Homework Set 1 : Data Representation

Q1. (a)What base 10 value does 1011 represent?
(b) Give the answer for the following representation systems. Assume each system uses 4 bits.

- UB (unsigned binary)
- 1C (one's complement)
- 2C (two's complement)
- SM (signed magnitude)
- excess 4 notation

Q2. Translate -4_{10} to the following representations. Assume 4 bits. Answers using fewer or more than 4 bits will not be considered correct. If it's not possible to represent -4 in a given representation system, write NR (not representable).

- UB (unsigned binary)
- 1C (one's complement)
- 2C (two's complement)
- SM (signed magnitude)
- excess 4 notation

Q3. Convert 0xcafe to octal. Convert it to binary.

Q4. Add **1000 + 0001**. State any reasonable assumptions.

Q5. Write -6.125 in IEEE 754 single precision. Label all bits properly (sign bit, exponent, fraction).

Q6. (a) Convert 1.110×2^{-130} to IEEE 754 single precision.
(b) How many significant bits in the previous number in IEEE 754 single precision?

Q7. How can you tell if a 32 bit number is NaN? That is, describe what bits you look at to determine that it's NaN.

Q8. (a) How many significant bits in the previous number in IEEE 754 single precision?
(b) What's the excess in the exponent for IEEE 754 for normalized numbers?
(c) What's the smallest, positive normalized number (write in scientific notation) that's representable in IEEE 754 single precision? Use binary for the significand/mantissa and base 10 for the exponent.

Q9. Someone claims that if you type "0100" into a file using a standard text editor, you have written 4 bits to the file, and thus created a binary file. Explain what's correct and incorrect about this statement.

Q10. You are reading in a file consisting of ***ints (written in binary, not in ASCII)*** only. You wish to read these numbers into an array. What problems might you run into? (Think about data representation).

Q11. Given a floating-point format with a k-bit exponent and an n-bit fraction, write formulas for the exponent E, significand M, the fraction f, and the value V for the quantities that follow. In addition, describe the bit representation.

a. The number 7.0

b. the largest odd integer that can be represented exactly

c. The reciprocal of the smallest positive normalized value

Q12. Around 250 B.C., the Greek mathematician Archimedes proved that 223/71 < pi < 22/7. Had he had access to computer and the standard library <math.h>, he would have been able to determine that the single-precision floating-point approximation of pi has the hexadecimal representation 0x40490FDB. Of course, all of these are just approximations, since pi is not rational.

a. What is the fractional binary number denoted by this floating-point value?

b. What is the fractional binary representation of 22/7?

c. At what bit position (relative to the binary point) do these two approximations to pi diverge?

Q13. You have been assigned the task of writing a C function to compute a floating-point representation of 2^x. You decide that the best way to do this is to directly construct the IEEE single precision representation of the result. When x is too small, your routine will return 0.0. When x is too large, it will return positive infinity. Fill in the blank portions of the code that follows to compute the correct result. Assume the function u2f returns a floating-point value having an identical bit representation of its unsigned argument.

```
float fpwr2(int x)
{
    /* result exponent and fraction */

      unsigned exp, frac;
      unsigned u;

      if (x < _____________________)
      {

         /* too small. return 0.0 */

         exp = _________________;
         frac = ____________________;
       }

      else if (x < _____________________) {

         /* denormalized result */
          exp = _________________;
          frac = ____________________;
       }

       else if (x < _____________________) {

           /* normalized result */
            exp = _________________;
            frac = ____________________;
        }

     else
              /* too big. Return +oo */
         exp = _________________;
         frac = ____________________;
     }

      u = exp << 23 | frac;  /* pack exp and frac into 32 bits */
      return u2f(u);         /* return as float */
}
```