Homework 6: Data Encoding

CS 200 • 10 + 5EC Points Total
Due Wednesday, March 22, 2017

Assignment

- Read Chapter 2 of the textbook. Answer the following questions.
  1. What causes divide underflow? (1 pt)

     Divide underflow is happens when the divisor is much smaller than the dividend. The computer may not be able to reconcile the two numbers of greatly different magnitudes, and thus the smaller gets represented by zero. The result of the division then becomes the equivalent of a division by zero error. Compare this to simple underflow, which is a condition that can occur when the result of a floating point operation would be smaller in magnitude (closer to zero, either positive or negative) than the smallest representable quantity.

  2. Let $a = 1.0 \times 2^{10}$, $b = -1.0 \times 2^{10}$ and $c = 1.0 \times 2^{0}$. Using a 14-bit normalized floating-point format (5 bits for the exponent with bias of 6, a normalized mantissa of 8 bits with a leading implied '1' in front, and a single sign bit), perform the following calculations, paying close attention to the order of operations. Remember that the computer normalizes the answer after each operation, **so you must do likewise**. What can you say about the algebraic properties of floating-point arithmetic in our finite model? Do you think this algebraic anomaly holds under multiplication as well as addition? (3 pts)

     $b + (a + c) =$
     $(b + a) + c =$

     b + (a + c) **$= b + [1.0 \times 2^{10} + 1.0 \times 2^{0}]$**
     $= b + [1.0 \times 2^{10} + .0000000001 \times 2^{10}\ )$
     $= b + 1.0000000001 \times 2^{10}$
     $= b + 1.00000000 \times 2^{10}$ but we can only have 8 bits in the mantissa (00110100000000) in the 14-bit format above
     $= b + 1.0 \times 2^{10}$
     $= (-1.0 \times 2^{10}) + (1.0 \times 2^{10})$
     $= 0$

     (b + a) + c **$= [(-1.0 \times 2^{10}) + (1.0 \times 2^{10})] + c$**
     $= 0 + c$
     $= c$
     $= 1.0 \times 2^{0}$
     $= 1$

     When one number is significantly larger than the other, round off error occurs due to the limited number of bits in the mantissa. Multiplication does

not require the values to be expressed with the same powers of 2, and does not suffer from this problem but bits of precision can still get lost.

3. When would you choose a CRC code over a Hamming code? A Hamming code over a CRC? I'm looking for a general rule, not specific examples. (2 pts)

   CRCs are useful for checking data sent over telecommunication lines. If a CRC error occurs, a retransmission is requested. You would choose a CRC when you can ask for retransmission and do not want to sustain the (space and time) overhead of Hamming codes. Hamming codes are good at forward error correction: they can correct errors when retransmission is not possible, such as when data is stored on a disk. CRC is better than Hamming in terms of speed, but Hamming is better than CRC in terms of complexity (Hamming codes do not require complex circuits).

4. Find the quotients and remainders for the following division problems modulo 2. (2 pts)

   a) $1100111_2 \div 1011_2$
   b) $1101110_2 \div 1101_2$
   c) $1110101101_2 \div 10110_2$
   d) $1110110101_2 \div 110110_2$

   a. 1110 Remainder 101

```
            1110
         ________
  1011  | 1100111
          1011
          ----
           1111
           1011
           ----
            1001
            1011
            ----
             101
```

   b. 1000 Remainder 110

```
            1000
         ________
  1101  | 1101110
          1101
          ----
            0110
```

c. 110000 Remainder 1101

```
           110000
10110 |1110101101
       10110
        10110
        10110
            01101
```

d. 10100 Remainder 1101

```
             10100
110110 |1110110101
        110110
          110101
          110110
              1101
```

5. Using the CRC polynomial 1011, compute the CRC code word for the information word 00010100. Check the division performed at the receiver. (2 pts)

Append three 0s to the end of the information word and divide:

```
1011 |00010100000
         1011
            1000
            1011
              110    ← remainder
```

The information word (with appended zeros) + remainder = codeword
so we have: 00010110000 + 101 = 00010110101

To check the division:

```
1011 |00010100110
         1011
            1011
            1101
               00    ← remainder
```

- For 5 points extra credit, write a program in C/C++ that encodes byte values as Hamming Code. Allows the user to enter a number between 0 and 255. Print out the original 8-bit number in binary, then print out the 12-bit Hamming Code representation. Attach your source code and the output of your program with three different numbers tested: 142, 204, and 255.

    Extra Credit code solution not given