

An Exercise in Technical Support for Scientific Computation
for a Lecture Course presented at SUN Microsystems
by Prof. W. Kahan, Univ. of Calif. at Berkeley

Floating-point computation is beset by truths, half-truths and mistakes to an extent little appreciated by the world at large, as this exercise will illustrate. Imagine that you work for CRAY providing technical support for its salesmen and customers, and you have been passed the following letter. This letter is based upon actual events embellished only slightly for didactic effect.

Dear xxxx,

The Fortran function AMOD in both the CFT and CFT77 compilers on a CRAY can give wrong results for certain arguments. Here is an example:

PROGRAM:

```
1      format(2z16)
2      format(3z18)
3      format( ..... )
4      format( ..... )
      read 1, x
      read 1, y
      r = amod(x, y)
      print 4, "x", "y", "r"
      print 2, x, y, r
      print 3, x, y, r
      end
```

INPUT:

```
4009f9fffffffffff
4009fa00000000000
```

OUTPUT:

x	y	r
4009F9FFFFFFFFFFF	4009FA00000000000	BDFAB000000000000
499.99999999999818	500.00000000000000	-1.8189894035458565e-12

This violates the definition of $\text{AMOD}(x, y) = x - \text{AINT}(x/y)*y$, because when x and y satisfy $0 < x < y$, as they do here, then $\text{AMOD}(x, y)$ should give a positive number x instead of the negative number $x - y$. This violation crashed a long benchmark code that had worked perfectly well on an IBM 3090, DEC VAX and SUN III. The code calculates $f(x, y) = \text{SQRT}(\text{AMOD}(x, y)) * \text{EXP}(-x)$ among other things for innocuous values x and y that are always positive.

The CRAY has very peculiar division; $240.0/3.0$ does not give exactly 80.0 and x/y above yields 1.0 exactly instead of 0.99999... as on all other computers and calculators that I have tried. I can tolerate small errors in quotients, but negative values for $\text{AMOD}(\text{positive}, \text{positive})$ is too bizarre to tolerate in an environment where we must share standard Fortran codes that run unexceptionably on all our other machines. Negative values cannot occur on the IBM machines because they chop quotients, so $\text{AINT}(x/y)$ cannot be wrong. The SUN III conforms to the IEEE standard 754, which prescribes an exact remainder, so it cannot malfunction. The DEC VAX has an EMOD instruction in its architecture, which may explain why its AMOD is always correct.

The CRAY is the odd man out here; if you can't fix it, we don't want it.

Yours

What do you recommend that CRAY do?

Discussion of
An Exercise in Technical Support for Scientific Computation
 for a Lecture Course presented at SUN Microsystems

The allegations about the CRAY's peculiar division are correct as of this date; almost all other computers and calculators do always deliver a quotient $x/y < 1$ despite roundoff whenever $0 < x < y$, so they can guarantee that $AMOD(x,y) = x$ exactly in this case. Indeed, on almost all machines with binary floating-point, $AMOD(x,y)$ is just fine so long as x lies between 0 and about $2.9999*y$; on non-binary machines, between 0 and about $1.9999*y$. After that, what happens is not easy to predict.

The trouble starts with the definition of $AMOD(x,y)$. Ideally, this is the remainder r you would get after you carried out long division (as you learned it in school) to compute x/y , but stopped as soon as all the digits of the quotient n that precede the decimal point had been generated. Then $r = x - n*y$ where $0 \leq r/y < 1$ and the quotient n is the integer nearest x/y on the same side as zero; n would equal $AINT(x/y)$ in the absence of roundoff. Ideally r can be computed and represented exactly (unless it underflows, but let's skip that for now) in the same floating-point format as x and y provided all the digits of n are generated correctly; that is a challenge because there can be so many of them when $|x/y|$ is very big, so many that most must be rounded away by $AINT(n)$ to fit into the same floating-point format. Now you see where the trouble begins; the definition

$$AMOD(x, y) = x - AINT(x/y)*y$$

could mean the ideal remainder r , or it could mean the result of computing the rounded values x/y , $AINT(x/y)$, $AINT(x/y)*y$ and $x - AINT(x/y)*y$ in turn. Which is the correct $AMOD$?

Originally, when Fortran was young, all computers computed the version of $AMOD$ contaminated by a few rounding errors; CRAYs still do it that way and they are not alone. Unfortunately, the contaminated $AMOD(x,y)$ can fail to lie where most users expect it, namely between 0 and y . Only if $AINT(x/y)$ is computed too big in magnitude, as could happen on machines like DEC VAXs that round quotients correctly, can $AMOD$ have the wrong sign; but that cannot happen on machines that chop quotients as did all the old IBM 7094s, the old CDC 6400 and 6600 (but not 7600) and many others machines, and as do all IBM 370's and Amdahls nowadays. Perhaps that explains why the wrong sign for $AMOD$ is such a surprise for old-timers and their programs. On the other hand, a contaminated $AMOD(x,y)$ can easily be bigger than y , but not likely by so much as would be obvious.

All machines that conform to IEEE 754/854 should be able to derive an ideal $AMOD$ quickly from the standards' mandatory remainder operation. Machines that lack a hardware remainder can compute it in software like that supplied in the C Math library distributed with 4.3 BSD Berkeley UNIX. Similar but proprietary software resides in DEC's VAX VMS Fortran library; it doesn't use an $EMOD$ instruction, which is a peculiar multiply.

What should CRAY do? CRAY's floating-point hardware is too inaccurate to support an ideal $AMOD$ at a tolerable cost. The least intrusive alternative may be a loop to test $AMOD(x,y)$ for the correct sign (the same as y 's), although only rarely will that test have to correct $AMOD$ (by adding y to it). W. K.