University of Stirling
Department of Computing Science and Mathematics

CSCU9V4 Systems I - Tutorial 3 Solutions

1. Explain why a simple (apparently exact) floating-point number like 1.2f (say) is represented on a computer by an approximation. What are the consequences of this when two (or three, or a hundred) floating-point numbers are added?

*Soln: Unless a number can be represented exactly as (binary expansion of appropriate length * 2^exponent of appropriate length) the floating point number will be an approximation.*
Effect is that there is inaccuracy in the number. Generally such inaccuracy doesn't build up, but it can under adverse circumstances. Care needs to be taken so that one knows the degree of accuracy of the final result.

2. What is meant by underflow, in respect of floating-point arithmetic? Write down an expression that might result in underflow.

Soln: Underflow is representing a number too near to zero (so that it gets represented as 0). E.g. (small number)/(large number)

3. What is meant by overflow, in respect of floating-point arithmetic? Write down an expression that might result in (i) positive overflow and (ii) negative overflow.

Soln: Overflow is attempting to represent a number bigger than floating-point can represent (may be too +ve or too –ve). Can be generated by multiplying two large (magnitude) numbers, or dividing a very large number by a very small number.

4. 23 bits are used for the fractional part and 8 bits are used for the exponent in the IEEE 754 Standard for single-precision floating-point numbers. Using the standard "calculate" the largest number that can be represented. What is the smallest normalised number? What range of values does the denormalised form of the standard give you? Show how you calculated the values.

Soln:
*1.111(23 times)1 * 2^(254-127) is largest (normalised)*
*1.0(22 times)1 * 2^(1-127) is smallest normalised*
*0.0(22 times)1 * 2^(1-127) is smallest denormalised*
(actual values:
*3.4 * 10^38, 1.8 * 10^-38, 5.9 * 10^-39 respectively).*

5. Consider (i) multiplying two floating-point numbers (ii) dividing one floating-point number by another. For what sorts of calculations are these operations likely to produce (relatively) large inaccuracies? Can you describe conditions where these inaccuracies might mount up? How might one avoid this?

Soln: Where the intermediate result is likely not to be represented accurately, or may cause overflow, or be represented by a denormalised number. For example, consider
*2 * 10^38 * 100.123/1234567.1*
or
*2^10^-38 * 1445.7 * 0.004*

6. You want to calculate x/a – y/a where you know that x and y may be very close to each other, and a is a large number. How would you do this in such a way as to minimise the rounding error?

Soln: calculate as: (x-y)/a