How the VAX Lost Its POLY (and EMOD and ACB_floating too)

Mary Payne was the maven of floating point at DEC. A physicist by training and an applied mathematician by choice, she had become involved with various implementations of PDP-11 floating point and was not pleased to discover that, due to lack of algorithmic understanding, the PDP-11 floating point unit could be off by as much as a full LSB (least significant bit). As the VAX project got underway, she was determined that the VAX would have the best floating point that was theoretically possible: “good to the last bit,” as she liked to say.

As part of the VAX architecture effort, Mary helped define two types of floating point instructions that were not present in the PDP-11: polynomial evaluation (POLY) and extended modulus (EMOD). Both had some unique features. First, they operated at extended precision compared to the standard f_floating (24b) and d_floating (56b). POLY did its math to 31b/63b for single/double precision, and EMOD to 32b/64b. Second, each had unique and exacting requirements for when and how intermediate results were truncated. For example, POLY specified that in the central a_n*x+b_n step:

- The multiply result was truncated to 31b/63b prior to normalization.
- The extended-precision multiply result was added to the next coefficient.
- The addition result was truncated to 31b/63b prior to normalization and rounding.

The purpose was to ensure that exactly the same number of bits were developed and retained at each step of the polynomial evaluation, regardless of normalization. This provided the best answer, theoretically.

POLY and EMOD were not popular with the hardware implementers from the outset. Both required building a wider data path for floating point than was required by the normal add, subtract, multiply, and divide instructions. In addition, the masking occurred at “unnatural” places in the flows (prior to normalization), introducing further complexity. Finally, EMOD was only one of two instructions in the VAX architecture with two write operands (the other was EDIV), a fruitful source of microcode errors.

POLY implementations started varying almost from the outset. The 11/780 microcode did masking one way; the 11/780 hardware-based floating point accelerator (FPA) did it differently. The divergences continued to accumulate, as the VAX architectural exerciser – the principal tool for validating a VAX implementation – wasn’t ready until the early 80s.

As a result, the language specifying how POLY worked started become looser and looser. The mask value could be either 31b/63b or 32b/64b. There was no requirement that a microcode implementation, and its matching FPA return the same result. The value of this precisely defined instruction diminished due to the implementation discrepancies, and the math library team started using normal floating point instructions to evaluate polynomials.

When the MicroVAX project was launched in 1982, the team defining the architectural subset had to jettison a fair amount of complexity in order to fit the processor into a single chip (with a second one for floating point). The two areas of contention were the string instructions and the complex floating point instructions. All the string instructions were tossed overboard, except MOVC3 and MOVC5, to reduce the size of the microcode. And EMOD and POLY (and ACB_floating) were jettisoned in order to allow reuse of the J-11’s FPA, which did not have the extra data path bits that would be needed.

However, as Bob Simcoe and the MicroVAX FPU team started working with the J-11 FPA, they concluded that the additional hardware to support the extra floating point instructions wouldn't be all that hard to add. Including these instructions would give the math library team more time to make the transition to a POLY-unsaturated world, and would cause less disruption for customers. So POLY and its friends were added back.

Unfortunately, the FPU team got it slightly wrong. The hardware masked after normalization, not before, creating yet another variation on the canonical answers. This could not be sanctioned, and MicroVAX (and V-11, which used essentially the same hardware) got a waiver. But at this point, POLY was essentially useless, except as a way of telling implementations apart; and the System ID register did that with a lot less effort.

When work began on CVAX in 1984, the VAX community revisited the MicroVAX architectural subset. The conclusion was that more string instructions needed to be there (CMPC3, CMPC5, LOCC, SKPC, SCANC, SPANC), and that POLY, EMOD, and ACB_floating really weren't needed. As kind of a "swing" machine between architectural variants, CVAX added the extra string instructions but retained POLY and its friends. (It got the implementation right, too.) However, starting with Rigel, POLY, EMOD, and ACB_floating were banished from the hardware for good.

VMS doesn't actually care. It has sported a full floating point emulator, for every floating point instruction, since the early 1980s. VMS doesn't really trust the hardware, either: it tests for the presence or absence of each floating point instruction individually. So a system that claimed to be a CVAX yet lacked POLY and friends would work just fine.

The point of this long-winded tale is that problems with POLY and EMOD persist to this day. Because SimH emulates a CVAX, it implements POLY, EMOD, and ACB_floating. Unfortunately, the verification software that was available was for Rigel; all test cases for those instructions are commented out. The SimH code was tested with the much simpler 780 diagnostics and ran just fine. Because modern versions of VMS don't use those instructions, there wasn't much incentive to tests POLY and friends more thoroughly.

Recently, Camiel Vanderhoeven of Migration Specialties ran AXE against SimH. Lo and behold, it turned up bugs in both EMOD and POLY. The EMOD problem was a simple oversight, failing to return a truncated operand on integer overflow. But POLY had multiple problems:

- Masking was done to 32b/64b (CVAX was 31b/63b).
- Masking was done after normalization in the multiply step (the same bug as MicroVAX).
- Masking wasn't done at all in the add step.

With these fixes, POLYF, POLYD, and POLYG now run without error. There's still has one discrepancy in ACBG, out of five million cases; this has to be traced to ground.