The Alpha AXP, part 11: Processor faults

devblogs.microsoft.com/oldnewthing/20170821-00

August 21, 2017

Raymond Chen

There are three types of faults on the Alpha AXP:

- Software faults
- Hardware faults
- Arithmetic faults

Software faults are those triggered by explicit instructions, such as `CALL_PAL`. These are calls that trap into the kernel and are used as the Alpha AXP version of `syscall`. Software faults are raised synchronously, and execution does not proceed past a software fault. Consequently, they are restartable.

Hardware faults are those triggered by things like page faults, hardware interrupts, or software emulation. Hardware faults are not necessarily raised synchronously; execution can proceed past a hardware fault before the fault is generated, but the fault is nevertheless restartable. Even though instructions past the faulting instruction may have already executed, they can safely be executed again.

Arithmetic faults are tricky.

The `ADDx`, `SUBx` and `MULx` instructions can take a `/V` suffix to indicate that the instruction should raise a processor trap if a signed integer overflow occurs.[1] There is a similar suffix that can be applied to floating point operations to trigger an arithmetic fault if something goes wrong in the floating point calculation.

The catch is that the trap is not required to be raised at the point of the operation. The processor is permitted to delay the overflow trap indefinitely, or until you do this:

```
TRAPB
```

The trap barrier instruction tells the processor to raise any overflow traps that are still pending. The previous arithmetic operations need not run to completion; they only need to run far enough to confirm that no overflow has occurred. The processor is allowed to execute

past the `TRAPB` instruction, as long as it can do so without violating the constraints of the `TRAPB` instruction.

In practice, you don't see the `/V` suffix because C-like programming languages don't raise overflow exceptions. They just define integer overflows to wrap, or leave the behavior undefined.

You usually see `TRAPB` instructions at the start and end of a function, and whenever code enters or exits a `__try` block. Basically, it happens any time there is a change to how exceptions are dispatched and unwound.

The fact that overflow traps can occur long after the operation that caused the overflow means that overflow traps are in general not recoverable, because you don't know which register contains the overflowed value. (Indeed, the overflowed value may not even be in a register any more.) If you want your overflow traps to be recoverable, you need to put the `TRAPB` immediately after the instruction that potentially creates the overflow condition.

Okay, so that's overflow. But what about carry? We'll look at that next time.

[1] The presence of overflow detection means that the `L` versions of the instructions are not quite the same as "Perform the `Q` operation, and then sign-extend the low-order 32 bits of the result." The numeric result is the same, but the overflow conditions are different.

Raymond Chen

Follow