABLE= option	Directs the linker to include global symbols as well as universal symbols in the symbol table file associated with a shareable image. By default, the linker includes only universal symbols.
SYMBOL_VECTOR= option	Used to declare universal symbols in Alpha shareable images.

Table 4–3 Linker Options Specific to OpenVMS VAX Systems

Options	Description
BASE= option	Specifies the base address (starting address) that you want the linker to assign to the image.
DZRO_MIN= option	Specifies the minimum number of contiguous, uninitialized pages that the linker must find in an image section before it can extract the pages from the image section and place them in a newly created demand-zero image section. By creating demand-zero image sections (image sections that do not contain initialized data), the linker can reduce the size of images.
ISD_MAX= option	Specifies the maximum number of image sections allowed in the image.
UNIVERSAL= option	Declares a symbol in a shareable image as universal, causing the linker to include it in the global symbol table of a shareable image.

4.3.2 MACRO–64 Assembler for OpenVMS Alpha Systems

The MACRO–64 assembler for OpenVMS Alpha systems is the native assembler for all Alpha computers. Unlike the VAX MACRO assembler which is included with the OpenVMS VAX operating system, the MACRO–64 assembler is not included with the OpenVMS Alpha operating system. It can be purchased separately. In general, the mid- and high-level language compilers generate higher performance code for OpenVMS Alpha systems than the MACRO–64 assembler. Therefore, Digital recommends you use mid- and high-level compilers for new program development for OpenVMS Alpha systems. For more information about the MACRO–64 assembler, see *MACRO–64 Assembler for OpenVMS AXP Systems Reference Manual*.

4.3.3 User-Written Device Drivers

Formal support for user-written device drivers and a new interface known as the Step 2 driver interface were introduced in OpenVMS Alpha Version 6.1. The Step 2 driver interface supports user-written device drivers in the C programming language. It replaced the temporary Step 1 driver interface that was provided in OpenVMS Alpha Versions 1.0 and 1.5.

There is no formal support for writing OpenVMS VAX device drivers in C. For example, OpenVMS VAX does not provide .h files for internal VMS (lib) data structures.

The Step 2 driver interface has increased the differences between OpenVMS Alpha and OpenVMS VAX device drivers. Device driver source files written in VAX MACRO or Bliss can be kept common between OpenVMS Alpha and VAX through the use of conditional compilation and user-written macros.

The advisability of this approach depends greatly on the nature of the individual driver. It is likely that in future versions of OpenVMS Alpha, the I/O subsystem will continue to evolve in directions that will have an impact on device drivers. This could increase the differences between OpenVMS Alpha and VAX device drivers and add more complexity to common driver sources. For this reason, a fully common driver source file approach might not be advisable for the long term.

Depending on the individual driver, it might be advisable to partition the driver into a common module and an architecture-specific one. For example, if one were writing a device driver that does disk compression, then the compression algorithm could readily be isolated into an architecture independent module. One could also avoid operating-system-specific data structures in such common modules with the intent of having some common modules across various types of operating systems; for example, OpenVMS, Windows NT, and OSF.

For more information about writing OpenVMS Alpha device drivers in C, see the *OpenVMS AXP Device Support: Developer's Guide*.

4.3.4 OpenVMS Debugger

On OpenVMS Alpha systems you can use the debugger with programs written in the following DEC languages:

- Ada
- BASIC
- C
- C++
- COBOL
- Fortran
- MACRO–32 (compiled with the MACRO-32 compiler)
- MACRO–64
- Pascal
- DEC PL/I

The OpenVMS Debugger includes several features that address the architectural differences of OpenVMS Alpha code. These enable you to more easily debug code that you are porting to OpenVMS Alpha systems. For example, you can use the /UNALIGNED_DATA qualifier with the SET command to cause the debugger to break directly after any instruction that accesses unaligned data (such as a load word instruction which accesses data that is not on a word boundary).

You can use the /RETURN qualifier with the SET command for any routine. It is not limited to routines called with a CALLS or CALLG instruction as it is on an OpenVMS VAX system. For more information about features specific to OpenVMS Alpha systems, see the *OpenVMS Debugger Manual*.

4.3.5 Delta/XDelta Debugger

The Delta/XDelta Debugger (DELTA/XDELTA), running on OpenVMS Alpha systems, provides enhancements to existing commands and several new commands necessitated by the Alpha architecture. The enhancements include the display of base registers in decimal instead of hexadecimal notation and the ability to look at the internal process identification (PID) number of another process. The new commands include ;Q, used to validate queues, and ;I, used to locate and display information about the current main image. For the Delta Debugger, the ;I command can also display information about all shareable images activated by the current main image. For more information about how the Delta/XDelta Debugger operates on OpenVMS Alpha systems, see the *OpenVMS Delta/XDelta Debugger Manual*.

4.3.6 OpenVMS Alpha System-Code Debugger

The OpenVMS Alpha System-Code Debugger is available for debugging nonpageable system code and device drivers running at any IPL. The OpenVMS Alpha System-Code Debugger is a symbolic debugger. You can specify variable names, routine names, and so on, precisely as they appear in your source code. You can also display the source code where the software is executing and step through it by source line.

You can use this debugger to debug code written in the following languages:

- C
- Bliss
- VAX MACRO

Note

A Bliss compiler is available on the OpenVMS Freeware CD that ships with OpenVMS VAX Version 6.2 and OpenVMS Alpha Version 6.2.

The OpenVMS Alpha System-Code Debugger recognizes the syntax, data typing, operators, expressions, scoping rules, and other constructs of a given language. If your program is written in more than one language, you can change the debugging context from one language to another during a debugging session.

For more information about Step 2 drivers and the OpenVMS Alpha System-Code Debugger, see the *OpenVMS AXP Device Support: Developer's Guide*.

4.3.7 System Dump Analyzer

The System Dump Analyzer (SDA) utility on OpenVMS Alpha systems is almost identical to the utility provided on OpenVMS VAX systems. Most commands, qualifiers, and displays are identical, although there are some additional commands and qualifiers, including several for accessing functions of the Crash Log Utility Extractor (CLUE) utility. Some displays have been adapted to show information specific to OpenVMS Alpha systems, such as processor registers and data structures.

While the SDA interface has changed only slightly, the contents of VAX and Alpha dump files and the entire process of analyzing a system crash from a dump differ significantly between the two computers. The Alpha execution paths leave more complex structures and patterns on the stack than VAX execution paths do.

To use SDA on a VAX computer, you must first familiarize yourself with the OpenVMS calling standard for VAX systems. Similarly, to use SDA on an Alpha system, you must familiarize yourself with the OpenVMS calling standard for Alpha systems before you can decipher the pattern of a crash on the stack.

The changes to SDA include the following:

- The displays of the SHOW CRASH and SHOW STACK commands contain additional information that make debugging fatal system exception bugchecks simpler.
- The SHOW EXEC command display includes additional information about executive images if they were loaded using image **slicing**. Slicing is a function performed by the executive image loader for executive images and by the OpenVMS Install utility for user-mode images. Slicing an executive image

(or a user-mode image) greatly improves performance by reducing contention for the limited number of translation buffer entries.

- The MAP command, a new SDA command, enables you to map an address in memory to an image offset in a map file.
- A new symbol, FPCR, has been added to the symbol table. This symbol represents a floating-point register.

4.3.8 Crash Log Utility Extractor

The Crash Log Utility Extractor (CLUE) is a tool for recording a history of crash dumps and key parameters for each crash dump and for extracting and summarizing key information. Unlike crash dumps, which are overwritten with each system crash and are available only for the most recent crash, the crash history file (on OpenVMS VAX) and the summary crash history file with a separate listing file for each crash (on OpenVMS Alpha), are permanent records of system crashes.

The implementation differences between OpenVMS VAX and OpenVMS Alpha are shown in Table 4–4.

Table 4–4 CLUE Differences Between OpenVMS VAX and OpenVMS Alpha

Attribute	OpenVMS VAX	OpenVMS Alpha
Access method	Invoked as a separate utility.	Accessed through SDA.
History file	A cumulative file that contains a one-line summary and detailed information from the crash dump file for each crash.	A cumulative file that contains only a one-line summary for each crash dump. The detailed information for each crash is put in a separate listing file.
Uses in addition to debugging crash dumps	None.	CLUE commands can be used interactively to examine a running system.
Documentation	*OpenVMS System Manager's Manual* and *OpenVMS System Management Utilities Reference Manual*	*OpenVMS System Manager's Manual* and *OpenVMS AXP System Dump Analyzer Utility Manual*

4.3.9 Mathematics Libraries

Mathematical applications using the standard VMS call interface to the OpenVMS Mathematics (MTH$) Run-Time Library (RTL) need not change their calls to MTH$ routines when migrating to an OpenVMS Alpha system. Jacket routines are provided that map MTH$ routines to their math$ counterparts in the Digital Portable Mathematics Library (DPML) for OpenVMS Alpha systems. However, there is no support in the DPML for calls made to JSB routine entry points and vector routines. Note that DPML routines are different from those in the OpenVMS MTH$ RTL, and you should expect to see small differences in the precision of the mathematical results.

To maintain compatibility with future libraries and to create portable mathematical applications, Digital recommends that you use the DPML routines available through the high-level language of your choice (for example, DEC Fortran or DEC C) rather than using the call interface. Significantly higher performance and accuracy are also available to you with DPML routines.

For more information about the DPML routines, see the *Digital Portable Mathematics Library* manual.

4.3.10 Determining the Host Architecture

Your application may need to determine whether it is running on an OpenVMS VAX system or an Alpha system. From within your program, you can obtain this information by calling the $GETSYI system service (or the LIB$GETSYI RTL routine), specifying the ARCH_TYPE item code. When your application is running on a VAX computer, the $GETSYI system service returns the value 1. When your application is running on an Alpha computer, the $GETSYI system service returns the value 2.

Example 4–1 illustrates how to determine the host architecture in a DCL command procedure by calling the F$GETSYI lexical function and specifying the ARCH_TYPE item code. For an example of calling the $GETSYI system service in an application to determine the page size of an Alpha computer, see *Migrating to an OpenVMS AXP System: Recompiling and Relinking Applications*.

Example 4–1 Using the ARCH_TYPE Keyword to Determine Architecture Type

```
$! Determine architecture type
$ type_symbol = f$getsyi("arch_type")
$ if type_symbol .eq. 1 then goto ON_VAX
$ if type_symbol .eq. 2 then goto ON_ALPHA
$ ON_VAX:
$ !
$ ! Do VAX-specific processing
$ !
$ exit
$ ON_ALPHA:
$ !
$ ! Do ALPHA-specific processing
$ !
$ exit
```

Note, however, that the ARCH_TYPE item code is available only on VAX computers running OpenVMS VAX Version 5.5 or later. If your application needs to determine the host architecture for earlier versions of the operating system, use one of the other $GETSYI system service item codes listed in Table 4–5.

Table 4–5 $GETSYI Item Codes That Specify Host Architecture

Keyword	Usage
ARCH_TYPE	Returns 1 on a VAX computer; returns 2 on an Alpha computer. Supported on Alpha computers and on VAX computers running OpenVMS VAX Version 5.5 or a later version.
ARCH_NAME	Returns text string “VAX” on VAX computers and text string “Alpha” on Alpha computers. Supported on Alpha computers and on VAX computers running OpenVMS VAX Version 5.5 or a later version.
HW_MODEL	Returns an integer that identifies a particular hardware model. All values equal to or larger than 1024 identify Alpha computers.
CPU	Returns an integer that identifies a particular CPU. The value 128 identifies a system as “not a VAX.” This code is supported on much earlier versions of VMS than the ARCH_TYPE and ARCH_NAME codes.

4.3.11 Uncovering Latent Bugs

Despite your best efforts, and following all the previous suggestions, you may encounter bugs that were in your program all along, but never caused a problem on an OpenVMS VAX system. For example, a failure to initialize some variable in your program might have been benign on a VAX computer but could produce an arithmetic exception on an Alpha computer. The same could be true moving between any other two architectures, because the available instructions and the way compilers optimize them is bound to change. There is no magic answer for bugs that have been in hiding, but you should test your programs after porting them before making them available to other users.

4.4 Application Compatibility with Future OpenVMS Alpha Releases

Programs that run on OpenVMS Alpha systems will continue to run unmodified on future OpenVMS Alpha releases, except for those programs that use inner modes or are linked against the system symbol table. As on OpenVMS VAX systems, you will need to recompile and relink any programs that use inner modes or are linked against the system symbol table for them to run on future OpenVMS Alpha releases.

4.5 Guidelines for Developing Applications for OpenVMS VAX and OpenVMS Alpha

The following guidelines are provided to facilitate the development of applications intended to run on both OpenVMS VAX systems and OpenVMS Alpha systems:

1. Familiarize yourself with the changes and the new features of the DEC compiler that you will use. For example:
   - If you are writing any C code with external symbols, familiarize yourself with the changes in DEC C to coding external symbols and compiling such code.
   - If your program needs to dynamically establish condition handlers, you may need to make some changes to your code, as described in Section 4.1.2.
2. Write your code in mid- or high-level languages whenever possible.
3. Follow good programming practices for program modularity.
4. Avoid creating any VAX architectural dependencies in your code. One area that can be troublesome is that of shared memory. The VAX architecture makes certain implicit guarantees about synchronization. If a program requires the same synchronization on the Alpha architecture, it must request it explicitly. This potential problem and others are briefly described in Section 4.1.1. For more information, see *Migrating to an OpenVMS AXP System: Recompiling and Relinking Applications* and the user's guide for the compiler you will use.

   If you cannot avoid relying on the VAX architecture for one or more operations, conditionalize your code for each architecture, as shown in Example 4–1.
5. Familiarize yourself with the differences between the linker on OpenVMS VAX and the linker on OpenVMS Alpha. Some of the differences are described in Section 4.3.1.

6. Examine your OpenVMS VAX build procedures and note what changes may be necessary for recompiling and relinking on an OpenVMS Alpha system.
7. Do not use OpenVMS VAX features that are not yet supported by the OpenVMS Alpha operating system. If in doubt, check with your Digital account representative or authorized reseller.
8. Use at least aligned longwords for in-memory data structures wherever possible. This has always been more efficient on VAX computers. On Alpha computers, the performance difference becomes even greater.

4.6 Guidelines for Developing Applications for Mixed-Architecture VMScluster Systems

These guidelines are provided to facilitate the evaluation and modification, if necessary, of applications intended to run in a mixed-architecture VMScluster system.

A basic assumption about applications running in a mixed-architecture VMScluster system is that they should work just as they do in a single-architecture VMScluster or VAXcluster system. VMScluster components, such as the lock manager, RMS, and the connection manager, behave the same in a mixed-architecture VMScluster as they do in a single-architecture VMScluster or VAXcluster system.

Most OpenVMS VAX applications require some migration work to run on an Alpha computer. The exceptions are the few applications that utilize only simple DCL commands, that is, they do not run any application images. In general, you need to at least recompile and relink your application so that it will run on an Alpha computer.

For an application to work in a mixed-architecture VMScluster system, consider the following aspects of an application:

- User interface
- System management interface
- Interactions between the application executing on VAX and Alpha nodes
  - File format compatibility
  - Data packing
  - Data-type selection
  - Buffer size
  - Data access and locking
- Missing features

Neither users nor system managers should notice a difference in the behavior of the application, regardless of whether the node in a VMScluster system is a VAX or an Alpha computer. Users should expect to see better application performance on OpenVMS Alpha systems. System managers should expect to maintain architecture-specific copies of image files. Other differences should be minimized.

Application compatibility with mixed-architecture VMScluster systems can vary, depending on the degree of differences visible to users and system managers. The following sections should help you determine how well your application will function in a mixed-architecture VMScluster environment and what modifications may be necessary.

4.6.1 User Interface

User interfaces for a given application should be identical on both VAX and Alpha nodes. At a minimum, one implementation of the user interface should be a subset of the other. In addition, the differences may need to be noted in user documentation.

4.6.2 System Management

The system management aspects of an application pertain to the similarity of the installation and licensing mechanisms on both VAX and Alpha. The goal for the application developer is to minimize the work required for managing different architectural versions of applications that are running in the mixed-architecture VMScluster system.

4.6.3 File Format Compatibility

It is important to clearly differentiate between two kinds of files—image files (including shareable images (.EXE)) and application data files. Image files cannot be activated across architectures. An image built for VAX will run only on a VAX computer and an image built for Alpha will run only on an Alpha computer. If an application kit is to be shared by Alpha and VAX systems, it must either contain separate images for each platform, perform the link during the installation, or use DECmigrate for OpenVMS Alpha Systems to translate the application. Note that DECmigrate is an optional product, released separately from the OpenVMS VAX operating system.

A common kit will probably need conditional statements for linking, as the process for linking shareable images differs between VAX and Alpha systems. F$GETSYI (“ARCH_TYPE”) can be used from DCL, as described in Section 4.3.10.

Applications should be able to share data files across the Alpha and VAX systems; doing so is usually required for the application to run properly in a mixed-architecture VMScluster environment.

Note

If full data file compatibility cannot be achieved, applications should ensure that the incompatible format is recognized by both types of systems and an error message is issued.

4.6.4 Data Packing

Aligned data can provide significant performance improvements on VAX systems (up to 4 times faster) and on Alpha systems (up to 100 times faster). Unaligned data was typical on older VAX systems where minimal consumption of memory was very important and data was packed tightly together, ignoring alignment issues.

If you share data between Alpha and VAX systems, avoid unaligned data structures. Use at least longword alignment for in-memory data structures whenever possible; quadword alignment is preferred, if possible.

If you have a real-time application or an application that exhibits poor file I/O characteristics and shares data through a file, you may find it advantageous to align the file data structures.

For more information, see *OpenVMS Compatibility Between VAX and Alpha* and *Migrating to an OpenVMS AXP System: Recompiling and Relinking Applications*.

4.6.5 Data-Type Selection

Check to see if your application depends on the size or bit representation of an underlying data type. Ideally, all data accessed by different nodes in the VMScluster system should have the identical format on VAX and Alpha nodes. Examples include the following:

- Only ASCII, integer, or F and G VAX floating-point data formats are used in files (no H_float, no IEEE float, no machine instructions).

  For Fortran, this can be relaxed, since Fortran provides run-time support for automatic conversion between floating-point types via the /CONVERT compiler qualifier, and so forth.

- All structures stored in files are equally aligned for VAX and Alpha systems.

Different data formats can be used if format translation is transparent to the user.

Support for floating-point data types differs between OpenVMS VAX and OpenVMS Alpha systems, as shown in Table 4–1.

You may want to use the IEEE floating-point formats on Alpha systems for coexistence with other open systems. However, using the IEEE floating-point formats may impede mixed-architecture VMScluster interoperability, because the T, S, and X IEEE floating-point data types are not supported on VAX systems.

For Fortran programs, this is not a major obstacle because of the /CONVERT compiler qualifier or the CONVERT= option on OPEN statements. For C programs, the automatic conversion of binary floating-point data in files is not possible, so this is an issue.

If your VAX application requires full 56-bit precision, that is, the D56_float data type, or if it requires the H_float data type, you can translate your image with DECmigrate. Because a translated image runs significantly slower than a native Alpha image, isolate the set of routines requiring 56-bit D_float precision or the H_float data type into its own small image for translation, and run the rest of the application native.

If your application contains data types that are not supported on OpenVMS Alpha, you can make the changes that are appropriate for it (either translation or recoding). You can then port your application to OpenVMS Alpha systems and share data between Alpha and VAX systems in a mixed-architecture VMScluster system. For more information about translation support, see the Section 4.2.3. For more information about supported data types, see the documentation for the compiler you are using.

4.6.6 Buffer Size

If you need to make changes to your code, such as aligning data structures and using different data types, you will need to review the buffer sizes in your application. Such changes generally require larger buffers.

4.6.7 Data Access and Locking

Data access is synchronized using the same resource names and locking sequences on VAX and Alpha nodes. This is not new for mixed-architecture VMScluster systems, since the need to ensure resource naming conventions is standard for VMScluster systems today. However, you should test your application to ensure correct operation.

4.6.8 Missing Features

Check your application for dependencies on features that might not exist on the target system (see Chapter 1). Also, check your application for dependencies on features provided by middleware applications that might not exist on the target system. Because a feature or application may not be available on both platforms, using the feature or application in a mixed-architecture VMScluster system could yield unanticipated results.

4.7 Application Compatibility Checklist

Use the following checklist to determine how well your application will work in a mixed-architecture VMScluster environment.

1. **User Interface**

   ☐ Does the application present the same user interface on both architectures?

   - If not, is one interface an easily described subset of the other?

   ☐ Does the application use the same documentation on both architectures?

   ☐ Are user interface differences documented for the user?

   ☐ Are error messages the same across the architectures for the same error condition?

2. **System Management**

   ☐ Are version numbers for feature-compatible versions identical for both architectures?

   ☐ Is a single version of the License Management Facility (LMF) possible? Can both the VAX and Alpha versions of the application run using comparable licensing mechanisms? There should not be a requirement that the application use LMF Version 1.*x* on a VAX system and a different LMF version on an Alpha system.

   ☐ Are separate installations required for each architecture?

   ☐ Does the application use the same installation procedure and documentation?

   ☐ Does the application work across architectures without additional system management steps?

   ☐ Does the application present comparable failure scenarios and recovery capabilities on both architectures?

3. **File Format Compatibility**

   ☐ Are all files associated with the application accessible and used equally on each architecture? Remember that object and image libraries and image files are different between VAX and Alpha systems. (Other library types, such as .TLB and .HLB, do not present a problem. They are the same.)

☐ Can files that do not contain machine instructions be used equally on either architecture?

- Is full compatibility the default or is user action required?
- How does the application manage differences in file formats?
- If your application does not allow files to be shared, does it recognize an incompatible file format and present a suitable warning message?

4. **Data Packing**

☐ Does your application use unaligned data structures?

- If so, what is the performance impact for your application?

5. **Data-Type Selection**

☐ Does your product use D56_float or H_float on VAX?

- If so, what is the performance impact if you use the emulation routines that provide full D56_float or full H_float support?
- What is the precision-loss impact if you replace D56_float with D53_float?

6. **Buffer Size**

☐ Will you need to change the alignment of any data structures or any data types in your application?

- If so, what are the implications on buffer size?

7. **Data Access and Locking Compatibility**

☐ Does your product provide compatibility of resource names?

☐ Does your product provide compatibility of locking sequences?

8. **Missing Features**

☐ Does your product utilize features that are only available on one platform?

☐ Are you documenting these differences to your users so they will know what will work in a mixed-architecture VMScluster?

A

DCL Differences

A.1 DIGITAL Command Language (DCL)

The DIGITAL Command Language (DCL), the standard user interface to OpenVMS, remains essentially unchanged with OpenVMS Alpha. All commands and qualifiers available on OpenVMS VAX are also available on OpenVMS Alpha, except for a few, as shown in Table A–1.

Because of architectural differences between VAX and Alpha computers, some differences exist in the implementation of DCL commands, qualifiers, and lexical functions. These differences are noted in Table A–1.

Note

The RECALL command and the SET HOST/LAT command achieved functional equivalence with the release of OpenVMS VAX Version 6.2 and OpenVMS Alpha Version 6.2.

Table A–1 DCL Differences Between OpenVMS VAX and OpenVMS Alpha

Command/Qualifier	On VAX	On Alpha
ANALYZE/IMAGE	System versions displayed are the versions of the system symbol table, the image that is linked against the executive.	System versions displayed are the versions of the system shareable image, the image that is linked against the executive.
	Cannot analyze an OpenVMS Alpha image.	Can analyze both OpenVMS Alpha and OpenVMS VAX images.
ANALYZE/PROCESS	You use the OpenVMS Debugger to analyze the dumped image.	In some cases, you cannot use the OpenVMS Debugger, such as when the dumped image's PC is set to an invalid address. Instead, you can use the Delta Debugger.
CLUE	Invokes CLUE.	CLUE commands are accessed through SDA. For more information, see Section 4.3.8.
DIAGNOSE	Not available; under investigation.	Invokes the DECevent utility and selectively reports the contents of one or more event log files.

(continued on next page)

Table A–1 (Cont.) DCL Differences Between OpenVMS VAX and OpenVMS Alpha

Command/Qualifier	On VAX	On Alpha
FONT	Converts an ASCII bitmap distribution format (BDF) into binary server natural form (SNF).	Converts an ASCII bitmap distribution format (BDF) into binary portable compiled format (PCF).
INITIALIZE /STRUCTURE=level	Supports Files–11 On-Disk Structure Level 1 Disks.	Does not support Files–11 On-Disk Structure Level 1 Disks.
MACRO/ALPHA	Not available.	Invokes the native MACRO–64 assembler, if installed. For more information, see Section 4.3.2.
MACRO/MIGRATION	Not available.	Invokes the MACRO–32 compiler. For more information, see Section 4.2.1.
PATCH	Invokes the Patch utility.	Not available; limited functionality under investigation.
SET HOST/DUP	Commands for installing FYDRIVER differ from those used on Alpha.	Commands for installing FYDRIVER differ from those used on VAX.
SET PASSWORD	The random password generator differs from that on Alpha. One difference is that the passwords presented on VAX are hyphenated.	The random password generator presents passwords that are not hyphenated. For more information, see Section 1.3.9.
SET TERMINAL /PROTOCOL=DDCMP	Controls whether the terminal port specified is changed into an asynchronous DDCMP line.	Not available.
SET TERMINAL /SWITCH=DECNET	Causes the terminal lines at each node to be switched to dynamic asynchronous DDCMP lines when specified with the /PROTOCOL=DDCMP qualifier.	Not available.
SHOW MEMORY/GH_ REGIONS	Not available.	Displays information about the granularity hint regions (GHR) that have been established.
SHOW MEMORY dynamic memory parameter	Each dynamic memory area displayed in bytes and pages.	Each dynamic memory area displayed in pagelets.
Working set qualifiers such as /WSDEFAULT, /WSEXTENT, and /WSQUOTA	Specified in units of 512-byte pages	Specified in units of 512-byte pagelets, rounded to the nearest CPU-specific page. For more information, see Section 1.4.1.6.

(continued on next page)

Table A–1 (Cont.) DCL Differences Between OpenVMS VAX and OpenVMS Alpha

Lexical Functions	On VAX	On Alpha
F$CONTEXT	Base priority valid range 0–31.	Base priority valid range 0–63.
F$GETSYI	For CPU, the integer that identifies the processor type is stored in the processor's system identification (SID) register.	For CPU, the integer that identifies the processor type is stored in the hardware restart parameter block (HWRPB).
	Not available	For CONSOLE_VERSION, returns the console firmware version of your system.
	For HW_MODEL, the integer that identifies the model type is less than or equal to 1023	For HW_MODEL, the integer that identifies the model type is greater than 1023
	Not available	For PALCODE_VERSION, returns the PALcode (privileged architecture library) version of your system.

Index

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X