IEEE Arithmetic
Short Reference

Lloyd Fosdick

August 22, 1995

High Performance Scientific Computing
University of Colorado at Boulder

The following are members of
the HPSC Group of the Department of Computer Science
at the University of Colorado at Boulder:

Lloyd D. Fosdick
Elizabeth R. Jessup
Carolyn J. C. Schauble
Gitta O. Domik

Contents

Trademark Notice

- ANSI is a trademark of the American National Standards Institute, Inc.
- Cyber 205 is a trademark of Control Data Corporation.
- Cray is a trademark of Cray Research, Inc.
- Intel, Intel 8087, Intel 80287, Intel 80387 are trademarks of Intel Corporation.
- IBM 3090 is a trademark of International Business Machines Corporation.
- MIPS, MIPS R2010, MIPS R3010 are trademarks of MIPS Computer Systems, Inc.

IEEE Arithmetic Short Reference*

Lloyd Fosdick

13 August 1989
Revised August 22, 1995

Good science depends on reproducibility of results. This principle applies to computations as much as it does to laboratory experiments. In particular, we would like to have a computation done on one machine yield the same results as that computation done on another machine. When the results are not the same their reliability is questionable, and time is wasted in explaining the discrepancies. In scientific computation a lot of work goes into verifying the correctness of the program and its numerical accuracy. If the premises of this work no longer hold, as might be the case if the computation is done on a machine having different numerical characteristics, then this work is no longer valid. Thus good science and economy of effort require that these characteristics, the rules by which computers represent numbers and perform arithmetic, are the same from one computer to another.

Until recently there was no standard for floating-point arithmetic and so machines produced by different manufacturers had different numerical characteristics: different numerical precision, different number ranges, and different rules for rounding numbers. In 1987, after much work by many people, especially Prof. W. Kahan of the University of California at Berkeley, a standard for floating-point arithmetic was approved by the Institute of Electrical and Electronics Engineers (IEEE)and by the American National

*This work has been supported by the National Science Foundation under an Educational Infrastructure grant, CDA-9017953. It has been produced by the HPSC Group, Department of Computer Science, University of Colorado, Boulder, CO 80309. Please direct comments or queries to Elizabeth Jessup at this address or e-mail `jessup@cs.colorado.edu`.

Standards Institute (ANSI). This standard is now followed by major chip manufacturers, including Intel (8087, 80287, 80387 processors) and MIPS (R2010 and R3010 processors). Some computers used in large-scale scientific computation do not follow the standard; for example, the Cray series of computers, the Cyber 205, and the IBM 3090.

Caution: When a manufacturer claims to have IEEE arithmetic in a chip this generally means that the rules for the number of bits used to represent the significand and exponent for single and double precision numbers have been followed, and that "round to nearest" (explained below) is done, but other aspects of IEEE arithmetic may be absent.

1 Single precision

1.1 Format

Thirty-two bits are used to represent a single precision number: one bit is used to represent the sign; eight bits are used to represent the biased exponent (`e`); and twenty-three bits are used to represent the magnitude of the fractional part of the significand (`f`). The format is shown in figure 1: bit (position) 0 is the sign bit, bits 1 through 8 are `e`, and bits 9 through 31 are `f`.

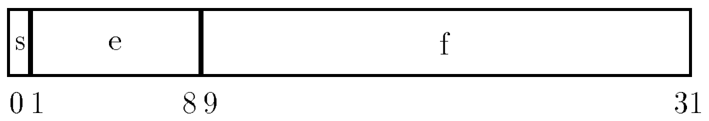

Figure 1: IEEE single precision format: `s` (bit 0) is the sign bit; `e` (bits 1 through 8) is the biased exponent; `f` (bits 9 through 31) is the magnitude of the fractional part of the significand.

The number represented has a sign (+ or −) according to the value of `s` (0 or 1). The bias on the exponent is 127, that is

```
e = p + 127,
```

where `p` is the exponent. The fraction has an implied binary point immediately preceding the string of bits represented by `f`, and the unsigned significand of the number is $1 + \texttt{f}$, as if a 1 were placed before the implied binary point: we sometimes express the unsigned significand as `1.f`.

Thus, if the 32 bits in this format are as shown in figure 2, then the significand is

$$1.10100000000000000000000,$$

and the exponent is

$$-01110001.$$

Thus the value of the number represented is

$$1.10100000000000000000000 \times 2^{-0111\ 0001}$$

The result for the significand should be obvious. The result for the exponent may be less obvious. Notice that 127 (decimal) = 0111 1111 (binary), the biased exponent is `0111 0001`, so the exponent, `p`, is given by

$$p = (00001110) - (01111111) = -(01110001).$$

The same value expressed in decimal form is

$$1.625 \times 2^{-113}.$$

Question: Suppose

```
f =  0101 0000 0000 0000 0000 0000
        e =  1000 1111
```

then what are the values of the significand and exponent; what is the number (binary) represented; what is the number (decimal) represented?

1.2 Special cases: $+0$, -0, $+\infty$, -∞, `NaN`

These special cases are represented with particular values 0 and 255 of `e`. (*Do you see that these are the minimum and maximum values of* `e`*?*)

The value

```
e = 0000 0000
```

is used for representing $+0$ or -0 as illustrated in figures 3 and 4.

The number -0, which may seem a little strange, is explained later.

0	0000 1110	1010 0000 0000 0000 0000 000

Figure 2: Example of single precision number.

0	0000 0000	0000 0000 0000 0000 0000 000

Figure 3: IEEE +0 in single precision.

1	0000 0000	0000 0000 0000 0000 0000 000

Figure 4: IEEE -0 in single precision.

0	1111 1111	0000 0000 0000 0000 0000 000

Figure 5: IEEE $+\infty$ in single precision.

1	1111 1111	0000 0000 0000 0000 0000 000

Figure 6: IEEE $-\infty$ in single precision.

The value

```
e = 1111 1111
```

is used for representing $+\infty$ or $-\infty$ as illustrated in figures 5 and 6.

Patterns with `e` $= 255$ and `f` $\neq 0$ are used to represent cases that are "Not a Number," `NaN`. (The value `NaN` results, for example, from trying to evaluate $0/0$.)

The meaning and effect of these special values is discussed shortly.

1.3 Number range

Since the values 0 and 255 are reserved for special cases, the number range for nonzero, single precision numbers is constrained by the requirement that

```
0 < e < 255 ,
```

hence

$$p\epsilon[-126, +127].$$

(*Do you see why?*).

Therefore the smallest and largest single precision, positive numbers are:

$$1.0000\ 0000\ 0000\ 0000\ 0000\ 000 \times 2^{-0111\ 1110} = 2^{-126} \approx 1.2 \times 10^{-38}$$

and

$$1.1111\ 1111\ 1111\ 1111\ 1111\ 111 \times 2^{11111110} = (2-2^{-23}) \times 2^{+127} \approx 3.4 \times 10^{+38}.$$

Corresponding limits hold for the negative numbers.

When a number is somewhat smaller than the smallest value, 2^{-126}, many systems set its value to zero. However, IEEE arithmetic allows for (but does not require) the representation of this smaller number in a form known as *denormalized*. In this form the unsigned significand is `0.f`. Thus, when denormalized numbers are allowed, the smallest nonzero, single precision number is $2^{-23} \times 2^{-126}$. The use of denormalized numbers is sometimes called *graceful underflow* or *gradual underflow*.

1.4 Numerical precision

Numerical precision is determined by the fact that the rightmost bit in `f` has weight 2^{-23}; i.e., a change of one unit in the least significant place of `f` results in a change in the value of `f` equal to 2^{-23}. To understand the implications of this fact consider the interval defined by the value

$$x1 = 1.00000000000000000000000 \times 2^{-3}$$

on the left end and the value

$$x2 = 1.00000000000000000000001 \times 2^{-3}$$

on the right end. In the usual mathematical notation this interval is $[x1, x2]$. Note that $x1$ and $x2$ can be represented exactly in IEEE single precision, but no value between $x1$ and $x2$ can be represented exactly (*Do you see why?*). Therefore, we must represent a value between $x1$ and $x2$ by either $x1$ or $x2$ depending on how we choose to round (see below). No matter which choice we make, there is an error in the representation of the number and this error does not exceed 2^{-23} in the value of `f`, and 2^{-26} in the number itself. Obviously if more than 23 bits were used for `f` we could represent numbers with higher precision.

Discuss the precision if 27 bits are used to represent `f`*.*

We adopt the convention that numbers that are exactly representable in the IEEE format are called *machine numbers*.

If x is the machine number

$$x = m \times 2^p,$$

where m is the significand, then the next larger machine number is

$$x_+ = (m + 2^{-23}) \times 2^p,$$

and the next smaller machine number is

$$x_- = (m - 2^{-23}) \times 2^p,$$

The spacing of these numbers is

$$x_+ - x = x - x_- = 2^{-23} \times 2^p = 2^{-23+p}.$$

The relative spacing of x and x_+ is defined by the ratio:

$$(x_+ - x)/x \approx 2^{-23} \approx 1.2 \times 10^{-7}$$

Notice that the relative spacing of a machine number and its successor is a constant (approximately), equal to about 2^{-23}. The relative spacing is a measure of the precision of the representation: the smaller the relative spacing, the higher the precision.

Question: Why do we say "approximately" above?

Question: Suppose we are given a single precision representation of 10^{30}*, which we denote by* $flt(10^{30})$*, and we add a small machine number,* ϵ*, to it. What is the smallest value of* ϵ *such that*

$$flt(10^{30}) + \epsilon \neq flt(10^{30})?$$

2 Double precision

2.1 Format

Sixty-four bits are used to represent a double precision number: 52 bits are used to represent `f`; 11 bits to represent the biased exponent; and one bit to represent the sign. The format, analogous to that for single precision, is shown in figure (7).

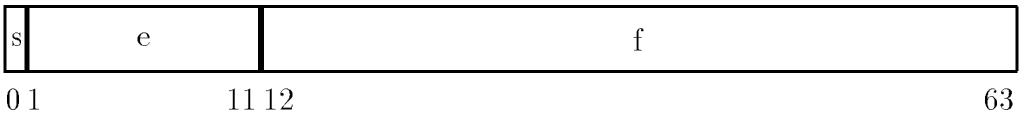

Figure 7: IEEE double precision format: `s` (bit 0) is the sign bit; `e` (bits 1 through 11) is the biased exponent; `f` (bits 12 through 63) is the magnitude of the fractional part of the significand.

The sign bit is interpreted as in single precision. The bias on the exponent is 1023, that is

```
e = p + 1023,
```

where `p` is the exponent.

You should be able to determine the range and relative spacing of the double precision numbers from the format and the previous discussion. The remarks made earlier about denormalized numbers also apply to double precision numbers *mutatis mutandis*.

Question: What are the largest and smallest double precision numbers?

Question: What is the relative spacing of the double precision numbers?

Question: Suppose we are given a double precision representation of 10^{30} *which we denote by* $dflt(10^{30})$ *and we add a small number* ϵ *to it. What is the smallest value of* ϵ *such that*

$$dflt(10^{30}) + \epsilon \neq dflt(10^{30})?$$

Values $+0$, -0, $+\infty$, $-\infty$, and `NaN` are represented in a fashion analogous to that for single precision.

3 Rounding

Arithmetic operations are performed as if they were correct to infinite precision. Rounding occurs when a number in a register of the floating-point unit of the computer is copied into memory, and so must be put into single precision or double precision format.

The default rounding mode is *round to nearest*, and *round to even* when there is a tie. There are of course only two possible choices for the rounded value of a number: the machine number to its left or the machine number to its right. In round to nearest we choose the closest of these machine numbers. If the exact value falls in the dead center of the interval then we choose the machine number that is even (i.e., has a 0 in the least significant place of `f`). Note that with this rounding convention the maximum error is just half a unit in the last significant place of the number.

Question: Explain the last statement "... error is just half a unit in the last significant place ...."

There are other rounding modes, called *directed rounding*. These are round toward $+\infty$, round toward $-\infty$, and round toward 0 (truncation).

Question: What is the difference in the error when using directed rounding as compared with round to nearest?

4 Infinity, NaN, and zero

4.1 Infinity

Infinity gets treated like an ordinary, but very large, number whenever it makes sense to do so. For example, the arithmetic operations below produce the results indicated on the right when x is a positive floating-point number:

$$\begin{aligned} +\infty + x &\to +\infty \\ +\infty \times x &\to +\infty \\ x/+\infty &\to +0 \\ +\infty/x &\to +\infty \end{aligned}$$

Corresponding results apply for the case x negative and $-\infty$.

4.2 NaN

This value is used for the results of operations that are indeterminate. Here are some examples:

$$\begin{aligned} 0/0 &\to \mathtt{NaN} \\ (+\infty) - (+\infty) &\to \mathtt{NaN} \\ x + \mathtt{NaN} &\to \mathtt{NaN} \end{aligned}$$

4.3 Zero

As we have seen, there are two forms of zero, $+0$ and -0. Most arithmetic that would produce a zero result gives the result $+0$. The intent of -0 is to provide a mechanism for recognizing that a number that is zero to machine precision is, in fact, a very tiny negative number.

5 Of things not said

There is much more to the IEEE standard than what has been said here, but we have touched on the main points. The standard also includes binary-to-decimal conversion, square root, and other operations. It allows for graceful underflow, a mechanism that allows for very small numbers to be represented even when they are outside the range as described above. It also specifies the detection of various types of exceptions, such as overflow and underflow. One of the more interesting exceptions is *inexact*, which means that a numerical result is not exact because of rounding.

6 Further reading

The formal document defining the standard is the document *An American National Standard & IEEE Standard for Binary Floating-Point Arithmetic*.[1] You can find help for understanding the standard in a series of articles that appeared in the journal *IEEE Computer*, volume 14, number 3, 1981.

[1] Available from The Institute of Electrical and Electronics Engineers, Inc., 345 E. 47^{th} St., New York, NY 10017.