Digital Hardware
Problems 4

Mike Spivey, Hilary Term 2004

This problem sheet is about floating-point arithmetic. To make the calculations easier, the questions refer to an invented 'half-precision' floating point format that is like those of IEEE–754, but with a sign bit, a 6-bit exponent and a 9-bit significand packed in a 16-bit word. The exponent is represented with a bias of 31, and the leading one bit of the significand is suppressed in a normalized number. The exponent value 0 is reserved for zero and denormalized values, and the exponent value 63 is reserved for exceptional values.

1 Find the representations of the numbers $\frac{1}{2}$, $\frac{7}{8}$, 1, $1\frac{1}{2}$, 2, $6\frac{1}{2}$.

2 What is the closest representable number to $\frac{1}{3}$?

3 Find the smallest and largest positive numbers that can be represented in normalized form.

4 Find the smallest positive number that can be represented as a denormalized value.

5 Find ϵ such that $1 + \epsilon$ is the smallest representable number greater than 1. Why does this give a better measure of the precision of the representation than the answers to the previous questions? Is there any representable number between $1 - \epsilon$ and 1?

6 Use a pocket calculator (or a computer) and Newton's method to compute the reciprocal of 3, taking 0.5 as the initial approximation. Explain why, once correct digits start to appear, they double in number with each successive iteration.

The remaining exercises are devoted to examples that illustrate the steps of the algorithm for floating point addition given in Figure 1. The first few exercises concentrate on the normalization steps, and the following ones on the rounding steps.

7 By considering the addition $1 + 1$, show that it may be necessary to shift s_z to the right in step (5) of the algorithm. How many digits to the right might it be necessary to shift z?

Inputs: two normalized floating-point numbers x and y.

(1) Unpack x and y to obtain significands s_x and s_y and exponents e_x and e_y, restoring the implied leading 1 to the significands. (At this stage, $1 \le |s_x| < 2$ and $1 \le |s_y| < 2$.)

(2) If $e_x < e_y$ then swap s_x and e_x with s_y and e_y.

(3) Shift s_y to the right and increase e_y until $e_x = e_y$.

(4) Compute $s_z = s_x + s_y$ and $e_z = e_x = e_y$, adding or subtracting the magnitudes as required by the signs.

(5) If $|s_z| \ge 2$ then shift s_z to the right and increase e_z; and if $0 < |s_z| < 1$ then shift s_z to the left and decrease e_z; in both cases shift far enough that $1 \le |s_z| < 2$.

(6) Round s_z to the required number of digits.

(7) If the previous step has resulted in $|s_z| \ge 2$ then normalize again, as in step (5).

(8) Pack s_z and e_z as the result, suppressing the leading 1 from the significand.

Figure 1: *Steps in floating-point addition*

8 By considering the addition $x + y$, where $x = 1 + \epsilon$ and $y = -1$, show that it may be necessary to shift a non-zero result s_z to the left in step (5) by up to D digits, where D is the number of significand digits that are stored in the floating-point representation.

9 Consider the addition $x + y$ where $x = 1$ and $y = -\frac{1}{4}$, and explain why, if s_y is shifted right by more than one place in step (3), it is necessary to shift s_z left by at most one place in step (5). (This observation will help with our treatment of rounding later.)

10 Consider the addition $1.111\,111\,111_2 + 1.111\,111\,111_2 \times 2^{-10}$, and explain why step (7) is needed.

The final few exercises concern the implementation of correct rounding, using the 'round to even' rule: that the result should be the nearest representable number to the true sum of the two operands, or if two representable numbers are equally close, to whichever has a zero in the least significant digit. Thus the rule about an even result is used only to break ties.

11 As the previous exercise shows, the true answer to the addition may have as many as $2D + 1$ digits after the radix point. Suppose that we discard all but $D + 1$ of these from s_x and s_y and s_z, keeping only one more digit that is present in the floating-point format. Consider an addition where s_y is shifted by at most one place in step (3) – such as $x + y$, where $x = 1 + \epsilon$ and $y = x/2$ – and show that in such a case a correctly rounded result may be obtained.

12 We are left with the case where s_y is shifted right by two or more places in step (3); as exercise 9 shows, cancellation can result in a shift of up to one place to the left in step (5). Consider the additions $x + y_1$ and $x + y_2$ where $x = 1$, $y_1 = -\frac{1}{4}(1 - 2\epsilon)$ and $y_2 = -\frac{1}{4}(1 - 3\epsilon)$, and show that although these additions have different correctly-rounded results, the intermediate results are the same when truncated to $D + 1$ digits after the radix point. This illustrates that $D + 1$ digits are not enough; show, however, that $D + 2$ digits are enough in this case.

13 Consider the calculation $x + y$ where $x = 1$ and $y = \frac{1}{2}\epsilon(1 + \epsilon)$, show that the correct answer is not obtained if the intermediate result is truncated to fewer than $2D + 1$ places after the radix point.

Instead of keeping all the digits of the intermediate results, we could try keeping just $D + 2$ of them, but also keeping track of whether any 1 bits have been lost be truncation: thus aligning the inputs of the addition might give $y = 1.000\,000\,001_2 \times 2^{-10} = 0.000\,000\,000\,10_{2+} \times 2^0$, where the '+' denotes the loss of the least significant 1 bit. Show that this representation allows the calculation to be completed with the correct result.

This last exercise completes the picture of what is needed for correct rounding of addition: in addition to the D bits after the radix point in the result, we must keep one bit for rounding, another bit to guard against the possibility that normalizing will require a shift of one place to the left, and a final 'sticky' bit that indicates whether any 1 bits have been lost by truncation.