CS33 Homework Assignment 3 Solutions

Fall 2020

1. We argued in class that one of the benefits of the two's-complement approach for representing numbers is that the computer can do arithmetic without concern for whether unsigned or signed numbers are being used, and the result will be correct in either case. Thus, for example, the Intel x86 instruction set does not have separate signed and unsigned versions of its *add* and *subtract* instructions, but *add* and *subtract* both produce correct results regardless of whether one is interpreting their operands (and results) as signed or unsigned.

   The Intel x86 *multiply* instruction, for 64-bit operands, can produce a 128-bit result, thus eliminating any concern about overflow. However, unlike the *add* and *subtract* instructions, there are both a signed multiply and an unsigned multiply. Explain why. (Hint: consider multiplying a 64-bit value that consists of all ones by a 64-bit encoding of positive 2.)

   Answer: When performing arithmetic on two's-complement-encoded numbers of word-size *w*, we can ignore the distinction between signed and unsigned as long as we use only the low-order *w* bits of the result, thus ignoring overflows as well as carries into and borrows from higher-order bits. But, consider for example, the result of multiplying a 64-bit value, all of whose bits are one, times a 64-bit encoding of positive 2. If we treat the values as signed quantities, then the result is negative two (since the multiplier is negative one). But if we treat the values as unsigned quantities, the result should be 2^{65}-2 (since the multiplier is 2^{64}-1). As a 128-bit value, negative two is encoded as all ones except for a zero as the least-significant bit. However, 2^{65}-2 has the same encoding for its low-order 64 bits, but the high-order 64 bits are all zeroes except for a one as the least-significant bit. Thus, for a 128-bit product of two 64-bit numbers, separate signed and unsigned multiply instructions are required because the high-order 64 bits will differ depending on whether signed or unsigned arithmetic is being used.

   Note that similar concerns apply to division, for which there might be a 128-bit dividend and a 64-bit divisor producing a 64-bit quotient and a 64-bit remainder.

2. IEEE floating point represents values closest to zero using its denormalized form. All other values, with the exception of a few special cases, are represented using its normalized form.

   a. Suppose there were no denormalized form, and thus an exponent value of zero would be treated as if the exponent were -1023 and there would be an implied leading one in the significand. What would be the smallest representable positive value, and how much larger would the next smallest be?

      Answer: 2^{-1023} and 2^{-1075}. The smallest significand would be 1.0 (coded as all zeroes). With an exponent of -1023, the value would be 2^{-1023}. The next smallest value would have a significand of 1.00...01, with 51 zeroes before the one, and thus would represent a number that is 2^{-1075} larger.

b. What is the minimum number of significant bits in the significand for normalized values? What is the minimum number of significant bits in the significand for denormalized values?

Answer: 53 and 1. (53 is also the maximum number of significant bits.)