Conversion of Binary, Octal and Hexadecimal Numbers

From Binary to Octal

Starting at the binary point and working left, separate the bits into groups of **three** and replace each group with the corresponding **octal** digit.

 $10001011_2 = 010 \ 001 \ 011 = 213_8$

From Binary to Hexadecimal

Starting at the binary point and working left, separate the bits into groups of **four** and replace each group with the corresponding **hexadecimal** digit.

 $10001011_2 = 1000 \ 1011 = 8B_{16}$

From Octal to Binary

Replace each octal digit with the corresponding 3-bit binary string.

 $213_8 = 010 \ 001 \ 011 = 10001011_2$

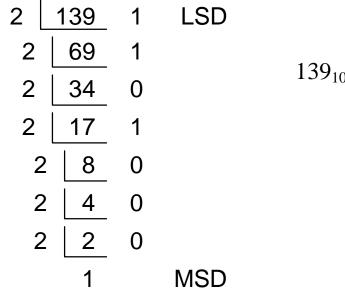
From Hexadecimal to Binary

Replace each **hexadecimal** digit with the corresponding **4-bit** binary string.

 $8B_{16} = 1000 \ 1011 = 10001011_2$

Conversion of Decimal Numbers

From Decimal to Binary



 $139_{10} = 10001011_2$

From Binary to Decimal

 10001011_2

$$= 1 \times 2^{7} + 0 \times 2^{6} + 0 \times 2^{5} + 0 \times 2^{4} + 1 \times 2^{3} + 0 \times 2^{2} + 1 \times 2^{1} + 1 \times 2^{0}$$
$$= 128 + 8 + 2 + 1$$

Conversion of Fractions

Starting at the binary point, group the binary digits that lie to the right into groups of three or four.

 $0.10111_2 = 0.101 \ 110 = 0.56_8$

 $0.10111_2 = 0.1011 \ 1000 = 0.B8_{16}$

Problems

Convert the following

Binary	Octal	Decimal	Hex
10011010			
	2705		
		2705	
			3BC

Binary	Octal	Decimal	Hex
10011010	232	154	9A
10111000101	2705	1477	5C5
101010010001	5221	2705	A91
1110111100	1674	956	3BC

8 🗌	2705	1	16 🗌	2705	1
8	338	2	16	169	9
8	42	2		10=A	

5

Add

	1	1	1	1		1	0	0	1	1
+	1	0	0	1	<u>+</u>		1	1	1	0
1	1	0	0	0	1	0	0	0	0	1

Subtract

1	1	0	0	0	1	1	0	0	1	1
	1	1	1	1	<u>-</u>		1	1	1	1
	1	0	0	1				1	0	0

Multiply

normally

				1	1	1	0	= 14
			*	1	1	0	1	= 13
				1	1	1	0	
			0	0	0	0		
		1	1	1	0			
+	1	1	1	0				_
1	0	1	1	0	1	1	0	

for implementation - add the shifted multiplicands one at a time.

				1	1	1	0	
			*	1	1	0	1	
				1	1	1	0	
		+	0	0	0	0		
			0	1	1	1	0	
	+	1	1	1	0			
	1	0	0	0	1	1	0	
+	1	1	1	0				
1	0	1	1	0	1	1	0	(8 bits)

Divide

1101	
1111)11000101	1101)1011
<u>1111</u>	<u>110</u>
1001101	100
1111	<u>11</u>
10001	1
0000	<u>0</u>
10001	1
<u>1111</u>	
1 0	
1001	
1101)1111001	
<u>1101</u>	
10001	
0000	
10001	
0000	
10001	
<u>1101</u>	
100	

_	110
101)	$1\ 0\ 1\ 1\ 0\ 0\ 1\mid$
	<u>1101</u>
	100101
	1101
	1011
	0000
	1011

Sign-Magnitude

0 = positive 1 = negative $n \text{ bit range} = -(2^{n-1}-1) \text{ to } +(2^{n-1}-1)$ 4 bits range = -7 to +72 possible representation of zero.

2's Complement

flip bits and add one. *n* bit range = $-(2^{n-1})$ to $+(2^{n-1}-1)$ 4 bits range = -8 to +7

0000	= 0
0001	= 1
0010	= 2
0011	
0100	
0101	
0110	
0111	= 7
1000	= -8
1001	= -7
1010	
1011	
1100	
1101	
1110	= -2
1111	= -1

Example

1110	= 14
0001	flip bits
0010	add one WRONG this is not -14. Out of range. Need 5 bits

 $0\ 1\ 1\ 1\ 0 = 14$ 1 0 0 0 1 flip bits 1 0 0 1 0 add one. This is -14.

Sign Extend

add 0 for positive numbers add 1 for negative numbers

Add 2's Complement

$1\ 1\ 1\ 0$	= -2	1110	=-2
+ 1101	= -3	+ 0011	= 3
$\frac{1}{4}$ 1 0 1 1	ignore carry = -5	$\frac{1}{100001}$	ignore carry = 1

Be careful of overflow errors. An addition overflow occurs whenever the sign of the sum if different from the signs of both operands. Ex.

0100	= 4	1100	=-4
+ 0101	= 5	+ 1011	= -5
1001	= -7 WRONG	1 0111	ignore carry = 7 WRONG

Multiply 2's Complement

	1110	= -2	1110	=-2
	* 1101	= -3	<u>* 0011</u>	= 3
	$1\ 1\ 1\ 1\ 1\ 1\ 0$	sign extend to 8 bits	11111110	sign extend to 8 bits
+	0000000		+11111110	
	$1\ 1\ 1\ 1\ 1\ 1\ 1\ 0$		$\frac{1}{1}$ 1 1 1 1 1 0 1 0	ignore carry = -6
+	111110			
	$\frac{1}{1}$ 1 1 1 1 0 1 1 0	ignore carry		
+	00010	negate -2 for sign bit		

$\frac{1}{2}$ 0 0 0 0 0 1 1 0	ignore carry	= 6
-------------------------------	--------------	-----

$1\ 0\ 0\ 1\ 0$	= -14
* 10011	= -13
1111110010	sign extend to 10 bits
+ 111110010	
$\frac{1}{4}$ 1 1 1 1 1 0 1 0 1 1 0	ignore carry
+ 000000000	
1111010110	
+ 0000000	
1111010110	
+ 001110	negate -14 for sign bit
± 0010110110	ignore carry = 182

Floating-Point Numbers

mantissa x (radix)^{exponent}

The floating-point representation always gives us more range and less precision than the fixed-point representation when using the SAME number of digits.

Mantissa sign	Sign exponent	Mantissa magnitude	
General for	mat		
0	1	9	31
Mantissa sign	8-bit excess-127 characteristic	23-bit normalized fraction	
32-bit standard Implied binary point			
0	1	12	63
Mantissa sign	11-bit excess 1023 charactstic	52-bit normalized fraction	
64-bit stand	lard		

Normalized fraction - the fraction always starts with a nonzero bit. e.g.

0.01 ... x 2^{e} would be normalized to 0.1 ... x 2^{e-1} 1.01 ... x 2^{e} would be normalized to 0.101 ... x 2^{e+1}

Since the only nonzero bit is 1, it is usually omitted in all computers today. Thus, the 23-bit normalized fraction in reality has 24 bits.

The exponent is represented in a **biased** form.

- If we take an *m*-bit exponent, there are 2^m possible unsigned integer values.
- Re-label these numbers: 0 to $2^{m-1} \rightarrow -2^{m-1}$ to $2^{m-1}-1$ by subtracting a constant value (or bias) of 2^{m-1} (or sometimes $2^{m-1}-1$).
- Ex. using m=3, the bias $= 2^{3-1} = 4$. Thus the series 0,1,2,3,4,5,6,7 becomes -4,-3,-2,-1,0,1,2,3. Therefore, the true exponent -4 is represented by 0 in the bias form and -3 by +1, etc.
- zero is represented by $0.0 \dots \ge 2^0$.

Ex. if n = 1010.1111, we normalize it to 0.10101111 x 2⁴. The true exponent is +4. Using the 32-bit standard and a bias of $2^{m-1}-1 = 2^{8-1}-1 = 127$, the true exponent (+4) is stored as a biased exponent of 4+127 = 131, or 10000011 in binary. Thus we have

The biased exponent representation is also called **excess** n, where n is $2^{m-1}-1$ (or 2^{m-1}).