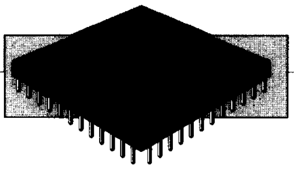


The Alpha AXP Architecture and 21064 Processor

The Alpha AXP 64-bit architecture forms the basis for a series of high-performance computer systems. Building on almost 10 years of internal research into reduced-instruction-set computer architecture, Alpha AXP emphasizes performance and longevity. The 21064 microprocessor is the first Alpha AXP implementation. Operating at speeds up to 200 MHz, this chip serves as the heart for current systems that offer the highest microprocessor-based performance in the industry.

Edward McLellan

Digital Equipment Corporation

The 64-bit Alpha AXP architecture[1,2] and the first implementation DECchip 21064 microprocessor grew out of a multiyear effort at Digital. Our aim was to develop a computer family capable of leadership performance for the foreseeable future over a wide variety of applications. Combining strengths in semiconductor technology, computer architecture, hardware design, operating systems, compilers, and applications software, this effort recently delivered a series of such machines. Systems range from personal computers to workstations to supercomputers. Operating systems support includes OpenVMS, full 64-bit Unix (DEC OSF/1), Microsoft Windows NT, and soon, native Novell NetWare. Figure 1 shows the packaged DECchip 21064 microprocessor.

Our rich history of computer design spans 35 years and includes the 16-bit PDP-11 and 32-bit VAX computer families. The Alpha AXP architecture represents a new step in that evolution, one that combines full 64-bit address and data capabilities with principles of RISC architecture. Roots of the AXP development go back to the mid 1980s, when multiple investigations of RISC technology culminated in the definition of the internal Prism architecture. That definition included the valuable experience of completely designing a 32-bit microprocessor.[3]

In 1988, a task force chartered with exploring future enhancements to the VAX concluded that a new architecture would soon be necessary to extend the increasingly cramped 32-bit addressing space of the VAX.[4] The Alpha AXP architecture went much further than that by addressing features such as multiple-instruction issue, multiple processors, and operating system independence. The Alpha AXP architecture benefits from the experience of a broad base of computer architects, hardware designers, and systems and applications software experts. The architecture strives to anticipate future trends as much as it attempts to provide current solutions. This architecture provides flexibility for both architectural evolution and hardware implementation over time in a variety of ways.

For any modern computer architecture to be successful, though, access to a strong semiconductor design and technology base is essential. Our semiconductor development began in the late 1970s with a double-metal NMOS process designed specifically to support high-performance microprocessors. Since then, our designers have developed four generations of CMOS technology. Each generation allows a straightforward path to shrink previous designs for advantages in speed, power, reliability, and cost. The 21064 microprocessor is designed in the CMOS-4 process, which offers 0.75-

0272-1732/93/0600-0036$03.00 © 1993 IEEE

µm feature sizes, three levels of aluminum interconnection, and a 3.3-volt power supply.

A new class of systems requires a tremendous amount of software development. Compatibility with older code was paramount for taking fullest advantage of the new architecture. The software task required more time than the hardware development. Therefore, we staged the hardware design to produce early development units that could assist software efforts. Prior to the final CMOS-4 version of the chip, we produced a CMOS-3 device that offered smaller caches and had no floating-point hardware support. This strategy allowed operating systems and internal developers to run code on actual AXP systems almost two years before the product shipment date. At the same time, the final hardware design got to take advantage of the latest semiconductor process advances.

Compatibility with a large, existing customer base of software also concerned us. Rather than burden the hardware with extensive support hooks, or restrict the architecture with compatibility issues, our design efforts adopted the idea of binary translation. Binary translation involves converting executable programs compiled for one hardware platform to another without requiring recompilation from original source code. For maximum reliability, the challenge also includes a runtime environment to support translation of almost all user mode applications and an interpreter to execute code that is not exposed by the initial translation. All of this must come together to produce a translated image that equals or exceeds the performance of the system being replaced. Translators for both VAX and Mips systems to AXP are available and have been invaluable in the highly successful migration of both system and user programs.

Alpha AXP architecture

Perhaps the most notable difference exhibited by the Alpha AXP is found not in a list of its features, but in its careful avoidance of quick-fix solutions to a variety of problems in computer design. Instead of a segmented address space, which can be more difficult to program, the design provides a large, 64-bit linear address space. Virtually all other computer manufacturers have 64-bit extensions planned, but only one other currently delivers 64-bit hardware. Only Alpha AXP offers a full 64-bit operating system with DEC OSF/1. A clean start rather than extension of a 32-bit architecture avoids hardware baggage that can include "orphan" 32-bit instructions (for example, 32-bit shifts) and other compatibility issues associated with old 32-bit software.

In Alpha AXP, all operations, including a small set for efficient 32-bit support, read and write full 64-bit quantities. To facilitate multiple-issue implementations, the architecture explicitly avoids condition codes, special registers, side effects, suppressed instructions, and branch delay slot instructions. These features fit well with single fetch and issue processors, but only complicate multiple-issue designs and often lead to performance bottlenecks. In a machine executing more than one instruction at a time, a single copy of any resource can become a point of contention. Likewise, a single skipped or forced instruction execution, as in the case of branch delay slots, does not fit well with the notion of a machine that fetches and executes multiple instructions each cycle.

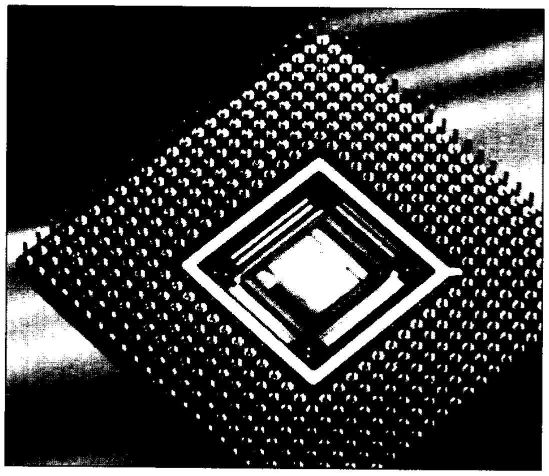


Figure 1. The packaged 21064 microprocessor.

The architecture also avoids direct hardware support for features that, although otherwise useful, are either uncommon or would likely limit the performance of anticipated systems through cycle-time restriction. Instead, the design provides support in a manner consistent with the architectural directions, but using software assistance for full functionality. Examples include the lack of direct-byte load/store instructions and precise arithmetic exceptions. A critical shift and multiplexer path is necessary for byte loads that can threaten cycle time. In addition, byte store operations require costly read-modify-write sequences in systems incorporating common error-correction code protection schemes. Byte writes with such ECC schemes can complicate and slow critical write-back cache designs. Recent experience has shown that some byte oriented codes run much faster when efficiently using the natural 64-bit (8-byte) data width and the byte manipulation support instructions provided.

Where byte operations are required, as in I/O support routines, designers of the first Alpha AXP PC have successfully used alternatives such as encoding sizes on address bits and encapsulating the byte manipulation code to port the Microsoft Windows NT operating system without changes to low-level driver code. In fact, byte manipulation encapsulation is identical to I/O operation encapsulation, which is necessary for using Intel X86 in and out instructions with high-level

The Alpha AXP architecture is a traditional RISC load-store architecture—all data moves between memory and registers without computation.

languages. A driver that already abstracts I/O operations need not be modified at all for use on Alpha AXP platforms.

As high-end processors such as Cray have done for years,[5] hardware support for arithmetic traps is imprecise with respect to the instruction stream. An operator can choose precise trap behavior when necessary through the use of the trap barrier instruction, typically during program debugging. General use of the trap barrier, however, can allow precise arithmetic exception behavior at all times without appreciably degrading performance. Measured differences on the 21064 range from less than 1 percent in integer to between 3 and 25 percent in floating-point codes. Advantages in cycle time and design complexity allowed by this approach, however, compare favorably with these differences.

The Alpha AXP architecture is a traditional RISC load-store architecture. That is, all data moves between memory and registers without computation. Computation is done between data in general-purpose registers only.

Operating system independence. Anticipating the need to support multiple operating system ports, a set of privileged software subroutines, called PALcode, can tailor some of the lowest level hardware-related tasks unique to a particular operating system. For flexibility in service, interruptions, exceptions, context switching, memory management, and error handling all have controlled entry points in PALcode. Neither the hardware nor the operating system then is burdened with a bad interface match, and the architecture itself is not biased toward a particular computing style. In addition, since PALcode mediates all access to physical hardware resources, including physical main memory and memory-mapped I/O device registers, users can also tailor the code for special purpose environments such as real-time and highly secure computing.

Addressing. Virtual addresses are a full 64-bits wide, although subsets are allowed. The AXP employs little-endian byte addressing, similar to Intel X86 and VAX computers. Systems can access both big- and little-endian data using the byte manipulation instructions with a single instruction modification to the sequence. In fact, Digital and its partners are building both big- and little-endian systems and software. Implementations may subset the address width, to a minimum of 43 bits with sign extension, but must check all 64 bits for compatibility with future systems. The AXP does virtual-to-physical-address mapping on a per-page basis, and its pages are 8 Kbytes with future expansion defined.

Data types. The fundamental unit of data is the 64-bit quadword, although the architecture also supports 32-bit longwords. Floating-point data types include both VAX and IEEE formats in both 32-bit single- and 64-bit double-precision formats. An extended-precision floating-point format is not included, but the designers have anticipated expansion by reserving a function field. Byte and word (16-bit) data types are not supported by direct load-and-store instructions but by short sequences of instructions. They can be manipulated in registers using normal arithmetic and the byte manipulation instructions.

Processor state. The hardware processor state includes separate 32-entry by 64-bit integer and floating-point register files. R31 is always zero in each file. Completing the required state are a longword-aligned, 64-bit program counter, floating-point control register for IEEE compliance, and a pair of lock registers for multiprocessor support. If the FETCH/FETCH_M instructions, or VAX-translated images are supported, additional hardware state is required.

The Privileged Architecture Library (PAL) gives designers the option of adding PAL state to the existing hardware state. The PALcode completes the architectural definition in an operating-system specific way. The hardware designers determine the implementation of PAL state, which can range from full hardware to full software or a combination of the two based on design constraints. Typical PALcode state include kernel stack pointer, user stack pointer, and translation look-aside buffers as well as a process-unique value for threads and a processor number for multiprocessor dispatch.

Instruction formats. As shown in Figure 2, the architecture uses four fundamental instruction formats: operate, memory, branch, and CALL_PAL. All instructions are 32-bits wide and contain zero to three register fields. To minimize register file port requirements, register B (RB) is never written and register C (RC) is never read.

The operate format includes arithmetic, logical, shift, and byte manipulation instructions. Scaled add/subtract and compare bytes instructions allow efficient operation on arrays and strings. Conditional move instructions for both integer and floating-point data, which test one input operand and optionally transfer data from another, remove branches in favor of a single instruction. Rather than using a single condition code location, the compare instructions write directly to any general-purpose register. They include an unsigned comparison operation for extended-precision arithmetic. There is no integer divide instruction. Where necessary, a 128-bit multiply can be used for emulation. The architecture enables traps on a per-instruction basis to avoid mode registers, and provides some longword (32-bit) operations for compatibility.

Memory format instructions are mainly loads and stores, but also include some additional instructions. Loads and stores use two registers, specifying a base address and a data source or destination. The effective address calculation sign extends a 16-bit displacement to 64 bits and adds the 64-bit base register value. The architecture also provides load-and-store operations for longword (32-bit) quantities. General operations move aligned data quantities and trap on unaligned references, but instructions that mask the unaligned address bits and do not trap are available for use with the byte manipulation instructions. Calculated jump instructions also use the memory format; these instructions determine the target address directly from the base address without using the displacement field. The unused bits, however, are defined as hints for hardware prefetching mechanisms to improve pipeline efficiency. The additional hint information designates a likely target, allowing the hardware to continue fetching before the true target is available from the register file. If the hint is wrong, a misprediction restart costs no more time than if the hardware stalled waiting for the true address. A pair of load address instructions also use the memory format and allow a convenient way to create large constants using the 16-bit displacement field.

The design provides branch format instructions for both integer and floating-point data. These instructions test a single register for an operation-code-specified condition, and either branch to the target or fall through. To calculate targets, the instructions add a 21-bit longword displacement field to the updated program code (PC) resulting in a ±4 Mbyte relative branch range. The large range effectively reduces the need for branches around or to other branches.

The CALL_PAL format instruction contains only a 6-bit operation code field and 26-bit function field. There are no explicit registers because individual instructions can be redefined for specific use. When executed, these instructions dispatch to PAL routines that perform an atomic, or uninterruptable sequence of instructions. CALL_PAL instructions then can serve to emulate complex instruction-set computer functionality.

Shared-memory multiprocessing. Scalable performance was an integral part of the architectural definition. Since cycle time and multi-issues for single processors are likely to become limiting factors over the lifetime of the architecture, multiprocessor support was critical to achieving both performance and longevity goals. The basic multiprocessor interlocking primitive for updating a shared-memory location is a RISC-style load-locked, in-register modify, store-conditional sequence of instructions. If the sequence completes without interruption, exception, or an interfering write from another processor, the store-conditional instruction succeeds and returns status indicating that an atomic update was performed. Otherwise, the store-conditional fails, and the program must branch back and retry the sequence. This mechanism scales well with processor performance and allows multiple simultaneous noninterfering sequences.

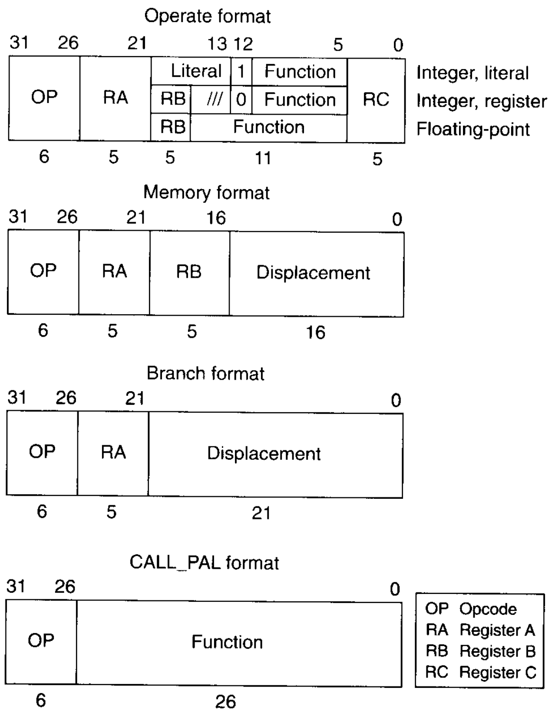


Figure 2. Instruction formats.

The Alpha AXP architecture is the first RISC architecture to offer a relaxed, or weak memory-ordering model. SPARC V9 and PowerPC have more recently announced support for a weak-ordering model. Relaxed ordering implies that the sequence of reads and writes as viewed by another processor need not be in order. Multiprocessors that employ strict-ordering models are possible, but can be subject to performance limitations. For example, if a processor is designed to retry writes that result in errors, a strict-ordering model implies that the retry must complete before any other read or write occurs. This constraint excludes pipelined memory systems that would otherwise allow operations begun prior to the error to complete before, and out of order with the retry.

When strict ordering is required, as is the case in some I/O or multiprocessor synchronization operations, the Alpha AXP architecture specifies a memory barrier (MB) instruction to force serialization of operations. Software then controls serialization, enforcing it only when necessary. The lack of implicit ordering enables a variety of high-performance implementation techniques.

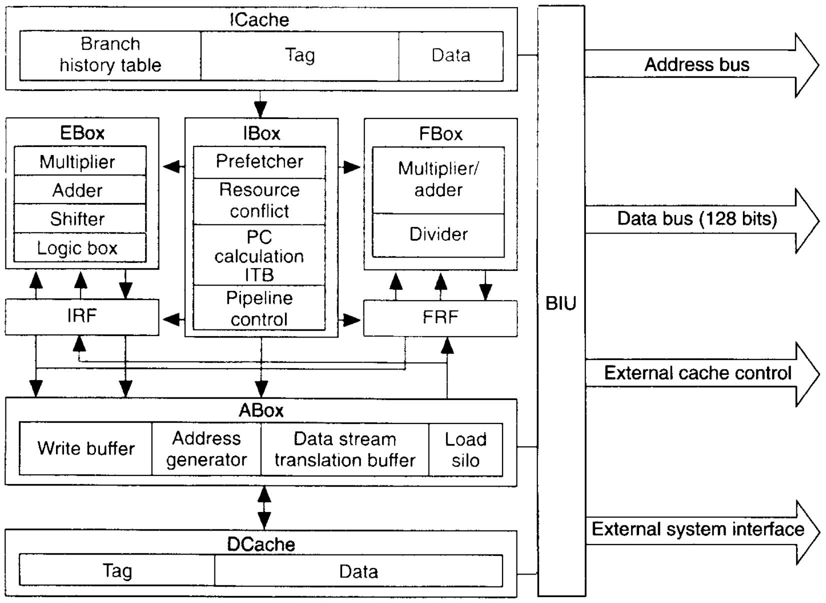


Figure 3. The 21064 block diagram.

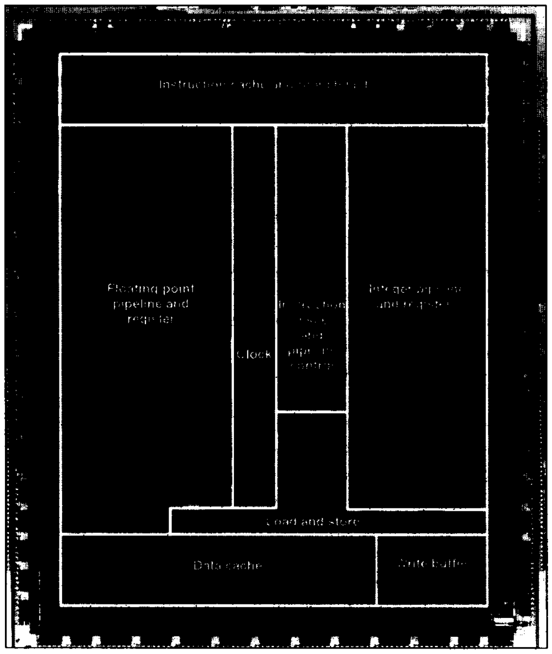

Floating point pipeline and register
Clock
Integer pipeline and register
Load and store
Data cache

Figure 4. The 1.4 × 1.7-cm CMOS 21064 chip.

The 21064

The 21064 microprocessor is the first implementation of the Alpha AXP architecture.[6,7] This 1.4 × 1.7-cm CMOS chip incorporates 1.68 million transistors using a 0.75-μm, three-metal process. Figure 3 shows the chip's block diagram and Figure 4 shows a photograph of the chip itself. The design provides high performance through superscalar (two instruction issue) operation with an exceptionally high frequency internal clock cycle. Production chips and systems are available at clock speeds up to 200 MHz.[8] Despite the fast internal cycle time, the 21064 provides a flexible external interface that can easily accommodate a range of system designs. These designs are well within the range of standard interface devices due to the on-chip programmable system clock. System designs can run the CPU at from two to eight times the system clock frequency. Initial system designs range from PC to workstation to supercomputer class. They offer the highest microprocessor-based system performance in the industry as measured by the System Performance Evaluation Corporation (SPEC) suite of benchmark programs. (SPEC benchmarks, a series of programs measuring both speed and throughput, have become the standard for measuring computer performance.)

Cycle time implications. Overall performance involves many factors, but the two controlled primarily by the microprocessor designer are cycle time and the amount of work or instructions completed per cycle. Experience developing an earlier short cycle-time microprocessor[1] combined with simulations of possible design alternatives reinforced the RISC conclusions that the more potent lever was cycle time reduction. Based on the aggressive cycle time goal of typical parts at 150 MHz and fast parts at 200 MHz, the design supports two levels of cache hierarchy. The high speeds require on-chip caches to supply data and instructions at the cycle time rate (5 ns). However, die size and speed constraints limit the maximum size of that cache, which then can reduce performance. A large off-chip, second-level cache can mitigate this effect. The combination provides better overall performance and promises a greater rate of improvement as process density increases. The relative performance gain in increasing a small cache is greater than made available by increasing a large cache. In addition, a small on-chip cache can scale with CPU cycle time much better than a large off-chip cache, allowing a design to take full advantage of advances in process technology.

To maintain the cycle time goals, we carefully evaluated all potential features—even a slight cycle time slip would likely cost more performance than the feature could give us. This philosophy extended through the ongoing architectural definition to avoid requirements that could limit implementation. For example, we postponed the decision regarding inclusion of the scaled add-and-subtract instructions in the architecture until we could demonstrate that implementations would not incur an adverse internal cycle time hit.

Dual issue. Dual-issue capabilities also exhibit cycle time influences. Rather than allow complete dual-issue flexibility, which only improves performance by an approximate increment of 2 percent over the final design, our design slightly restricts the instruction pairs for multiple issue. Compilers can group dual-issue operations in pairs when possible, but excess code expansion arises due to instruction padding if they are required to always align within the pair as well. When necessary, the hardware swaps pairs capable of dual issue. Hardware also serializes pairs that cannot dual issue to streamline internal control and data paths. All important pairs allowed by combinations of functional units can dual issue.

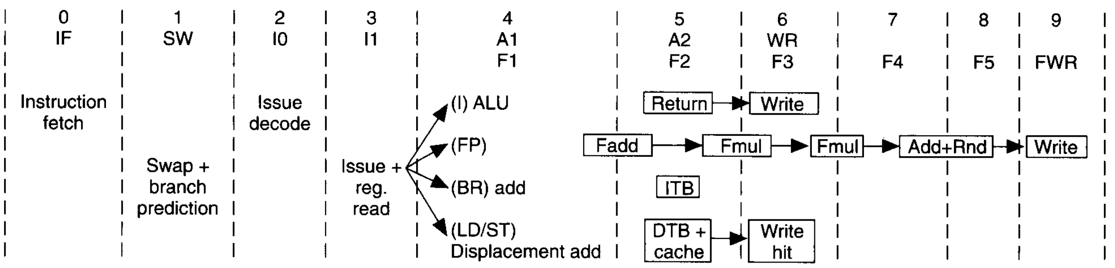
Since load-and-store operations predominate in RISC codes, the design provides a separate address unit to allow load and stores to execute with operate instructions. Table 1 shows the general instruction pairings for dual issue. There are only two exceptions to these rules. Branches cannot dual issue with stores of the same format because they share a register file port, and stores or branches cannot dual issue with operates of a different format because they share an instruction bus. For example, integer stores cannot dual issue with floating-point operates.

Table 1. General dual-issue rules.

Instruction A	Instruction B
Integer operate	Floating-point operate
Load/store	Operate
Branch	Load/store/operate

Pipeline. As shown in Figure 5, the integer and floating-point pipelines are, respectively, seven- and 10-stages deep. The first four stages are common to the two pipes and comprise the instruction fetch-and-issue section of the chip. Each stage can process up to two instructions in parallel. In the instruction fetch (IF) stage, the processor fetches a pair of instructions each cycle from the 8-Kbyte instruction cache. The swap (SW) stage controls instruction prefetching, doing branch prediction and cache index calculation as well as the swap or serialization operation described earlier. The issue-zero (I0) stage checks for intrafetch dependencies. This stage also completes the decoding and set up for the issue-one (I1) stage, which includes the register conflict detection and instruction issue to the datapath function units. Both the integer and floating-point register files are read in the I1 stage to supply data to integer, floating-point, load/store, and branch calculation units as shown.

The integer calculation pipeline writes results back to the integer register file in pipe stage 6, while floating-point calculations write to the floating-point register file in stage 9. Pipe stage 4 resolves branches, after which the prefetcher is redirected, resulting in a four-cycle misprediction penalty.



A1	ALU-one	IF	Instruction fetch	ITB	Instruction translation buffer
BR	Branch	I0	Issue-zero	LD/ST	Load/store
DTB	Data stream translation buffer	FP	Floating point	SW	Swap
F1	Floating-point one	FWR	Floating write	WR	Write back

Figure 5. Pipeline.

Branch instructions and condition codes

Branches pose an increasingly severe problem in heavily pipelined and superscalar computer designs as a pipeline flush costs more potential instruction issue slots. The Alpha AXP architecture offers an advantage in handling branch instructions. Most computer architectures include condition codes to hold the result of arithmetic operations that can later be used to determine the outcome of branches. Unfortunately, the condition code register itself can become a point of resource contention if you assume that multiple instructions are executed simultaneously.

Recognizing this problem, some architectures offer combined compare-and-branch instructions that both reduce the number of instructions and eliminate the intermediate condition code storage. These instructions, though, force the arithmetic operation to be performed immediately before the branch. Arithmetic operations typically are one of the last pipeline stages, which therefore increases the misprediction penalty. In a superscalar implementation, it may be possible to resolve the branch decision early and overlap its execution with unrelated instructions.

The Alpha AXP architecture does not use condition codes. Instead, it resolves all branches based on the test of a single register. In effect, any register can hold branch condition information, eliminating the resource problem. In addition, since the branch need only test a single register, all that is required to resolve any branch immediately after reading the register file are the most- and least-significant bits, and a zero detection, which could be stored at register write time.

Loads that hit in the 8-Kbyte on-chip data cache write the associated integer or floating-point register file in pipe stage 6, simultaneously with integer calculations. The chip checks data cache misses in the large off-chip backup cache under complete CPU control. It only generates a system read block command if both caches miss. Instruction cache misses also get checked in the backup cache and fetch a second sequential 32-byte block into an on-chip streaming buffer. If the CPU reports an additional instruction cache miss that hits in the stream buffer, the data is simply moved into the cache and the next block is fetched in parallel with instruction execution.

Despite the apparent two-cycle delay for integer calculations, data is available after the first cycle in most cases. As shown in Figure 6, an extensive set of data bypass paths allows many back-to-back dependent operations to execute at fully pipelined speeds. In all, the chip uses 45 different bypass paths to minimize the effect of pipeline latency on dependent operations. All register conflict checking is done in hardware. Up to 22 operations thus can be in various stages of completion simultaneously, including 14 within pipeline stages 0 to 6, three in the extended floating-point pipe, three outstanding load misses, a floating-point division, and an integer multiplication.

Branch handling. With such a deep pipeline, branch handling is particularly important. The Alpha AXP architecture reduces branches through the use of the conditional move instructions and also includes hints for hardware-assisted branch prediction. The 21064 uses these hints and includes additional features. The chip can statically predict conditional branches using the sign of the displacement field to predict backward branches as taken and forward branches as not taken. In addition, the 21064 contains a 2K by 1-bit branch history table for dynamic prediction that provides approximately 80 percent accuracy for most programs.

The instruction prefetcher also contains a last-in, first-out stack of recent subroutine return addresses used to predict return paths for subroutines. The stack is repaired during pipeline flushes and therefore allows two additional benefits. As explained in the box, branch misprediction can become a performance issue at these fast cycle times due to the length of the pipeline. The chip uses the subroutine return stack to source the alternate, and correct, branch path one cycle earlier than the program counter datapath could provide it upon branch misprediction. In addition, the stack is used to accurately predict the return address for exceptions, since most exceptions (such as translation look-aside buffer miss) as measured by frequency, return to the original routine.

Integer unit. The integer register file contains thirty-two 64-bit general-purpose registers. It provides six ports, including four reads and two writes to allow the parallel execution of both integer calculations and load, store, or branch operations. The data path includes dedicated adder, shifter, multiplier, and logic units. Both the logic unit and adder provide results in one cycle. The shifter requires two cycles for results but is fully pipelined. The multiplier is not pipelined for area savings, and it supports the Alpha AXP UMULH instruction that returns the upper 64 bits of a 128-bit product for extended-precision operations and integer division support.

Floating-point unit. The floating-point unit combines maximum throughput with short latencies. It contains a 32-entry by 64-bit register file with three read and two write ports. The multiplier uses a radix-8 Booth algorithm in a fully pipelined two-way interleaved array. The rounding operation is completed simultaneously with the last adder stage for all operations. For compatibility, this unit supports both VAX and IEEE single- and double-precision data formats. It can initiate new instructions every cycle with dependent operations requiring six-cycle latency. The fast cycle-time goal translates into longer total latency as measured in cycles. For many floating-point codes, though, throughput and optimized compiler algorithms deliver exceptional performance as demonstrated by the >200 SPECfp92 values measured on DEC 10000 systems.

Address unit. The address unit performs all load-and-store operations. To do so in parallel with other units, it contains a dedicated displacement adder rather than sharing the integer calculation adder. The address unit contains a 32-entry data translation look-aside buffer. Entries can be used to translate single pages or groups of contiguous pages. The unit allows ranges of 8 Kbytes, 64 Kbytes, 512 Kbytes, or 4 Mbytes for each entry. The address unit can process up to three outstanding load misses to avoid blocking nondependent instructions.

Store instructions aggregate data in a 4-entry × 32-byte write buffer. The write buffer reduces off-chip bandwidth requirements by merging data from adjacent stores. It also allows early service for critical load data by temporarily delaying stores that would have otherwise occupied the data bus. The AXP architecture allows this reordering to improve performance; the memory barrier instruction can inhibit the reordering when necessary. The address unit allows back-to-back load-and-store operations in any order by accessing the current store tag with the last store data in separate cache tag and data arrays. The address unit supports wrapped reads (target word first) on primary cache misses, while filling 32-byte cache blocks. This minimizes the latency incurred when the return data is immediately needed. If the pipeline was blocked waiting for load data, it can continue as soon as the target word is returned at the same time that it fills the remainder of the cache line in the background.

Pipeline control/exceptions. The pipeline can be interrupted for a number of reasons including branch mispredictions, instruction cache misses, and interruption and exception conditions. Either a conditional branch or calculated jump instruction can produce branch mispredictions. They do not require hardware unrolling because both mispredictions are detected before the write back stage. Interruptions and exceptions cause traps to PALcode. These traps resemble mispredictions but also drain the pipeline before executing the new flow. Hardware reduces the idle pipeline time by overlapping the drain with the prefetch of the new instructions.

Privileged Architecture Library. A unique feature of the AXP architecture is the privileged architecture library. The PAL routines used with the 21064 allow flexibility in the definition of a hardware/software interface by assisting some hardware-related tasks and completely emulating others. For example, the hardware traps to PALcode to parse and service interruptions as well as to update translation look-aside buffers.

A second method of entering PALcode is through explicit CALL_PAL instructions. The chip supports 128 direct hardware dispatches for individual CALL_PAL-type instructions. Upon execution, a CALL_PAL instruction both branches to the selected PAL routine and enables PALmode privileges. These privileges allow PAL routines to access a complete internal state, otherwise hidden from the architected hardware/software interface. PAL can physically access both instruction and data-stream memory by disabling memory mapping, and assure atomic sequences of instructions by disabling interruptions. CALL_PAL routines support a variety of operations, generally too complex to be implemented in hardware. For example, the PAL provides the swap process context operation as a CALL_PAL instruction that can be unique for each operating system.

PALcode routines can be completely customized because they use a superset of the AXP instruction set. The architecture exclusively reserves five operation codes for PAL which allows each implementation to define these instructions for best use. A hardware implementation and PAL routines form a matched set that together make up the operating system and programmer interface. Since only the interface must remain consistent between implementations, future chips have complete flexibility to make low-level hardware trade-offs without impacting existing code. In addition, designers can redefine this interface to meet the needs of each operating system. The 21064 currently supports three operating systems with individual interface requirements.

Performance tuning. In the first production implementation of any new architecture, performance feedback is important for both software tuning and future hardware projects. With improving integration and cycle times, this information is increasingly difficult to obtain at the pin interface, or is so far removed from program execution that it is of little value. The 21064 contains two methods of providing more relevant information directly from running systems.

First, the Alpha AXP architecture offers a cycle counter capable of recording absolute and process virtual times at very fine intervals (single cycle on 21064). Its use, however, requires code modification. Second, the 21064 contains on-chip performance counters that count selected events and produce interruptions upon counter overflow. Through the use of PALcode and operating system utilities, these counters can collect data on unmodified applications. The design provides two counters that can select from a variety of sources

```
SUBQ     R0, R5, R1         ; cycle 0
ADDQ     R1, R2, R3         ; cycle 1
CMPLT    R3, #500, R4       ; cycle 2
BEQ      R4, Target         ; cycle 3
```

Figure 6. Multiple bypass paths allow many instructions to execute in sequential cycles despite the deep pipeline.

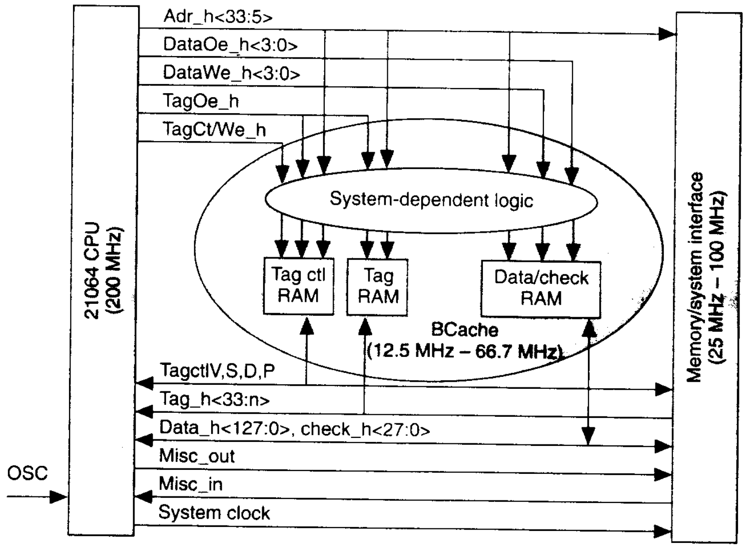


Figure 7. Chip interface. Three time domains exist in a typical system, with the CPU executing out of the internal caches at up to 200 MHz, the backup cache loop ranging from one third to one sixteenth of the CPU frequency, and the remaining system logic executing at one half to one eighth of the CPU frequency.

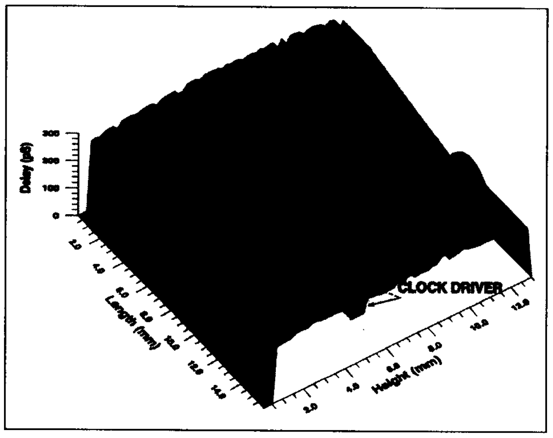


Figure 8. Alpha clock delay.

including instruction issue, pipeline stalls, and instruction mix as well as cache miss, branch misprediction, and two input pins that can be further broken down to gather external system data.

Interface. To accommodate a range of system designs, the interface is extremely flexible. See Figure 7 for a drawing of the chip interface. Although the chip operates with a 3.3-volt power supply, it can also interface with more common 5-volt logic. Data bus widths of 128 and 64 bits for reduced-cost systems are available, and the system clock speed can be set at any submultiple of the CPU speed from one half to one eighth. We have designed a 25-MHz EISA bus-based system that uses a one-to-six CPU clock divisor using standard PC interface parts. Even at 150-MHz CPU operation, cooling only requires a heat sink.

The chip supports up to 16 Gbytes of physical memory and an optional second-level back-up cache ranging in size from 128 Kbytes up to 16 Mbytes. The cache access path is combinatorial. It can support a variety of static RAM speeds, selectable through an on-chip register at 3 to 16 times the CPU clock cycle time. The interface supports parity or ECC protection. In the event of a back-up cache miss, the chip issues a system command to perform the necessary read or write operation. System commands interact with board logic in a handshake manner and operate at the selected system clock multiple, not the CPU clock speed.

Multiprocessing. The chip provides multiprocessing support in a flexible manner. Through valid, dirty, and shared (write protected) tag control signals, we can configure a write-back external cache on the 21064 to support a variety of cache coherence policies. Digital's systems use a conditional write-through policy although the chip can also support an ownership policy. Internal cache invalidate controls allow a system to maintain coherence of the internal cache. Through pin support for maintaining a backmap, the system can also implement cache invalidate filtering based on the contents of the primary cache if desired.

Clocking. Since it operates at speeds of up to 200 MHz, designing the 21064 required us to rethink many aspects of CMOS circuit design. Most critical was the decision to use a single-wire, two-phase clocking scheme. This type of clocking helps to eliminate dead time between phases. To ensure correct latching operation, though, the clock edge rate had to be extremely fast to avoid race-through of the latch data. Our solution included a very large clock driver with a final stage containing 10-11/64-inch-wide PMOS and 4-5/64-inch-wide NMOS devices. The driver switches the clock load in

Table 2. Measured performance under OpenVMS AXP V1.

	DEC 3000 Model 400	DEC 3000 Model 500	DEC 4000 Model 610	DEC 7000 Model 610	DEC 10000 Model 610
CPU frequency (MHz)	133	150	160	182	200
BCache size (Kbytes)	512	512	1,000	4,000	4,000
TPC-A (Rdb v.6)*	NA	NA	NA	302	NA
SPECint92	63.8	72.6	81.2	94.8	104.3
SPECfp92	112.2	126.0	143.1	182.1	200.4
SPECrate_int92	NA	NA	NA	NA	NA
SPECrate_fp92	2,631.6	2,967.4	3,317.1	4,126.0	
2 process			6,214.5	8,135.1	
3 process				11,859.8	
4 process				15,739.4	17,187.2
Linpack 100 x 100**	26.4	30.2	36.3	38.6	42.5
1,000 x 1,000**	90	107	114	141	155
Perfect BM suite†	18.1	20.4	22.9	26.0	28.6
Cernlib (CERN units)	16.9	19.0	21.0	23.6	26.0
Livermore loops	18.7	21.3	22.9	25.6	28.1
Slolom patches	5,644	6,022	6,384	7,018	7,248
SPECint89	65.8	73.5	83.7	95.1	104.5
SPECfp89	150.6	169.9	188.4	244.2	268.6
SPECmark89	108.1	121.5	136.2	167.4	184.1

* Transactions per second
** Project linear scaling for microprocessor configurations
† Geometric mean

0.5 ns, drawing a peak switching current of 43A. We extensively analyzed the clock both to ensure the integrity of the supply voltage during switching and to guarantee that the adjacent latches saw very little clock skew for proper operation. To address the supply voltage problem, we added 0.13 μF of on-chip decoupling capacitance. This was sufficient to supply all the charge associated with a complete CPU cycle with only 10-percent degradation of the supply voltage.

The skew problem required analysis of the 1.2-million-element RC clock grid. We used a simulator derived from the Carnegie-Mellon AWEsim circuit simulation program to examine the grid at 10-ps intervals. As shown in Figure 8, a monotonic clock wave propagates outward from the center clock driver. Any inward movement of the wave or large discrepancies would indicate potential timing hazards in the design. Such analysis proved necessary for correct operation of the chip, as early simulation results did, in fact, identify errors in the grid connection.

System performance

Performance tuning is an ongoing effort with work continuing in both compiler algorithms and optimizations as well as system tuning. However, as shown in Tables 2 and 3 (next page), initial data demonstrate excellent performance. These tables include results over a variety of commonly used benchmarks under both OpenVMS AXP V1 and DEC OSF/1 V1.2. Both SPECint92 and SPECfp92 values establish a new high point for system performance. Linpack among other floating-point intensive benchmarks demonstrates impressive floating-point capability, with the 1,000 x 1,000 values representing greater than 75 percent of the peak theoretical rate. Integer performance is equally impressive—the DEC 3000 Model 500X workstation achieves a SPECint92 rating of over 110. The more recent SPECrate benchmarks show nearly linear scaling across all multiprocessor configurations as demonstrated by the SPECrate data run under OpenVMS. *[For a fuller description of SPEC benchmarks, see H.G. Sachs et al., "Design and Implementation Trade-offs in the Clipper 400 Architecture," IEEE Micro, Vol. 11, No. 3, June 1991, pp. 18-21, 74-80.—Ed.]*

Table 4 shows results of benchmark performance for translated VAX images. The goal of the translation effort was to match or exceed the performance of similarly priced VAX systems. We met this goal, and many VAX user applications

Table 3. Measured performance under DEC OSF/1 V1.2.

	DEC 3000 Model 400	DEC 3000 Model 500	DEC 3000 Model 500X	DEC 4000 Model 610	DEC 7000 Model 610	DEC 10000 Model 610
CPU frequency (MHz)	133	150	200	160	182	200
BCache size (Kbytes)	512	512	512	1,000	4,000	4,000
SPECint92	74.7	84.4	110.9	94.6	103.1	116.5
SPECfp92	112.5	127.7	164.1	137.6	176.0	193.6
SPECrate-int92	1,763.0	1,997.0	2,611.0	2,198.0	2,571.7	2,765.0
SPECrate-fp92	2,662.0	3,023.0	3,910.0	3,247.0	4,178.7	4,368.4
Linpack 100 x 100	26.0	29.6	39.8	35.0	36.9	40.5
1,000 x 1,000*	91.7	103.5	133.2	110.1	137.8	151.1
X11perf (2D Kvec/s)	579.0	662	670	NA	NA	NA
X11perf (2D Mpix/s)	27.2	31.0	31.0	NA	NA	NA
Dhrystones/s V1.1	235,939	266,487	349,785	297,345	330,577	363,743
V2.1	238,095	263,157	333,333	294,117	333,333	357,142
Perfect BM suite**	18.4	20.7	26.2	23.1	26.4	29.2
Cernlib (CERN units)	18.8	21.3	28.9	23.2	26.0	29.0
Livermore loops**	17.4	19.5	26.3	22.3	25.4	27.8
Slalom patches	5,776	6,084	7,134	6,496	6,902	7,248
SPECint89	73.1	83.1	108.6	92.9	107.4	116.2
SPECfp89	141.7	162.8	208.9	177.0	249.8	275.8
SPECmark89	111.1	126.1	160.8	137.3	175.5	192.1

*64 bit, double precision
**Geometric mean

Table 4. Measured performance of VAX translated code under OpenVMS.

	DEC 7000 Model 610 Alpha AXP*	VAX 7000/610	DEC 3000 Model 500 Alpha AXP*	VAX 4000/90
SPECmark-89	44.43	42.09	34.37	32.77
SPECint-89	26.71	31.48	20.74	26.71
SPECfp-89	62.36	51.08	48.14	37.55

*Translated with DECmigrate

report between 10 to 15 percent faster times running the translated image on the AXP platform.

Table 5 shows results for translated MIPS images using DECmigrate. As can be seen, performance of the translated image approaches that of native compiled code on the DEC 3000 Model 500 system running DEC OSF/1.

The Alpha AXP architecture marks a new beginning for Digital. The combination of binary translation and PALcode affords the luxury of starting fresh, while maintaining strong compatibility with existing code. The architecture provides for growth in multiple fields as well as flexibility in the implementation of exception handling and operating system specific state. We carefully considered trends in computing such as multiple-instruction issue and multiprocessing to avoid restrictive requirements on future systems. Finally, 64-bit data capability and a 64-bit linear address space, offering over four billion times the range of a 32-bit space, should provide ample power and programming flexibility for years to come.

The goal of the chip design was to deliver the highest performance single-chip microprocessor in the industry, capable of forming the core of a range of systems from PC class to high-end server. Benchmark results attest to that accomplishment. A wide range of system designs are currently available and expanding. In fact, the 21064 chip also forms the basis for the announced massively-parallel processing (MPP) supercomputer from Cray Research, Inc. With the availability of Microsoft Windows NT and native Novell NetWare, the architecture will offer an easy bridge for adding PC applications to the growing list of over 2,500 OpenVMS and Unix applications available today.

Unlike our previous architectures, Alpha AXP is an open computer architecture. Mitsubishi Electric Corp. recently joined over 35 other corporate Alpha AXP partners at all levels of design integration and will offer a second source for the 21064 as well as new designs in the future. Within Digital, we are currently designing multiple chips. High-performance parts include a speed enhanced version of the 21064 that will double the internal cache sizes and a next-generation, quad-issue processor. Design of a high integration device for reduced system cost is also in progress.

With the demonstrated performance of the current designs, availability of software, and a growing list of suppliers, the Alpha AXP architecture is well positioned for the future.

Acknowledgments

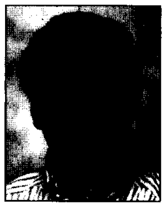
Many people contributed to the design of the Alpha AXP architecture, with a large majority of the current definition attributable to principal architects Dick Sites and Rich Witek. The 21064 design and verification team consisted of Dan Dobberpuhl, Rich Witek, Randy Allmon, Rob Anglin, Dave Bertucci, Sharon Britton, Linda Chao, Rob Conrad, Dan Dever, Bruce Gieseke, Soha Hassoun, Greg Hoeppner, John Kowaleski, Kathy Kuchler, Maureen Ladd, Mike Leary, Liam Madden, Ed McLellan, Dirk Meyer, Jim Montanaro, Don Priore, Vidya Rajagopalan, Sri Samudrala, Sri Santhanam, Scott Kreider, Stephan Meier, Andy Payne, Homayoon Akhiani, Mike Kantrowitz, and Dave Conroy, as well as others, who devoted tremendous amounts of their time and talents to ensure the successful completion of the design. Jim Montanaro, Mark Coiley, John Faricelli, and John Kowaleski created the clock analysis and graphical output.

Table 5. MIPS translated code performance on DEC 3000 Model 500 (DEC OSF/1).

	Native	DECmigrate
SPECint92	84.4	68.6 (81%)
SPECfp92	127.7	81.9 (64%)

References

1. R.L. Sites, "Alpha AXP Architecture," *Comm. ACM*, Vol. 36, No. 2, Feb. 1993, pp. 33-44.
2. *Alpha Architecture Reference Manual*, R.L. Sites, ed., Digital Press, Bedford, Mass., 1992.
3. R. Conrad et al., "A 50-MIPS (Peak) 32/64-Bit Microprocessor," *Digest Tech. Papers, IEEE Solid-State Circuits Conf.*, Piscataway, N.J., Feb. 1989, pp. 76-77.
4. *VAX Architecture Reference Manual, Second Ed.*, R. Brunner, ed., Digital Press, Bedford, Mass., 1991.
5. *Cray-1 Computer System Reference Manual*, Form 2240004, Cray Research, Inc., 1977.
6. D. Dobberpuhl et al., "A 200-MHz 64-Bit Dual-Issue CMOS Microprocessor," *IEEE J. Solid-State Circuits*, Vol. 27, No. 11, Nov. 1992, pp. 1555-1567.
7. D. Dobberpuhl et al., "A 200-MHz 64-Bit Dual-Issue CMOS Microprocessor," *Digest Tech. Papers, IEEE Solid-State Circuits Conf.*, Feb. 1992, pp. 106-107.
8. *DECchip 21064-AA Microprocessor Hardware Reference Manual*, Digital Press, Bedford, Mass., October 1992.
9. R.L. Sites et al., "Binary Translation," *Comm. ACM*, Vol. 36, No. 2, Feb. 1993, pp. 69-81.



Edward McLellan is a principal engineer in Digital's Semiconductor Engineering Group. He has contributed to multiple chip designs, including PDP-11 and VAX floating-point processors, prior to the Alpha project. He was a coarchitect of the 21064 processor.

McLellan received a BS in computer and systems engineering from Renssalaer Polytechnic Institute and has done graduate work at the University of Massachusetts. Currently, he is working on a third-generation Alpha AXP processor.

Direct any questions concerning this article to the author at the Digital Equipment Corporation, HLO2-3/J03, Hudson, MA 01749; mclellan@ad.enet.dec.com.

Reader Interest Survey

Indicate your interest in this department by circling the appropriate number on the Reader Service Card.

Low 156	Medium 157	High 158