# 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.
$10001011_{2}=010001011=213_{8}$

## 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.

$$
10001011_{2}=1000 \quad 1011=8 \mathrm{~B}_{16}
$$

## From Octal to Binary

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

$$
213_{8}=010001011=10001011_{2}
$$

## From Hexadecimal to Binary

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

$$
8 \mathrm{~B}_{16}=1000 \quad 1011=10001011_{2}
$$

## Conversion of Decimal Numbers

From Decimal to Binary


From Binary to Decimal
10001011 ${ }_{2}$

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

## Conversion of Fractions

Starting at the binary point, group the binary digits that lie to the right into groups of three or four.
$0.10111_{2}=0.101 \quad 110=0.56_{8}$
$0.10111_{2}=0.1011 \quad 1000=0 . \mathrm{B8}_{16}$

Problems convert the following

| Binary | Octal | Decimal | Hex |
| :---: | :---: | :---: | :---: |
| 10011010 |  |  |  |
|  | 2705 |  |  |
|  |  | 2705 | 3 BC |


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


| 8 | 2705 | 1 | 16 | 2705 | 1 |
| ---: | ---: | ---: | ---: | ---: | ---: |
| 8 | 338 | 2 | 16 | 169 | 9 |
| 8 | 42 | 2 |  | $10=\mathrm{A}$ |  |

## Add

| 1111 |
| ---: |
| $+\quad 10001$ |
| 100000 |

## Subtract

| 11 | 0 | 0 | 0 |  |
| ---: | ---: | ---: | ---: | ---: |
| - | 1 | 1 | 1 | 1 |
|  | 0 | 0 | 1 |  |

## Multiply



|  | 100 | 0 | 1 | 1 |
| ---: | ---: | ---: | ---: | ---: |
| + | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1


| 100 | 0 | 1 |  |
| ---: | ---: | ---: | ---: | ---: |
| - | $1 \quad 1$ | 1 | 1 |
|  | 1 | 0 | 0 |

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

$$
\begin{array}{lllllllll} 
& & & 1 & 1 & 1 & 0 \\
& & & & & \\
& & & 1 & 1 & 0 & 1 \\
& & & 1 & 1 & 0 \\
& & & 0 & 0 & 0 & 0 & & \\
& & 0 & 1 & 1 & 1 & 0 & \\
& & 1 & 1 & 1 & 0 & & & \\
& 1 & 0 & 0 & 0 & 1 & 1 & 0 & \\
& 1 & 1 & 1 & 0 & & & & \\
\hline & 0 & 1 & 1 & 0 & 1 & 1 & 0 & \\
\hline
\end{array}
$$

## Divide

| 1101 | 110 |
| :---: | :---: |
| 1111)11000101\| | 1101)1011001\| |
| 1111 | $\underline{1101}$ |
| 1001101 \| | 100101 |
| $\underline{1111}$ | $\underline{1101}$ |
| 10001 | 1011 |
| $\underline{00001}$ | $\underline{00001}$ |
| 10001 \| | 1011 |
| 1111 |  |
| 10 |  |
| 1001 |  |
| 1101)1111001\| |  |
| $\underline{1101}$ |  |
| 100011 |  |
| $\underline{0000-1}$ |  |
| 100011 |  |
| $\underline{0000-1}$ |  |
| 100011 |  |
| $\underline{1101}$ |  |
| 100 |  |

## Sign-Magnitude

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

## 2's Complement

flip bits and add one.
$n$ bit range $=-\left(2^{n-1}\right)$ to $+\left(2^{n-1}-1\right)$
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
$01110=14$
10001 flip bits
10010 add one. This is -14 .

## Sign Extend

add 0 for positive numbers
add 1 for negative numbers

## Add 2's Complement

| 1110 | $=-2$ |
| ---: | :--- |
| +1101 | $=-3$ |$\quad$| 1110 | $=-2$ |
| ---: | :--- |
| +0011 | $=3$ |
| 11011 | ignore carry $=-5$ |$\quad+0001 \quad$ 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 | $=4$ |
| ---: | :--- |
| +0101 | $=5$ |
| 1001 | $=-7$ WRONG |

$$
\begin{aligned}
1100 & =-4 \\
+\quad 1011 & =-5 \\
\hline+0111 & \text { ignore carry }=7 \text { WRONG }
\end{aligned}
$$

## Multiply 2's Complement

| 1110 | $=-2$ | 1110 | $=-2$ |
| :---: | :---: | :---: | :---: |
| * 1101 | $=-3$ | * 0011 | $=3$ |
| 11111110 | sign extend to 8 bits | 11111110 | sign extend to 8 bits |
| + 0000000 |  | +1111110 |  |
| 11111110 |  | 411111010 | ignore carry $=-6$ |
| $\begin{array}{r}1 \\ +\quad 111110 \\ \hline\end{array}$ |  |  |  |
| 411110110 | ignore carry |  |  |
| $\begin{array}{r} \\ +\quad 00010 \\ \hline\end{array}$ | negate -2 for sign bit |  |  |
| 400000110 | ignore carry $=6$ |  |  |
| 10010 | $=-14$ |  |  |
| * 10011 | $=-13$ |  |  |
| 1111110010 | sign extend to 10 bits |  |  |
| + 111110010 |  |  |  |
| +1111010110 | ignore carry |  |  |
| $\begin{array}{r}+00000000 \\ \hline\end{array}$ |  |  |  |
| 1111010110 |  |  |  |
| $\begin{array}{r} \\ +\quad 0000000 \\ \hline\end{array}$ |  |  |  |
| 1111010110 |  |  |  |
| $\begin{array}{r}1 \\ +\quad 001110 \\ \hline\end{array}$ | negate -14 for sign bit |  |  |
| 40010110110 | ignore carry $=182$ |  |  |

## Floating-Point Numbers

mantissa $x(\text { radix })^{\text {exponent }}$
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 <br> sign | Sign <br> exponent | Mantissa magnitude |
| :---: | :---: | :---: |

General format

| 0 | 1 |  |
| :--- | :--- | :--- |
| Mantissa <br> sign | 8-bit excess-127 <br> characteristic | 23-bit normalized fraction |
| 32-bit standard | Implied binary point |  |
| 0 | 1 | 12 |


| Mantissa <br> sign | 11-bit excess <br> 1023 charactstic | 52-bit normalized fraction |
| :---: | :---: | :---: |

64-bit standard

Normalized fraction - the fraction always starts with a nonzero bit. e.g.
$0.01 \ldots \times 2^{\mathrm{e}}$ would be normalized to $0.1 \ldots \times 2^{\mathrm{e}-1}$
$1.01 \ldots \times 2^{\mathrm{e}}$ would be normalized to $0.101 \ldots \times 2^{\mathrm{e}+1}$
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 $0.10101111 \times 2^{4}$. 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
$0|10000011| 0101111000000000000000$
Notice that the first 1 in the normalized fraction is omitted.
The biased exponent representation is also called excess $\boldsymbol{n}$, where $n$ is $2^{m-1}-1$ (or $2^{m-1}$ ).

