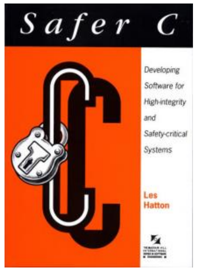
Not-a-Number in IEEE 754 Floating Point Arithmetic

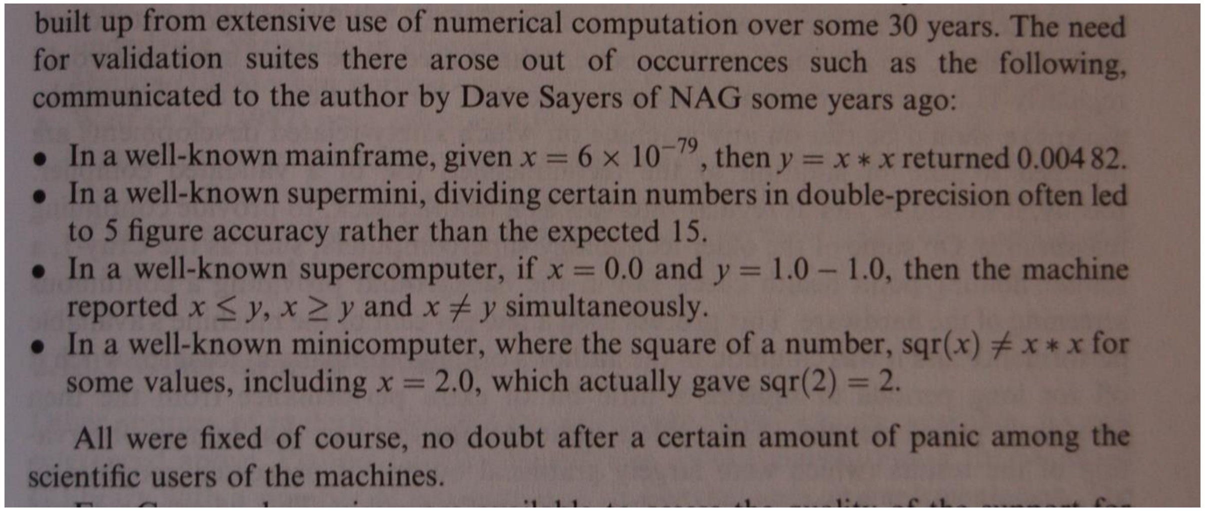
Arnaud Desitter

February 2012

IEEE 754 – History (1)

Before…


Safer C
Developing Software for High-integrity and Safety-critical Systems
Les Hatton



built up from extensive use of numerical computation over some 30 years. The need for validation suites there arose out of occurrences such as the following, communicated to the author by Dave Sayers of NAG some years ago:

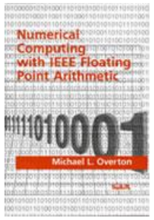
- In a well-known mainframe, given $x = 6 \times 10^{-79}$, then $y = x * x$ returned 0.004 82.
- In a well-known supermini, dividing certain numbers in double-precision often led to 5 figure accuracy rather than the expected 15.
- In a well-known supercomputer, if $x = 0.0$ and $y = 1.0 - 1.0$, then the machine reported $x \leq y$, $x \geq y$ and $x \neq y$ simultaneously.
- In a well-known minicomputer, where the square of a number, $\mathrm{sqr}(x) \neq x * x$ for some values, including $x = 2.0$, which actually gave $\mathrm{sqr}(2) = 2$.

All were fixed of course, no doubt after a certain amount of panic among the scientific users of the machines.

Safer C, *Les Hatton*, McGraw-Hill, 1994, p 83

This standard is arguably the most important in the computer industry, the result of an unprecedented cooperation between academic computer scientists and the cutting edge of the industry

-- *Michael L. Overton*, Numerical Computing with IEEE Floating Point Arithmetic, SIAM, 2004



IEEE 754 – History (2)

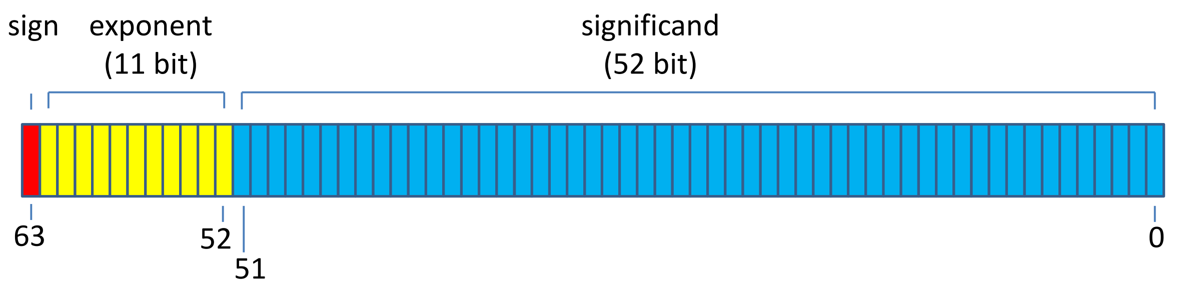
- Originated from the development of Intel i8087 in late 70s. William Kahan was the lead architect and was awarded the ACM Turing award in 1989.

  http://www.cs.berkeley.edu/~wkahan/19July10.pdf

- Accepted in 1981 and formally standardised in 1985: IEEE 754-1985
- Revised in 2008: IEEE 754-2008
- Widely implemented since early 1980s.
- Absolutely ubiquitous from mobiles to GPUs.

NaN encoding

binary64:



- Sign: 0 or 1- this is irrelevant to NaN.
  Exponent: set to 1
  Significand: at least one non 0 bit. This may be used for a payload.
  (If the significand is zero, this represents infinities)

- Bit 51:
  - If 1: this is a *Quiet NaN*.
  - If 0 and non zero significand: *Signaling NaN*.

  (Left unspecified in IEEE 754-1985 and clarified in IEEE 754-2008. PA-RISC (now defunct) and MIPS (now irrelevant for HPC) processors implement the opposite convention.)

Quiet and Signaling NaN

- Quiet NaN are created as:
  1. Indeterminate forms (0/0, etc.). These operations raise the INVALID exception.
  2. Operation resulting in complex numbers (sqrt(-1), etc.). These raise INVALID.
  3. Operation using a NaN. This will *not* raise INVALID.
- Signaling NaN are not created by any arithmetic operation.

  Most operations on Signaling NaN will raise INVALID and return Quiet NaN.

- Therefore:
  – Quiet NaN are used to propagate ”quietly” the results of invalid operations. These raise an initial INVALID exception.
  – Signaling NaN (“SNaN”) are most useful to detect uninitialised values.

Exception

- By default, exceptions are *not* trapped. They set a bit in a status word that can be queried.

  API have been standardised in C99.

- Exceptions can be trapped by setting a flag in a control word.

  This can be done or undone *at run-time*.

  No API has been formally standardised but exist for Windows and Linux.

Enabling exception trapping

```
/* Windows */
#include <float.h>
unsigned int control_word;
_controlfp_s(&control_word, 0, 0);
_controlfp_s(NULL, control_word & ~_EM_INVALID, _MCW_EM);
```

```
/* Linux, Cygwin, FreeBSD, OpenBSD, NetBSD */
#define _GNU_SOURCE 1
#include <fenv.h>
feenableexcept(FE_INVALID);
```

Using SNaN to detect unset variables

Populating floating point variables with SNaN and enabling trapping of INVALID exception is a great way to detect unset variables *at full speed*.

- Commonly offered as an option by Fortran compilers (e.g. GNU Fortran's "-finit_real=snan") but rarely by C++ compilers. A notable exception was the defunct SGI compiler ("*-DEBUG:trap_uninitialized*").
- So in C++ it has to be done manually. This can be made conditional in a production executable under an option.
  - Use "std::numeric_limits<double>::signaling_NaN()".
- Unlike other architectures, a copy of SNaN on x87 (typically Win32) will raise INVALID.
  - "std::numeric_limits<double>::signaling_NaN()" is not usable as it returns a value on the stack.
  - The easiest to set an adequate bit pattern using integer arithmetic (for instance "0xfff00000fff00000LL" for binary64 and "0xff800001" for binary32) .

Conclusions

- Tracking down the origin of NaN in results is best done by enabling exception delivery for INVALID in the FPU the control word.
- This can be enabled or disabled at run-time.
- Populating floating variables with Signaling NaN makes it possible to detect uninitialised variables at full speed.

Acknowledgments

- Photo of William Kahan: http://en.wikipedia.org/wiki/File:William_Kahan.jpg
- Safer C, *Les Hatton* , McGraw-Hill, 1994: http://www.leshatton.org/tag/safer-c/
- Numerical Computing with IEEE Floating Point Arithmetic, *Michael L. Overton*, SIAM, 2004: http://www.cs.nyu.edu/overton/book/