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

2	139	1	LSD
2	69	1	
2	34	0	
2	17	1	
2	8	0	
2	4	0	
2	2	0	
	1		MSD

$139_{10} = 10001011_2$

From Binary to Decimal

10001011_2

$= 1\times2^7 + 0\times2^6 + 0\times2^5 + 0\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$

$= 128 + 8 + 2 + 1$

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

$$\begin{array}{r|r l}
8 & 2705 & 1 \\ \hline
8 & 338 & 2 \\ \hline
8 & 42 & 2 \\ \hline
 & 5 &
\end{array}
\qquad\qquad
\begin{array}{r|r l}
16 & 2705 & 1 \\ \hline
16 & 169 & 9 \\ \hline
 & 10=A &
\end{array}$$

Add

```
    1   1   1   1
+   1   0   0   1
1   1   0   0   0
```

```
    1   0   0   1   1
+       1   1   1   0
1   0   0   0   0   1
```

Subtract

```
    1   1   0   0   0
-       1   1   1   1
        1   0   0   1
```

```
    1   0   0   1   1
-       1   1   1   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

```
                 1 1 0 1
1 1 1 1 ) 1 1 0 0 0 1 0 1 |
          1 1 1 1         |
          1 0 0 1 1 0 1   |
            1 1 1 1       |
              1 0 0 0 1   |
              0 0 0 0     |
              1 0 0 0 1   |
                1 1 1 1   |
                    1 0
```

```
                 1 1 0
1 1 0 1 ) 1 0 1 1 0 0 1 |
          1 1 0 1       |
          1 0 0 1 0 1   |
            1 1 0 1     |
              1 0 1 1   |
              0 0 0 0   |
              1 0 1 1
```

```
               1 0 0 1
1 1 0 1 ) 1 1 1 1 0 0 1 |
          1 1 0 1       |
            1 0 0 0 1   |
            0 0 0 0     |
            1 0 0 0 1   |
              0 0 0 0   |
              1 0 0 0 1 |
                1 1 0 1 |
                  1 0 0
```

Sign-Magnitude

0 = positive
1 = negative
n bit range = $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$
4 bits range = -7 to +7
2 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

0 0 0 0	= 0
0 0 0 1	= 1
0 0 1 0	= 2
0 0 1 1	
0 1 0 0	
0 1 0 1	
0 1 1 0	
0 1 1 1	= 7
1 0 0 0	= -8
1 0 0 1	= -7
1 0 1 0	
1 0 1 1	
1 1 0 0	
1 1 0 1	
1 1 1 0	= -2
1 1 1 1	= -1

Example

1 1 1 0	= 14
0 0 0 1	flip bits
0 0 1 0	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
\+ 1 1 0 1 = -3
~~1~~ 1 0 1 1 ignore carry = -5

1 1 1 0 = -2
\+ 0 0 1 1 = 3
~~1~~ 0 0 0 1 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.

0 1 0 0 = 4
\+ 0 1 0 1 = 5
1 0 0 1 = -7 WRONG

1 1 0 0 = -4
\+ 1 0 1 1 = -5
~~1~~ 0 1 1 1 ignore carry = 7 WRONG

Multiply 2's Complement

1 1 1 0 = -2
* 1 1 0 1 = -3
1 1 1 1 1 1 1 0 sign extend to 8 bits
\+ 0 0 0 0 0 0 0
1 1 1 1 1 1 1 0
\+ 1 1 1 1 1 0
~~1~~ 1 1 1 1 0 1 1 0 ignore carry
\+ 0 0 0 1 0 negate -2 for sign bit
~~1~~ 0 0 0 0 0 1 1 0 ignore carry = 6

1 1 1 0 = -2
* 0 0 1 1 = 3
1 1 1 1 1 1 1 0 sign extend to 8 bits
\+ 1 1 1 1 1 1 0
~~1~~ 1 1 1 1 1 0 1 0 ignore carry = -6

1 0 0 1 0 = -14
* 1 0 0 1 1 = -13
1 1 1 1 1 1 0 0 1 0 sign extend to 10 bits
\+ 1 1 1 1 1 0 0 1 0
~~1~~ 1 1 1 1 0 1 0 1 1 0 ignore carry
\+ 0 0 0 0 0 0 0 0
1 1 1 1 0 1 0 1 1 0
\+ 0 0 0 0 0 0 0
1 1 1 1 0 1 0 1 1 0
\+ 0 0 1 1 1 0 negate -14 for sign bit
~~1~~ 0 0 1 0 1 1 0 1 1 0 ignore carry = 182

Floating-Point Numbers

mantissa x (radix)$^{\text{exponent}}$

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 format

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 standard

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

$0.01 \ldots \times 2^{e}$ would be normalized to $0.1 \ldots \times 2^{e-1}$

$1.01 \ldots \times 2^{e}$ would be normalized to $0.101 \ldots \times 2^{e+1}$

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 $0.0 \ldots \times 2^0$.

Ex. if *n* = 1010.1111, we normalize it to 0.10101111×2^4. 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

0 | 1 0 0 0 0 0 1 1 | 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Notice that the first 1 in the normalized fraction is omitted.

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