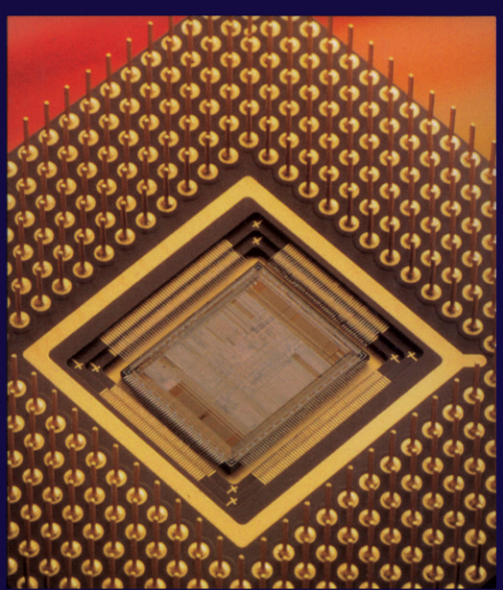
Alpha Architecture Reference Manual

EDITED BY

RICHARD L. SITES



Alpha Architecture Reference Manual

This page intentionally left blank

Alpha Architecture Reference Manual

Contributing Authors

Richard Witek
Alpha co-architect

and
Ellen M. Batbouta
Richard A. Brunner
Wayne M. Cardoza
Daniel W. Dobberpuhl
Robert A. Giggi
Henry N. Grieb
Richard B. Grove
Robert H. Halstead, Jr.
Michael S. Harvey
Nancy P. Kronenberg
Raymond J. Lanza
Stephen J. Morris
William B. Noyce
Charles G. Nylander
Mary H. Payne
Audrey R. Reith
Robert M. Supnik
Benjamin J. Thomas
Catharine Van Ingen

Edited by
Richard L. Sites
Alpha co-architect

digital DIGITAL PRESS

Printed in the United States of America.

9 8 7 6 5 4 3 2 1

Order number EY-L520E-DP
ISBN 1-55558-098-X

Technical Writer: Charles Greenman
Production Editor: Kathe Rhoades
Technical Illustrator: Lynne Kenison
Cover Design: Marshall Henrichs

Contents

Appendixes

Foreword

In the foreword to the *VAX Architecture Reference Manual*, Sam Fuller, Digital's Vice President for Research and Architecture, wrote, "Computer design continues to be a dynamic field; I expect we will see more rather than less change and innovation in the decades ahead." *The Alpha Architecture Reference Manual* demonstrates the accuracy of that prediction.

Alpha follows VAX by about fifteen years. Those fifteen years have witnessed a torrent of change in computer technology, one that shows no sign of abating:

- More than a 1000-fold increase in the performance of microprocessors
- More than a 1000-fold increase in the density of semiconductor memories
- More than a 500-fold increase in the density of magnetic storage devices
- More than a 100-fold increase in the speed of network connections

During the same period, the internal organization of computer systems has changed as well, based on developments such as RISC architecture, symmetric multiprocessing, and coherent distributed systems. Moreover, the fundamental paradigms of computing have changed not once, but several times, with the introduction of personal computers, graphics workstations, local area networks, and client/server computing.

These developments present an enormous challenge for computing in the 21st century. Future computers will be called upon to solve problems of great scale and complexity, worldwide, in a distributed manner. They will have to provide unprecedented performance, flexibility, reliability, and scalability in order to implement a global infrastructure of information, and to give users an untrammeled window on the world.

Alpha is Digital's response to the challenges of 21st-century computing. It represents the culmination of the company's knowledge and belief about how the next generations of computers should be built. Alpha is based on a decade's experimental and engineering work in RISC architecture, high-speed implementation, software compatibility and migration, and system serviceability. It provides the foundation for implementations ranging from mobile computing units to massively parallel supercomputers.

Alpha is designed to handle the largest computing problems of today and tomorrow. When the Alpha architecture is compared to its predecessor, the VAX architecture, two differences stand out immediately. First, Alpha is a 64-bit architecture; VAX is a 32-bit architecture. This means that Alpha's virtual address extends to a 64-bit linear range of bytes in memory. Supporting this extended virtual address space are an extended maximum physical address range (up to 48 bits) and larger pages (8KB to 64KB). Alpha's extended virtual address range allows direct manipulation

of the gigabytes and terabytes of data produced in electrical and mechanical design, database and transaction processing, and imaging.

Second, Alpha is a RISC architecture; VAX is a CISC architecture. RISC stands for Reduced Instruction Set Computer, CISC for Complex Instruction Set Computer. RISC architectures are characterized by simple, fixed-length instruction formats; a small number of addressing modes; large register files; a load-store instruction set model; and direct hardware execution of instructions. CISC architectures are characterized by variable-length instruction formats; a large number of addressing modes; small-to-medium-sized register files; a full set of register-to-memory (or even memory-to-memory) instructions; and microcoded execution of instructions. Alpha's streamlined organization facilitates high-speed implementation in a variety of technologies, while providing strong compatibility with today's programs and data.

The following tabulation contrasts the architectural differences between VAX and Alpha:

	VAX	**Alpha**
Architecture	CISC	RISC
Virtual address range	32 bits	Up to 64 bits
Physical address range	Up to 32 bits	Up to 48 bits
Page size	512 bytes	8KB–64KB
Instruction lengths	1–51 bytes	4 bytes
General registers	16 × 32 bits	64 × 64 bits
Addressing modes	21	3
Instruction set architecture	General	Load-store
Directly supported data types	Integer, floating, bit field, queue, character string, decimal string	Integer, floating

This book is the culmination of an effort begun three years ago. In that time, Alpha has grown from a paper specification to a cohesive set of chips, systems, and software, spanning the computer spectrum. This achievement is due to the efforts of many hundreds of people in Engineering, Marketing, Sales, Service, and Manufacturing. This book is documentation of, and a tribute to, the outstanding work they have done.

Bob Supnik
Corporate Consultant,
Vice President

Preface

The Alpha architecture is a RISC architecture that was designed for high performance and longevity. Following Amdahl, Blaauw, and Brooks,[1] we distinguish between architecture and implementation:

- Computer architecture is defined as the attributes of a computer seen by a machine-language programmer. This definition includes the instruction set, instruction formats, operation codes, addressing modes, and all registers and memory locations that may be directly manipulated by a machine-language programmer.
- Implementation is defined as the actual hardware structure, logic design, and data-path organization.

This architecture book describes the required behavior of all Alpha implementations, as seen by the machine-language programmer. The architecture does not speak to implementation considerations such has how fast a program runs, what specific bit pattern is left in a hardware register after an unpredictable operation, how to schedule code for a particular chip, or how to wire up a given chip; those considerations are described in implementation-specific documents.

Various Alpha implementations are expected over the coming years, starting with the Digital 21064 chip.

Goals

When we started the Alpha project in the fall of 1988, we had a small number of goals:

1. High performance
2. Longevity
3. Run VMS and UNIX
4. Easy migration from VAX (and soon-to-be MIPS) customer base

As principal architects, Rich Witek and I made design decisions that were driven directly by these goals.

We assumed that high performance was needed to make a new architecture attractive in the marketplace, and to keep Digital competitive.

We set a 15–25 year design horizon (longevity) and tried to avoid any design elements that we thought would become limitations during this time. The design horizon led directly to the conclusion that Alpha could not be a 32-bit architecture: 32-bit addresses will be too small within 10 years. We thus adopted a full 64-bit

1. Amdahl, G.M., G.A. Blaauw, and F.P. Brooks, Jr. "Architecture of the IBM System/360." *IBM Journal of Research and Development*, vol. 8, no. 2 (April 1964): 87–101.

architecture, with a minimal number of 32-bit operations for backward compatibility. Wherever possible, 32-bit operands are put in registers in a 64-bit canonical form and operated upon with 64-bit operations.

The longevity goal also caused us to examine how the performance of implementations would scale up over 25 years. Over the past 25 years, computers have become about 1000 times faster. This suggested to us that Alpha implementations would need to do the same, or we would have to bet that the industry would fall off the historical performance curve. We were unwilling to bet against the industry, and were unwilling to ignore the issue, so we seriously examined the consequences of longevity.

We thought that it would be realistic for implementors to improve clock speeds by a factor of 10 over 25 years, but not by a factor of 100 or 1000. (Clock speeds have improved by about a factor of 100 over the past 25 years, but physical limits are now slowing down the rate of increase.)

We concluded that the remaining factor of 100 would have to come from other design dimensions. If you cannot make the clock faster, the next dimension is to do more work per clock cycle. So the Alpha architecture is focused on allowing implementations that issue many instructions every clock cycle. We thought that it would be realistic for implementors to achieve about a factor of 10 over 25 years by using multiple instruction issue, but not a factor of 100. Even a factor of 10 will require perhaps a decade of compiler research.

We concluded that the remaining factor of 10 would have to come from some other design dimension. If you cannot make the clock faster, and cannot do more work per clock, the next dimension is to have multiple clocked instruction streams, that is, multiple processors. So the Alpha architecture is focused on allowing implementations that apply multiple processors to a single problem. We thought that it would be realistic for implementors to achieve the remaining factor of 10 over 25 years by using multiple processors.

Overall, the factor-of-1000 increase in performance looked reasonable, but required factor-of-10 increases in three different dimensions. These three dimensions therefore formed part of our design framework:

- Gracefully allow fast cycle-time implementations
- Gracefully allow multiple-instruction-issue implementations
- Gracefully allow multiple-processor implementations

The cycle-time goal encouraged us to keep the instruction definitions very simple, and to keep the interactions between instructions very simple. The multiple-instruction-issue goal encouraged us to eliminate specialized registers, architected delay slots, precise arithmetic traps, and byte writes (with their embedded read-modify-write bottleneck). The multiple-processor goal encouraged us to consider the memory model and atomic-update primitives carefully. We adopted load-locked/store-conditional sequences as the atomic-update primitive, and eliminated strict read-write ordering between processors.

All of the above design decisions were driven directly by the performance and

longevity goals. The lack of byte writes, precise arithmetic traps, and multiprocessor read/write ordering have been the most controversial decisions, so far.

Clean Sheet of Paper

To run both OpenVMS and UNIX without burdening the hardware implementations with elaborate (and sometimes conflicting) operating system underpinnings, we adopted an idea from a previous Digital RISC design. Alpha places the underpinnings for interrupt delivery and return, exceptions, context switching, memory management, and error handling in a set of privileged software subroutines called PALcode (privileged architecture library code). PALcode subroutines have controlled entries, run with interrupts turned off, and have access to real hardware (implementation) registers. By having different sets of PALcode for different operating systems, the architecture itself is not biased toward a specific operating system or computing style.

PALcode allowed us to design an architecture that could run OpenVMS gracefully without elaborate hardware and without massively rewriting the VMS synchronization and protection mechanisms. PALcode lets the Alpha architecture support some complex VAX primitives (such as the interlocked queue instructions) that are heavily used by OpenVMS, without burdening a UNIX implementation in any way.

Finally, we also considered how to move VAX and MIPS code to Alpha. We rejected various forms of "compatibility mode" hardware, because they would have severely compromised the performance and time-to-market of the first implementation. After some experimentation, we adopted the strategy of running existing binary code by building software translators. One translator converts OpenVMS VAX images to functionally identical OpenVMS Alpha images. A second translator converts MIPS ULTRIX images to functionally identical DEC OSF/1 Alpha images.

Fundamentally, PALcode gave us a migration path for existing operating systems, and the translators (and native compilers) gave us a migration path for existing user-mode code. PALcode and the translators provided a clean sheet of design paper for the bulk of the Alpha architecture. Other than an extra set of VAX floating-point formats (included for good business reasons, but subsettable later), no specific VAX or MIPS features are carried directly into the Alpha architecture for compatibility reasons.

These considerations substantially shaped the architecture described in the rest of this book.

Organization

The first part of this book describes the instruction-set architecture, and is largely self-contained for readers who are involved with compilers or with assembly language programming. The second and third parts describe the supporting PALcode routines for each operating system—the specific operating system PALcode architecture.

Acknowledgments

My collaboration with Rich Witek over the past few years has been extremely rewarding, both personally and professionally. By combining our backgrounds and viewpoints, we have produced an architecture that is substantially better than either of us could have produced alone. Thank you, Rich.

A work of this magnitude cannot be done on a shoestring or in isolation. Rich and I were blessed with a rich environment of dozens and later hundreds of bright, thoughtful, and outspoken professional peers. I thank the management of Digital Equipment Corporation for providing that rich environment, and those peers for making the architecture so much more robust and well-considered.

Three people have especially influenced my views of computer architecture, through personal interaction and landmark machine design: Fred Brooks, John Cocke, and Seymour Cray. This work is built directly upon theirs, and could not exist without them.

The organization, editing, and production of this text in final form is largely the work of Charlie Greenman, whose clear writing is much appreciated.

Richard L. Sites
May 1992

A Note on the Structure of This Book

The *Alpha Architecture Reference Manual* is divided into three parts, three appendixes, and an index. Each part describes a major portion of the Alpha architecture. Each contains its own table of contents.

The following tabulation outlines the book's contents:

Name	Contents
Part I	Common Architecture This part describes the instruction-set architecture that is common to and required by all implementations.
Part II	OpenVMS Alpha Software This part describes how the OpenVMS operating system relates to the Alpha architecture.
Part III	DEC OSF/1 Alpha Software This part describes how the DEC OSF/1 operating system relates to the Alpha architecture.
Appendixes	The appendixes describe implementation considerations, IEEE floating-point conformance, and instruction encodings.
Index	Index entries are called out by the symbol (I), (II), or (III). Each symbol is associated with the corresponding Part. Index entries for the appendixes are called out by appendix name and page number.

This page intentionally left blank

Part I Common Architecture

This part describes the common Alpha architecture and contains the following chapters:

1. Introduction
2. Basic Architecture
3. Instruction Formats
4. Instruction Descriptions
5. System Architecture and Programming Implications
6. Common PALcode Architecture
7. Console Subsystem Overview
8. Input/Output

This page intentionally left blank

Contents

Common Architecture (I)

Chapter 1 Introduction (I)

Chapter 2 Basic Architecture (I)

Chapter 3 Instruction Formats (I)

Chapter 4 Instruction Descriptions (I)

Chapter 5 System Architecture and Programming Implications (I)

Chapter 6 Common PALcode Architecture (I)

Chapter 7 Console Subsystem Overview (I)

Chapter 8 Input/Output (I)

Figures

Tables

Chapter 1
Introduction (I)

Alpha is a 64-bit load/store RISC architecture that is designed with particular emphasis on the three elements that most affect performance: clock speed, multiple instruction issue, and multiple processors.

The Alpha architects examined and analyzed current and theoretical RISC architecture design elements and developed high-performance alternatives for the Alpha architecture. The architects adopted only those design elements that appeared valuable for a projected 25-year design horizon. Thus, Alpha becomes the first 21st century computer architecture.

The Alpha architecture is designed to avoid bias toward any particular operating system or programming language. Alpha initially supports the OpenVMS Alpha and DEC OSF/1 operating systems, and supports simple software migration from applications that run on those operating systems.

This manual describes in detail how Alpha is designed to be the leadership 64-bit architecture of the computer industry.

1.1 The Alpha Approach to RISC Architecture

Alpha Is a True 64-Bit Architecture

Alpha was designed as a 64-bit architecture. All registers are 64 bits in length and all operations are performed between 64-bit registers. It is not a 32-bit architecture that was later expanded to 64 bits.

Alpha Is Designed for Very High-Speed Implementations

The instructions are very simple. All instructions are 32 bits in length. Memory operations are either loads or stores. All data manipulation is done between registers.

The Alpha architecture facilitates pipelining multiple instances of the same operations because there are no special registers and no condition codes.

The instructions interact with each other only by one instruction writing a register or memory and another instruction reading from the same place. That makes it particularly easy to build implementations that issue multiple instructions every CPU cycle. (The first implementation issues two instructions per cycle.)

Alpha makes it easy to maintain binary compatibility across multiple implementations and easy to maintain full speed on multiple-issue implementations. For example, there are no implementation-specific pipeline timing hazards, no load-delay slots, and no branch-delay slots.

Alpha's Approach to Byte Manipulation

The Alpha architecture does byte shifting and masking with normal 64-bit register-to-register instructions, crafted to keep instruction sequences short.

Alpha does not include single-byte store instructions. This has several advantages:

- Cache and memory implementations need not include byte shift-and-mask logic, and sequencer logic need not perform read-modify-write on memory locations. Such logic is awkward for high-speed implementation and tends to slow down cache access to normal 32-bit or 64-bit aligned quantities.
- Alpha's approach to byte manipulation makes it easier to build a high-speed error-correcting write-back cache, which is often needed to keep a very fast RISC implementation busy.
- Alpha's approach can make it easier to pipeline multiple byte operations.

Alpha's Approach to Arithmetic Traps

Alpha lets the software implementor determine the precision of arithmetic traps. With the Alpha architecture, arithmetic traps (such as overflow and underflow) are imprecise—they can be delivered an arbitrary number of instructions after the instruction that triggered the trap. Also, traps from many different instructions can be reported at once. That makes implementations that use pipelining and multiple issue substantially easier to build.

However, if precise arithmetic exceptions are desired, trap barrier instructions can be explicitly inserted in the program to force traps to be delivered at specific points.

Alpha's Approach to Multiprocessor Shared Memory

As viewed from a second processor (including an I/O device), a sequence of reads and writes issued by one processor may be arbitrarily reordered by an implementation. This allows implementations to use multibank caches, bypassed write buffers, write merging, pipelined writes with retry on error, and so forth. If strict ordering between two accesses must be maintained, explicit memory barrier instructions can be inserted in the program.

The basic multiprocessor interlocking primitive is a RISC-style load_locked, modify, store_conditional sequence. If the sequence runs without interrupt, exception, or an interfering write from another processor, then the conditional store succeeds. Otherwise, the store fails and the program eventually must branch back and retry the sequence. This style of interlocking scales well with very fast caches, and makes Alpha an especially attractive architecture for building multiple-processor systems.

Alpha Instructions Include Hints for Achieving Higher Speed

A number of Alpha instructions include hints for implementations, all aimed at achieving higher speed.

- Calculated jump instructions have a target hint that can allow much faster subroutine calls and returns.
- There are prefetching hints for the memory system that can allow much higher cache hit rates.

- There are granularity hints for the virtual-address mapping that can allow much more effective use of translation lookaside buffers for large contiguous structures.

PALcode—Alpha's Very Flexible Privileged Software Library

A Privileged Architecture Library (PALcode) is a set of subroutines that are specific to a particular Alpha operating system implementation. These subroutines provide operating-system primitives for context switching, interrupts, exceptions, and memory management. PALcode is similar to the BIOS libraries that are provided in personal computers.

PALcode subroutines are invoked by implementation hardware or by software CALL_PAL instructions.

PALcode is written in standard machine code with some implementation-specific extensions to provide access to low-level hardware.

One version of PALcode lets Alpha implementations run the full OpenVMS operating system by mirroring many of the OpenVMS VAX features. The OpenVMS PALcode instructions let Alpha run OpenVMS with little more hardware than that found on a conventional RISC machine: the PAL mode bit itself, plus 4 extra protection bits in each Translation Buffer entry.

Another version of PALcode lets Alpha implementations run the OSF/1 operating system by mirroring many of the RISC ULTRIX features. Other versions of PALcode can be developed for real-time, teaching, and other applications.

PALcode makes Alpha an especially attractive architecture for multiple operating systems.

Alpha and Programming Languages

Alpha is an attractive architecture for compiling a large variety of programming languages. Alpha has been carefully designed to avoid bias toward one or two programming languages. For example:

- Alpha does not contain a subroutine call instruction that moves a register window by a fixed amount. Thus, Alpha is a good match for programming languages with many parameters and programming languages with no parameters.
- Alpha does not contain a global integer overflow enable bit. Such a bit would need to be changed at every subroutine boundary when a FORTRAN program calls a C program.

1.2 Data Format Overview

Alpha is a load/store RISC architecture with the following data characteristics:

- All operations are done between 64-bit registers.
- Memory is accessed via 64-bit virtual little-endian byte addresses.
- There are 32 integer registers and 32 floating-point registers.
- Longword (32-bit) and quadword (64-bit) integers are supported.