Questions

1. Give the 16 bit signed (twos complement) representation of the following decimal numbers, and convert to hexadecimal:

   (a) 3012

   (b) - 435

2. For each of the following bit strings:

   - Which decimal number is represented, assuming the bit string is a signed integer ?
   - Which decimal number is represented, assuming the bit string is an unsigned integer?
   - Convert the bit string to hexadecimal.

   (a) 0100100101

   (b) 1111101001

3. Convert the following two decimal numbers to signed binary (twos complement). Be careful to choose enough bits.

   (a) -200

   (b) 259

4. Convert `0xff3e` from hexadecimal to binary, treat this binary number as a signed integer, and convert it to decimal.

5. (a) Convert the decimal number -499 to signed binary (twos complement).

   (b) Convert `0xfc` to binary, treat it as a signed integer, and convert to decimal.

6. (more challenging) Writing a positive integer in binary means expressing it as a sum of powers of 2:

$$(b_{n-1}\ b_{n-2}\ \ldots\ b_2\ b_1\ b_0)_2 \ \equiv\ \sum_{i=0}^{n-1} b_i\ 2^i$$

   This representation makes sense for unsigned numbers. But what about signed numbers? How can we represent a signed number as a sum of powers of 2? Which powers of two are represented and what are the b_i ?

7. Convert -5.75 to binary.

8. Represent the following two decimal numbers as single precision float. Show:

   - a scientific notation representation
   - the significand, exponent, and sign bits (32 bits total)
   - a hexadecimal representation of these 32 bits

   (a) 15.0625

   (b) -35.6875

   (c) 86.5625

9. What are the largest and smallest positive, finite, normalized numbers that can be represented as IEEE single precision float? It is sufficient for you to write the answer using scientific notation.

10. Consider representing the set of consecutive numbers

$$\{\ 1,\ 2,\ 3,\ 4,\ \ldots,\ n\ \}$$

    using:

    (a) 32 bit unsigned

    (b) 32 bit signed

    (c) IEEE single precision floating point (a bit tricky)

    For each of these representations,

    - what is the largest n such that *every* number in the above set can be represented, that is, no gaps between numbers?
    - write this largest n using hexadecimal (rather than binary)

11. (a) Approximate the decimal number 43.00008 as a normalized binary number in scientific notation, with eight bits of significand.

    (b) Convert the decimal number 0.125 into IEEE single precision floating point. Give your answer in hexadecimal.

12. (challenging, you need to consider various cases)

    (a) Consider two normalized floating point numbers `f1` and `f2` that are represented as IEEE single precision floats. This defines two 32-bit strings.

    Suppose `f1` is less than `f2`, that is, `f1` $<$ `f2`.

    If the same two 32-bit strings were treated as *unsigned* integers (call them `i1` and `i2` respectively), can we conclude that `i1` $<$ `i2` ?

(b) Similar to (a), if the two 32 bit strings were treated as *signed* integers `j1` and `j2`, could we conclude that `j1` $<$ `j2` ?

13. The IEEE single precision floating point standard allows us to represent less than 2^{32} different numbers. Of these numbers:

    (a) How many are strictly between 2^{-5} and 2^{-4} ?

    (b) How many are strictly between 2^{13} and 2^{14} ?

    (c) How many are strictly between 2^{47} and 2^{48} ?

14. (a) Write the following binary number as an IEEE single precision float: $0.10101010\ldots$, that is, bits 10 repeating infinitely many times to the right of the binary point.

    (b) Convert $0.10101010\ldots$ from binary to decimal. (cute)

Solutions

1. (a) The signed (twos complement) representation of 3012 is 0000 1011 1100 0100.
   In hexadecimal, this is 0x0bc4.

   (b) The signed representation of 435 is 0000000110110011.
   Thus, the signed representation of -435 is 1111111001001101
   In hexadecimal, the latter is 0xfe4d.

2. (a) if signed then 293
   if unsigned then (also) 293
   Convert to hex: 0x125

   (b) if signed then -23
   if unsigned then 1001 ("one thousand and one")
   Convert to hex: 0x3e9

3. (a) To convert -200 to binary, we convert 200 to binary, invert the bits, and add 1.
   200 in binary is $\ldots 0011001000$. For this to be a signed number, the most significant bit must be 0 since 200 is positive. Let's use ten bits: 0011001000.
   Inverting the bits yields 1100110111. Adding 1 yields the answer:

   $$1100111000.$$

   (b) To convert 259 is easier. The answer is 0100000011.

4. Converting the hexadecimal number `0xff3e` to signed binary, gives 1111 1111 0011 1110 . This is a negative number since the most significant bit is 1. To determine what this number is, we take its negative by inverting the bits and adding 1, yielding 0000000011000010. This is $2 + 2^6 + 2^7$ which is $2 + 64 + 128 = 194$.

   Thus, the answer is -194.

5. (a) $(499)_{10} = (0001\,1111\,0011)_2$. Inverting the bits, we obtain 1110 0000 1100. Then add 1 to obtain the answer 1110 0000 1101. This is in 2's complement with 12 bits. You needed at least one leading bit, e.g. 10 0000 1101 is accepted.

   (b) 0xFC in binary is 1111 1100. This is a negative number, so we invert the bits and add 1 to obtain $(0011)_2 + 1 = (0100)_2 = (4)_{10}$ so the answer is -4.

6. For a signed number, we have

   $$(b_{n-1}\ b_{n-2}\ \ldots\ b_2\ b_1\ b_0)_2 \ \equiv\ -\,b_{n-1}\ 2^{n-1} + \sum_{i=0}^{n-2} b_i\ 2^i.$$

   For example, take $n = 8$.

   $$(10000000)_2 = -128 = -2^7,$$
   $$(10000001)_2 = -127 = -2^7 + 2^0,$$
   $$(10000010)_2 = -126 = -2^7 + 2^1, \ldots$$

   That is, the 1 in the highest order bit defines the smallest (most negative) number represented and the lower order other bits just add positive values bringing the number closer to 0.

7. Note that - 5.75 $\neq$ -5 + .75. Rather 5.75 = -(5.75).

   In the former case, you might have converted -5 to binary by saying $(5)_{10} = (101)_2 = (0101)_2$ and then flipping the bits and adding 1 to get $(1011)_2$. You then might have converted .75 to binary as $(.11)_2$ and concluded the answer is (1011.11) which would be *incorrect*.

   The correct way to go is to write 5.75 in binary as 0101.11, where we have added a high order 0 bit so that the number is positive when interpreted as a signed number. We then compute its negative by inverting the bits 1010.00 and adding .01 to get the answer 1010.01. Verify that adding this to 5.75 gives 0.

8. (a) Converting 15.0625 to binary gives 1111.0001 or $1.1110001 * 2^3$ in scientific notation. Since the exponent is 3, the exponent code e is determined by $e - 127 = 3$, and so the exponent code is the (unsigned) binary representation of 130 which is 10000010.

   The sign bit is 0 since the number is positive.

   Writing as a string (sign bit, exponent, significand) gives

   0 10000010 11100010000000000000000

   which in hexadecimal is: $0x41710000$

   (b) Converting 35.6875 to binary is 100011.1011 which is $1.000111011 * 2^5$

   The sign bit is 1. The exponent code is the unsigned representation of 132 (since $e-127 = 5$). Thus the exponent code is 10000100. The significand is 00011101100000000000000.

   Writing as a string (sign bit, exponent, significand) gives

   1 10000100 00011101100000000000000

   which in hexadecimal is: 0xc20ec000.

   (c) $86.5625 = 1.0101\ 1010\ 0100 \times 2^6$

   $e = 127 + 6 = 133$.

   So we have 0 10000101 01011010010000000000000.

   Re-group the bits to obtain:

   $(0100\ 0010\ 1010\ 1101\ 0010\ 0000\ 0000\ 0000)_2$ = 0x42AD2000

9. The largest exponent code is 11111110 i.e. 254 (since 11111111 is reserved). Thus, the largest exponent that can be represented is $254 - 127 = 127$. Thus, the largest (finite) float is $1.11111111111111111111111 * 2^{127}$.

   The smallest exponent code for normalized numbers is 00000001 (since 00000000 is reserved). This exponent code has value 1 since it is treated as an unsigned number. Thus, the smallest exponent that can be represented is $1 - 127 = -126$.

   It follows that smallest (normalized positive) float is $1.00000000000000000000000 * 2^{-126}$ which is just 2^{-126}.

10. (a) The largest unsigned is $2^{32} - 1$ which in hex is 0xffffffff.

    (b) The largest signed is $2^{31} - 1$ which in hex is 0x7fffffff.

(c) First, note that 1 to $2^{24} - 1$ can all be represented exactly. Why? Because when you write out one of the those numbers in binary (as unsigned), you only need 24 bits. When you try to write any of these numbers in IEEE format, the most significant 1 bit isn't used because we are representing the numbers as normalized. So we have room for up to 23 bits, in the significand and we're ok.

In particular, the number 111111111111111111111111 which has 24 1's can be represented exactly. This number is $2^{24} - 1$.

Next, what about $2^{24} = (100000000000000000000000)_2$? Can it be written exactly? Yes, because it is just $1.0 * 2^{24}$ and I hope by now it is obvious that this can be written exactly in IEEE format, namely the exponent value is 24 and the significand is all 0's. So, all the numbers from 1 to 2^{24} can be represented exactly.

What about

$$2^{24} + 1 = (100000000000000000000001)_2 = (1.00000000000000000000001)_2 * 2^{24} \ ?$$

Can it be represented exactly? No it can't. Why not? Because there is only room for 23 bits in the significand and those bits would be the 23 0's to the right of the binary point, and so we wouldn't be able to get in the 2^{-24} bit.

Thus, the largest n is such that 1, ..., n can be represented exactly in single precision IEEE format is 2^{24}.

11. (a) To approximate 43.00008 as a normalized binary number in scientific notation, we first convert the part to the left of the decimal point to binary. Converting 43 to binary is 101011. This is already close to the number of bits of significand that we need.

At this time, you should suspect that the 0.00008 will not play a role in your answer. If you wish to verify this, you can try:

$$\begin{aligned} & 101011.(00008)_{10} \\ = \ & 1010110.(00016)_{10} * 2^{-1} \\ = \ & 10101100.(00032)_{10} * 2^{-2} \\ = \ & 101011000.(00064)_{10} * 2^{-3} \end{aligned}$$

but at this point we have our eight bits of significand. So,

$$43.00008 \approx 1.01011000 * 2^8 * 2^{-3} \ = \ 1.01011000 * 2^5$$

This is our answer.

(b) First write the decimal number 0.125 in normalized scientific notation: $1.0 * 2^{-3}$. The significant is thus all 0's. To code the exponent -3, we need $x - 127 = -3$. Thus we need the 8-bit binary of $x = 124$, which is 01111100.

The sign bit is 0 (positive number), and the IEEE encoding is the

$$(0\ 01111100\ 00000000000000000000000).$$

To convert to hexadecimal, we group into 4-tuples:

$$(0011, 1110, 0000, 0000, 0000, 0000, 0000, 0000)$$

and we get `0x3e000000`.

12. (a) No. As a counterexample, suppose that the most significant bit of `f1` is 1 and the most significant bit of `f2` is 0. Then `f1` $<$ `f2` since the most significant bit is the sign bit: 1 is negative and 0 is positive. Treated as *unsigned* numbers (`i1` and `i2`), however, the MSB's alone would imply that `i1` $>$ `i2`.

Another way to see this is to consider the ordered list of all 2^{32} possible words.

$0000\ldots0000$
$0000\ldots0001$
$0000\ldots0010$
:
$0111\ldots1111$
$1000\ldots0000$
$1000\ldots0001$
:
$1111\ldots1111$

In the first half of these (MSB = 0), both the float and the int representing are increasing as we proceed down the list. Within the second half of these (MSB = 1), the float values are decreasing as we proceed down the list because the sign is negative and either the exponent or significand is decreasing. However the int values are increasing as we proceed down the list. Thus, within the second half, the ordering of the float values is opposite to the ordering of the int values.

(b) No. Assume `f1` $<$ `f2` and consider whether this implies `j1` $<$ `j2`. There are three cases. Only in the second case is `j1` $<$ `j2`.

Note that once one of the cases fails, the proof is done, you have found a counterexample to the statement "`f1` $<$ `f2` implies `j1` $<$ `j2`".

- Case 1: the MSB's of both 32-bit numbers are 0 and so the float values are both positive. There can be two ways in which `f1` $<$ `f2`. The first is that the exponent of `f1` is less than the exponent of `f2`. The second is if the exponent of `f1` is equal to the exponent of `f2`, but the significand of `f1` is less than that of `f2`. In either case, `f1` $<$ `f2` implies that `j1` $<$ `j2`. (If you cannot see why, please see me or send me email.)
- Case 2: the MSB's of both 32-bit numbers are 1. In this case, the float values are both negative (and the integers `j1` and `j2` are both negative). Since we are assuming `f1` $<$ `f2` and both are negative, it follows that the absolute value of `f1` must be greater than the absolute value of `f2`. From this, it follows that either the exponent of `f1` is *greater than* the exponent of `f2`, or the exponent of `f1` is equal to the exponent of `f2` but the significand of `f1` is *greater than* that of `f2`. In either case, it follows that `j1` $>$ `j2`. To see this, recall the argument from case 1, but notice that both ints are on the negative part of the "circle".

- Case 3: The MSB of one of `f1` or `f2` is 1 and the MSB of the other is 0. If `f1` < `f2`, then there is only one for this to happen, namely that the MSB of `f1` is 1 (`f1` is negative) and the MSB of `f2` is 0 (`f2` is positive). In this case, if the numbers are interpreted as signed integers, then `j1` would be negative (since MSB is 1) and `j2` would be positive (since MSB is 0), and hence `j1` < `j2`.

13. For all three questions, the answer is $2^{23} - 1$ different floating point numbers.
For example, between 2^{-5} and 2^{-4} there are all the floating point numbers with $127 - 5 = 122 = (0111\ 1010)_2$ as the exponent and 1 as the sign bit. Since there are 2^{23} bits in the significand, and any setting of these produces a valid FPN, there are 2^{23} different FPNs. We exclude the number with significand all 0 s, since this is the value 1.0×2^{-5}. Thus the solution $2^{23} - 1$.

14. (a)

$$0.101010\cdots = 1.0101010101\cdots \times 2^{-1}$$

So, the sign bit is 0, the exponent code is 126 or $(01111110)_{two}$, and the 23 bit significand is
010 1010 1010 1010 1010 1010. So the 32 bits in the float and their corresponding hex representation are

0011 1111 0010 1010 1010 1010 1010 1010 or 0x3f2aaaaa

(b) There are different ways to crank out the answer. Here is one:

$$\begin{aligned}
.101010\cdots &= \frac{2}{2^2} + \frac{2}{2^4} + \frac{2}{2^6} + \frac{2}{2^8} + \dots \\
&= \frac{2}{2^2}(1 + 2^{-2} + 2^{-4} + 2^{-6} + 2^{-8} + \cdots) \\
&= \frac{1}{2}(1 + a + a^2 + a^3 + a^4 + \cdots) \text{where } \ a = 2^{-2} \\
&= \frac{1}{2}\frac{1}{(1-a)}, \text{ i.e.geometric series} \\
&= \frac{1}{2}(\frac{4}{3}) \\
&= \frac{2}{3}
\end{aligned}$$