Week 2: Representation of Information (Solutions)

1 Positive Decimal Numbers

1. Let X be a number in decimal for which we look for its unknown binary pattern denoted by $0.b_w b_x b_y b_z$. The expression of X in binary is therefore $X = b_w \times 2^{-1} + b_x \times 2^{-2} + b_y \times 2^{-3} + b_z \times 2^{-4}$. The steps are:

   (a) $2 \times X = b_w \times 2^0 + b_x \times 2^{-1} + b_y \times 2^{-2} + b_z \times 2^{-3}$.

   (b) The integer part is $b_w \times 2^0$ which gives us b_w as the first weight obtained.

   (c) Keep only the fractional part, that is $b_x \times 2^{-1} + b_y \times 2^{-2} + b_z \times 2^{-3}$. If it is zero, the conversion is finished. Otherwise, repeat (a) with this quantity.

2. If X is 0.375 the algorithm runs as follows:

   (a) $2 \times X$ is 0.750.

   (b) b_w is 0.

   (c) The fractional part 0.750 is not zero: we continue.

   (a) 2×0.750 is 1.5.

   (b) b_x is 1.

   (c) The fractional part 0.5 is not zero: we continue.

   (a) 2×0.5 is 1.

   (b) b_y is 1.

   (c) The fractional part is zero and we are done; therefore b_z is 0.

   Applying this method to 0.1_{10} gives you the following binary pattern $0.0\overline{0011}$, which is an infinitely long pattern because the overline means that this pattern repeats indefinitely. This means that there is no exact representation of this number in binary.

   Solutions:

   (a) 0.375_{10}: 0.011

   (b) 0.1_{10}: $0.0\overline{0011}$

   (c) 0.625_{10}: 0.101

   (d) 0.125_{10}: 0.001

2 Overflow and Capacity

1. Conversions: 8-bit binary patterns.

   (a) 00000110 is positive, with a value of 4 + 2 = 6.

   11111001 is negative because its most significant bit is 1. To know its absolute value, we calculate its opposite by taking its 2's complement, i.e., the 1's complement to which we add 1.

	11111001	
	00000110	1's complement
+	00000001	+1
	00000111	which is 7, so the initial pattern is -7.

10000110 is negative because its most significant bit is 1. To know its absolute value, we calculate its opposite by taking its 2's complement, i.e., the 1's complement to which we add 1.

	10000110	
	01111001	1's complement
+	00000001	+1
	01111010	which is 2 + 8 + 16 + 32 + 64 = 122, so the initial pattern is -122.

(b) 0 = 00000000.

-12 first calculate 12 in binary then take its 2's complement

	00001100	12
	11110011	1's complement
+	00000001	+1
	11110100	2's complement = represents -12.

-1 first calculate 1 in binary and then take its complement.

	00000001	1
	11111110	1's complement
+	00000001	+1
	11111111	2's complement = represents -1.

127_{10} = 01111111 = maximum positive numbers for 8 bits.

-128_{10} = 10000000 = minimum of negative numbers for 8 bits.

(c) Conclusion: the domain is not symmetric. One must watch out for the singular case where one asks for the opposite of the minimum negative number, as this would give an incorrect result.

2. The value 64 is representable with this bit pattern: 01000000.

	01000000	64
+	01000000	+64
	10000000	gives -128.

This is an overflow, the addition of 2 positive numbers gives a negative number, which is incorrect.

3 Overflow in Signed Integer Addition and Subtraction

Recall that the MSB of a signed integer in two's complement representation is the sign, i.e., if MSB=0, then the number is positive and if MSB=1, then the number is negative. Consider the second line in Table 1, in this case we are adding two positive numbers but the result is a negative number, therefore an overflow must have happened. In the second to last line of Table 1, we have the opposite case, we are adding new negative numbers and the result is a positive number, which is also not possible without an overflow.

MSB of operand a	MSB of operand b	MSB of the sum $a+b$	Overflow
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Table 1: Addition of signed integers.

MSB of the minuend	MSB of the subtrahend	MSB of the subtraction	Overflow
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Table 2: Subtraction of signed integers.

4 Hexadecimal Representation

In binary $fa_{16} = 11111010_2$, $ca_{16} = 11001010_2$, $de_{16} = 11011110_2$. In decimal $fa_{16} = 250_{10}$, $ca_{16} = 202_{10}$, $de_{16} = 222_{10}$. We see that the weight of red channel has the stronger weight but the other components are also high. It suggests a very light color, with a red overweight: a light pink. More precisely, it's like this.

5 Representation of Floating-Point Numbers

1. With 2 bits of exponent, there will be $2^2 = 4$ successive intervals in which the numbers represented will have the same distance between them. There are $2^3 = 8$ numbers per interval. We use only the following normalized formula: $2^{exponent} \cdot 1.\text{mantissa}$.

2. The min is given by $2^0 \cdot 1.0$, i.e. 1.
   For the max: the exponent = 11, i.e., 3 (in decimal) and the mantissa is 111, i.e., 0.875 (in decimal). The max is given by : $2^3 \cdot 1.875 = 8 \cdot 1.875 = 15$.
   The representative values are

mant. / exp.	000	001	010	011	100	101	110	111
00	1	1.125	1.25	1.375	1.5	1.625	1.75	1.875
01	2	2.25	2.5	2.75	3	3.25	3.5	3.75
10	4	4.5	5	5.5	6	6.5	7	7.5
11	8	9	10	11	12	13	14	15

   The absolute error is not constant: it is the difference between two successive numbers; it doubles when one goes from one interval associated with a power of 2 to the next.

3. The maximum relative error is bounded by LSB of the mantissa, i.e., it is bounded by $2^{-3} = 0.125$, i.e., 12.5%. This bound for the relative error is the same for all 4 intervals.

6 Decimal to Floating-Point Representation

1. 0001111 can be converted to $2^0 \cdot 1.1111$ (the first three bits, 000, are the exponent, and the remaining four, 1111, the mantissa). Thus, $2^0 \cdot 1 + 2^{-1} \cdot 1 + 2^{-2} \cdot 1 + 2^{-3} \cdot 1 + 2^{-4} \cdot 1 = 1.9375_{10}$. Similarly, 1101001 can be written as $2^6 \cdot 1.1001$, which is equal to 100_{10}.

2. To convert a decimal number to a floating point representation, we must first convert the number to binary, then normalize it, and finally keep only as many bits as the representation allows (in this case, 3 bits for the exponent, and 4 for the mantissa).

   We can convert 1.8_{10} to binary, where we will obtain $1.\overline{1100}_2$. We can normalize this as $2^0 * 1.\overline{1100}$. Ignoring the extraneous bits, we can then write it in our simplified 7-bit representation, 0001100. To find the absolute error, we must first compute $2^0 \cdot 1.1100 = 1.75$. Thus, the absolute error is given by $1.8 - 1.75 = 0.05$, and the relative one by $0.05/1.8 \approx 0.028 \leq 0.0625$.

   Similarly, 2.625_{10} can be converted into 10.101_2. We can normalize this as $2^1 \cdot 1.0101$. There are no extraneous bits, which means this number will be represented exactly with the simplified 7-bit representation, 0010101. Both errors are 0.

   Finally, we follow the same process a third time for 114_{10}. We obtain its binary representation, 1110010_2, which can be normalized as $2^6 \cdot 1.110010$. Excluding the extraneous bits, we obtain $2^6 \cdot 1.1100$ (which is equal to 112). The absolute error is then given by $114 - 112 = 2$, and the relative one by $2/114 \approx 0.018 \leq 0.0625$.

For reference, all the values that can be represented exactly with this 7-bit simplified floating point representation:

mant. / exp.	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
000	1	1.0625	1.125	1.1875	1.25	1.3125	1.375	1.4375	1.5	1.5625	1.625	1.6875	1.75	1.8125	1.875	1.9375
001	2	2.125	2.25	2.375	2.5	2.625	2.75	2.875	3	3.125	3.25	3.375	3.5	3.625	3.75	3.875
010	4	4.25	4.5	4.75	5	5.25	5.5	5.75	6	6.25	6.5	6.75	7	7.25	7.5	7.75
011	8	8.5	9	9.5	10	10.5	11	11.5	12	12.5	13	13.5	14	14.5	15	15.5
100	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
101	32	34	36	38	40	42	44	46	48	50	52	54	56	58	60	62
110	64	68	72	76	80	84	88	92	96	100	104	108	112	116	120	124
111	128	136	144	152	160	168	176	184	192	200	208	216	224	232	240	248