10.8 Instructions unique to Alpha

(Original section[1])

The Alpha was intended to be an architecture that made it easy to build high-performance implementations. Toward that goal, the architects originally made two controversial decisions: imprecise floating-point exceptions and no byte or halfword data transfers.

To simplify pipelined execution, Alpha does not require that an exception should act as if no instructions past a certain point are executed and that all before that point have been executed. It supplies the `TRAPB` instruction, which stalls until all prior arithmetic instructions are guaranteed to complete without incurring arithmetic exceptions. In the most conservative mode, placing one `TRAPB` per exception-causing instruction slows execution by roughly five times but provides precise exceptions (see Darcy and Gay [1996]).

Code that does not include `TRAPB` does not obey the IEEE 754 floating-point standard. The reason is that parts of the standard (NaNs, infinities, and denormals) are implemented in software on Alpha, as they are on many other microprocessors. To implement these operations in software, however, programs must find the offending instruction and operand values, which cannot be done with imprecise interrupts!

When the architecture was developed, it was believed by the architects that byte loads and stores would slow down data transfers. Byte loads require an extra shifter in the data transfer path, and byte stores require that the memory system perform a read-modify-write for memory systems with error correction codes, since the new ECC value must be recalculated. This omission meant that byte stores required the sequence load word, replaced the desired byte, and then stored the word. (Inconsistently, floating-point loads go through considerable byte swapping to convert the obtuse VAX floating-point formats into a canonical form.)

To reduce the number of instructions to get the desired data, Alpha includes an elaborate set of byte manipulation instructions: extract field and zero rest of a register (`EXTxx`), insert field (`INSxx`), mask rest of a register (`MSKxx`), zero fields of a register (`ZAP`), and compare multiple bytes (`CMPGE`).

Apparently, the implementors were not as bothered by load and store byte as were the original architects. Beginning with the shrink of the second version of the Alpha chip (21164A), the architecture does include loads and stores for bytes and halfwords.

Remaining instructions

Below is a list of the remaining unique instructions of the Alpha architecture:

- *PAL code*—To provide the operations that the VAX performed in microcode, Alpha provides a mode that runs with all privileges enabled, interrupts disabled, and virtual memory mapping turned off for instructions. PAL (privileged architecture library) code is used for TLB management, atomic memory operations, and some operating system primitives. PAL code is called via the `CALL_PAL` instruction.
- *No divide*—Integer divide is not supported in hardware.
- *"Unaligned" load-store*—`LDQ_U` and `STQ_U` load and store 64-bit data using addresses that ignore the least significant three bits. Extract instructions then select the desired unaligned word using the lower address bits. These instructions are similar to `LWL/R`, `SWL/R` in MIPS.
- *Floating-point single precision represented as double precision*—Single-precision data are kept as conventional 32-bit formats in memory but are converted to 64-bit double-precision format in registers.
- *Floating-point register F31 is fixed at zero*—To simplify comparisons to zero.
- *VAX floating-point formats*—To maintain compatibility with the VAX architecture, in addition to the IEEE 754 single- and double-precision formats called S and T, Alpha supports the VAX single- and double-precision formats called F and G, but not VAX format D. (D had too narrow an exponent field to be useful for double precision and was replaced by G in VAX code.)
- *Bit count instructions*—Version 3 of the architecture added instructions to count the number of leading zeros (`CTLZ`), count the number of trailing zeros (`CTTZ`), and count the number of ones in a word (`CTPOP`). Originally found on Cray computers, these instructions help with decryption.

(*1) This section is in original form.

Provide feedback on this section