2.1 Binary Numbers

The number system we use is a **positional number system** meaning that the position of each digit has an associated weight.

The value of a given number is equivalent to the weighted sum of all its digits. e.g.

$$1234.56_{10} = 1\times10^{3} + 2\times10^{2} + 3\times10^{1} + 4\times10^{0} + 5\times10^{-1} + 6\times10^{-2}$$

Here, 10 is the **base** or **radix** of the number system.
Use a subscript to indicate the radix of the number.
In general

$$d_{m-1}d_{m-2}\cdots d_1 d_0 d_{-1}d_{-2}\cdots d_n = \sum_{i=-n}^{m-1} \mathrm{d_i}\cdot r^{i}$$

The leftmost digit is called the **most-significant digit** (**MSD**).
The rightmost digit is called the **least-significant digit** (**LSD**).

Digital systems use binary digits with a binary radix.

$$110.101_2 = 1\times2^{2} + 1\times10^{1} + 0\times10^{0} + 1\times10^{-1} + 0\times10^{-2} + 1\times10^{-3} = 6.625_{10}$$

2.3 Number System Conversion

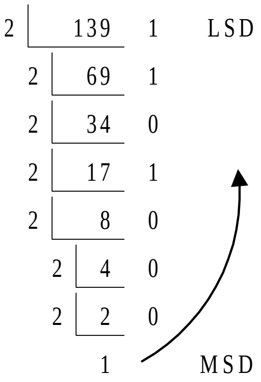
From Binary to Decimal

10001011_2

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

$= 128 + 8 + 2 + 1$

From Decimal to Binary



$139_{10} = 10001011_2$

2.2 Octal and Hexadecimal Numbers

Binary numbers are too long to write so we use a shorthand notation:
Octal – base 8; needs 8 different values; 0 to 7.
Hexadecimal – base 16; needs 16 different values; 0 to 9, A to F.

Binary (radix 2)	Octal (radix 8)	Decimal (radix 10)	Hexadecimal (radix 16)
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F

2.3 Number System Conversion

From Binary to Octal

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

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

$$11.10111_2 = 011\ .\ 101\ \ 110 = 3.56_8$$

From Octal to Binary

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

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

From Binary to Hexadecimal

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

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

$$11.10111_2 = 0011\ .\ 1011\ \ 1000 = 3.B8_{16}$$

From Hexadecimal to Binary

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

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

From Octal to Decimal

5221_8

$= 5\times8^3 + 2\times8^2 + 2\times8^1 + 1\times8^0$

$= 2560 + 128 + 16 + 1$

$= 2705_{10}$

From Decimal to Octal

```
8 | 2705   1   LSD
  ------
   8 | 338   2
     -----
   8 |  42   2
     -----
         5       MSD
```

$2705_{10} = 5221_8$

From Hexadecimal to Decimal

$A9C_{16}$

$= 10\times16^2 + 9\times16^1 + 12\times16^0$

$= 2560 + 144 + 12$

$= 2716_{10}$

From Decimal to Hexadecimal

```
16 | 2716   C   LSD
   ------
   16 | 169   9
      -----
          A       MSD
```

$2716_{10} = A9C_{16}$

Examples:

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

2.4 Add

$$\begin{array}{ccccc} & 1 & 1 & 1 & 1 \\ + & 1 & 0 & 0 & 1 \\ \hline 1 & 1 & 0 & 0 & 0 \end{array}$$

$$\begin{array}{cccccc} & 1 & 0 & 0 & 1 & 1 \\ + & & 1 & 1 & 1 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 1 \end{array}$$

2.4 Subtract

$$\begin{array}{cccccc} & 1 & 1 & 0 & 0 & 0 \\ - & & 1 & 1 & 1 & 1 \\ \hline & & 1 & 0 & 0 & 1 \end{array}$$

$$\begin{array}{cccccc} & 1 & 0 & 0 & 1 & 1 \\ - & & 1 & 1 & 1 & 1 \\ \hline & & & 1 & 0 & 0 \end{array}$$

2.7 Multiply

normally

$$\begin{array}{ccccccccl} & & & & 1 & 1 & 1 & 0 & = 14 \\ & & & * & 1 & 1 & 0 & 1 & = 13 \\ \cline{4-8} & & & & 1 & 1 & 1 & 0 & \\ & & & 0 & 0 & 0 & 0 & & \\ & & 1 & 1 & 1 & 0 & & & \\ + & 1 & 1 & 1 & 0 & & & & \\ \hline 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & \end{array}$$

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

$$\begin{array}{ccccccccl} & & & & 1 & 1 & 1 & 0 & \\ & & & * & 1 & 1 & 0 & 1 & \\ \cline{4-8} & & & & 1 & 1 & 1 & 0 & \\ & & + & 0 & 0 & 0 & 0 & & \\ \cline{3-8} & & & 0 & 1 & 1 & 1 & 0 & \\ & + & 1 & 1 & 1 & 0 & & & \\ \cline{2-8} & 1 & 0 & 0 & 0 & 1 & 1 & 0 & \\ + & 1 & 1 & 1 & 0 & & & & \\ \cline{1-8} 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & \text{(8 bits)} \end{array}$$

2.8 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
```

2.5 Representation of Negative Numbers

Sign-Magnitude

The most significant bit is the sign bit and the rest of the number is the 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, "+0" and "-0".

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	= 3
0 1 0 0	= 4
0 1 0 1	= 5
0 1 1 0	= 6
0 1 1 1	= 7
1 0 0 0	= -8
1 0 0 1	= -7
1 0 1 0	= -6
1 0 1 1	= -5
1 1 0 0	= -4
1 1 0 1	= -3
1 1 1 0	= -2
1 1 1 1	= -1

Example

0 1 1 0	= 6
1 0 0 1	flip bits
1 0 1 0	add one = -6

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

$$\begin{array}{rrl} & 1\ 1\ 1\ 0 & = -2 \\ \underline{+} & \underline{1\ 1\ 0\ 1} & = -3 \\ & \cancel{1}\ 1\ 0\ 1\ 1 & \text{ignore carry} = -5 \end{array}$$

$$\begin{array}{rrl} & 1\ 1\ 1\ 0 & = -2 \\ \underline{+} & \underline{0\ 0\ 1\ 1} & = 3 \\ & \cancel{1}\ 0\ 0\ 0\ 1 & \text{ignore carry} = 1 \end{array}$$

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

$$\begin{array}{rrl} & 0\ 1\ 0\ 0 & = 4 \\ \underline{+} & \underline{0\ 1\ 0\ 1} & = 5 \\ & 1\ 0\ 0\ 1 & = -7\ \text{WRONG} \end{array}$$

$$\begin{array}{rrl} & 1\ 1\ 0\ 0 & = -4 \\ \underline{+} & \underline{1\ 0\ 1\ 1} & = -5 \\ & \cancel{1}\ 0\ 1\ 1\ 1 & \text{ignore carry} = 7\ \text{WRONG} \end{array}$$

Subtract 2's Complement

$$\begin{array}{rrl} & 0\ 0\ 1\ 0 & = 2 \\ \underline{+} & \underline{1\ 1\ 1\ 0} & = -2 \\ & \cancel{1}\ 0\ 0\ 0\ 0 & \text{ignore carry} = 0 \end{array}$$

Multiply 2's Complement

$$\begin{array}{rrl} & 1\ 1\ 1\ 0 & = -2 \\ \underline{*} & \underline{1\ 1\ 0\ 1} & = -3 \\ & 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0 & \text{sign extend to 8 bits} \\ \underline{+} & \underline{0\ 0\ 0\ 0\ 0\ 0\ 0} & \\ & 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0 & \\ \underline{+} & \underline{1\ 1\ 1\ 1\ 1\ 0} & \\ & \cancel{1}\ 1\ 1\ 1\ 1\ 0\ 1\ 1\ 0 & \text{ignore carry} \\ \underline{+} & \underline{0\ 0\ 0\ 1\ 0} & \text{negate -2 for sign bit} \\ & \cancel{1}\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0 & \text{ignore carry} = 6 \end{array}$$

$$\begin{array}{rrl} & 1\ 1\ 1\ 0 & = -2 \\ \underline{*} & \underline{0\ 0\ 1\ 1} & = 3 \\ & 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0 & \text{sign extend to 8 bits} \\ \underline{+} & \underline{1\ 1\ 1\ 1\ 1\ 1\ 0} & \\ & \cancel{1}\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 0 & \text{ignore carry} = -6 \end{array}$$

$$\begin{array}{rrl} & 1\ 0\ 0\ 1\ 0 & = -14 \\ \underline{*} & \underline{1\ 0\ 0\ 1\ 1} & = -13 \\ & 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 0 & \text{sign extend to 10 bits} \\ \underline{+} & \underline{1\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 0} & \\ & \cancel{1}\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 0 & \text{ignore carry} \\ \underline{+} & \underline{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0} & \\ & 1\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 0 & \\ \underline{+} & \underline{0\ 0\ 0\ 0\ 0\ 0\ 0} & \\ & 1\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 0 & \\ \underline{+} & \underline{0\ 0\ 1\ 1\ 1\ 0} & \text{negate -14 for sign bit} \\ & \cancel{1}\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0 & \text{ignore carry} = 182 \end{array}$$

2.9 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… 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 → -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 … x 2^0.

Ex. if *n* = 1010.1111, we normalize it to 0.10101111 x 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}).

2.10 Binary Coded Decimals (BCD)

2.11 Character Codes (ASCII – The American Standard Code for Information Interchange)