Alpha Architecture: Hardware Implementation and Software Programming

Implications

Derrick Meyer
Digital Equipment Corporation, Semiconductor Engineering Group
Hudson, MA 01749

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

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

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

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

The instructions interact with each other only by one instruction writing to a register or memory location and another instruction reading from the same register or memory location. This use of resources makes it easy to build implementations that issue multiple instructions every CPU cycle. The first Alpha implementation issues two instructions per cycle.

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

RISC architectures typically provide full hardware support for frequent and therefore performance sensitive operations, while striving to ease the task of building high performance hardware implementations by relegating support for less frequent or complex operations to software. Alpha extends this design philosophy in a few important ways.

1. BYTE MANIPULATION

The Alpha architecture does byte shifting and masking with normal 64-bit register-to-register instructions. Alpha does not include single-byte load/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 implementations 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.

2. ARITHMETIC TRAPS

Alpha lets the software implementer 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.

3. 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, and interfering write from another processor, or a CALL_PAL instruction, then the conditional store succeeds. Otherwise, the store fails and the program eventually must branch back and retry the sequence. This style of interlocking works well with very fast caches and makes the architecture especially useful for building multiple-processor systems.
4. INSTRUCTION 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.

5. PALCODE—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 is considered to be more a part of the hardware than the software.
- 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.

A version of PALcode lets implementations run the full VMS operating system by mirroring the VAX VMS features. The VMS PALcode instructions let Alpha run VMS 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.

PALcode makes Alpha especially attractive if multiple operating systems are a goal.

6. PROGRAMMING LANGUAGES

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

1. 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.

2. 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.