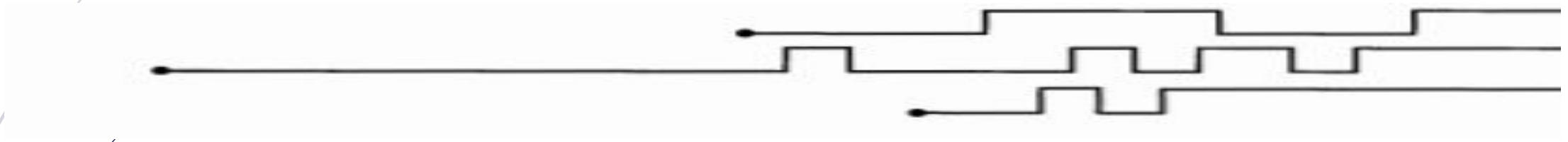


Chapter 1

Number systems

Dr. Iness NEDJI MILAT

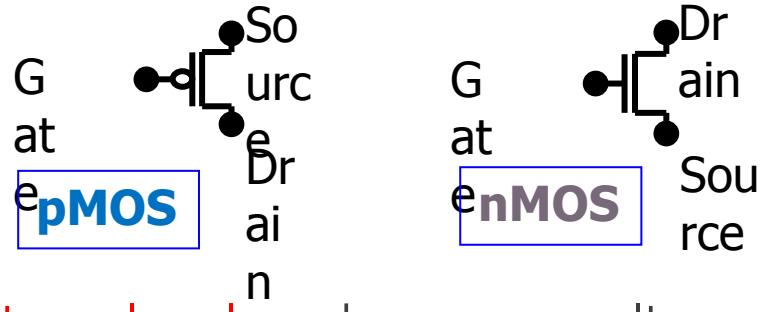
iness.nedji@ensia.edu.dz

Digital systems

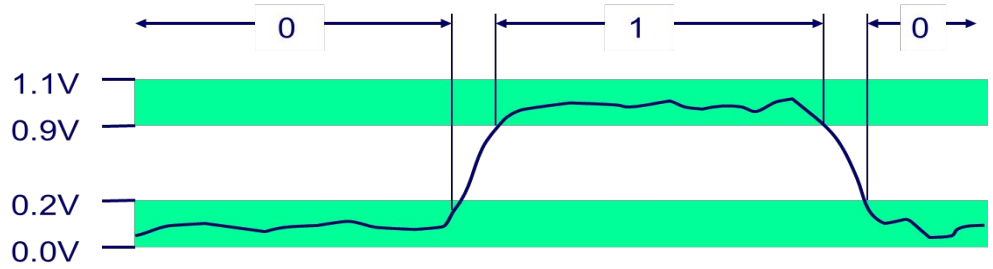
- Digital system is **a combination of devices** (most often electronic) designed to manipulate information that are represented in digital form (discrete values).
- Some of the more familiar digital systems include digital computers,
- Information processed by computer can be on different formats : number, text, image, audio, video,
- All these formats are represented in numeric values (number systems),
- In computer, they are always coded in binary form (BIT: BinarydigIT),
- Bits : sequence of 0 and 1.

Bits : Binary digITs

- Computer systems do not represent numeric values using the decimal system but using binary system.
- Computers are built around the idea of two states : true (1) and false (0).
- Transistors represent this in hardware, and bits represent this in software!

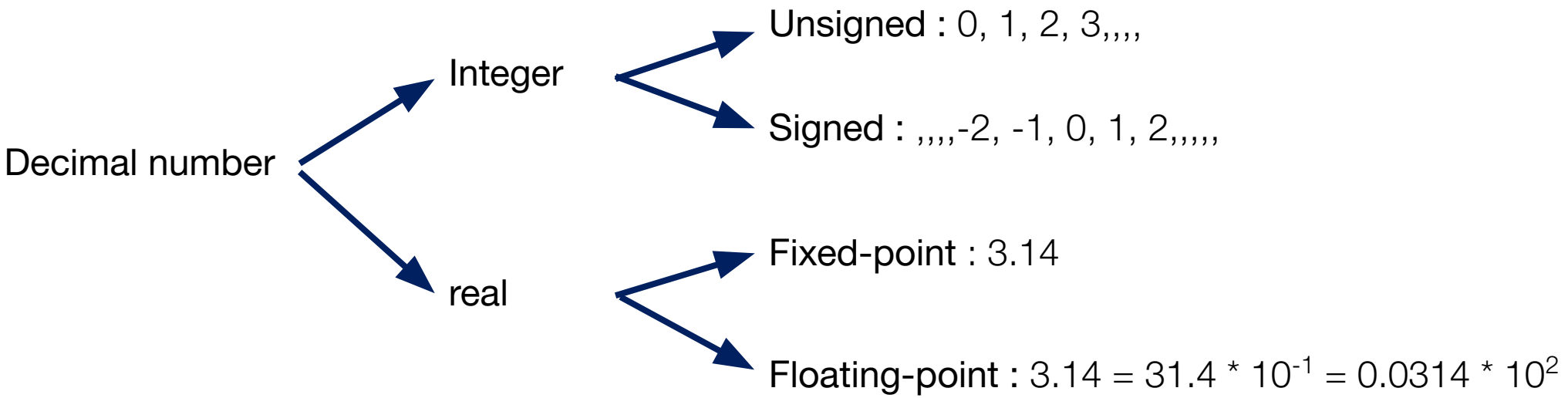


- Bit (Binary digIT) is represented by two voltage levels, where any voltage between 0 and 0.2 V represents a 0 and any voltage between 0,9 and 1,1 Vrepresents a 1.



Number systems

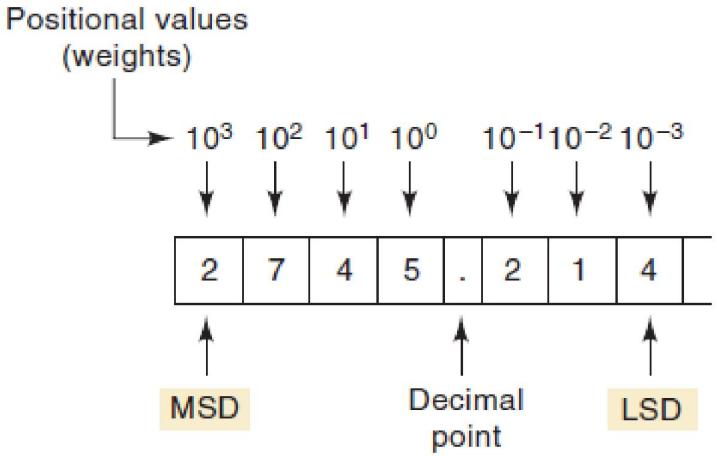
- All data in digital systems are represented in **number** systems,
- Many number systems are in use in digital technology. The most common are the **decimal**, **binary**, **octal**, and **hexadecimal** systems.
- The decimal system is clearly the most familiar to us because it is a tool that we use every day.



Decimal System : base 10

- The decimal system is called the base (radix)10 system because it has 10 digits (0,1,2,3,4,5,6,7,8,9)
- The decimal system is **a positional-value system** in which the value of a digit depends on its position.

For example :



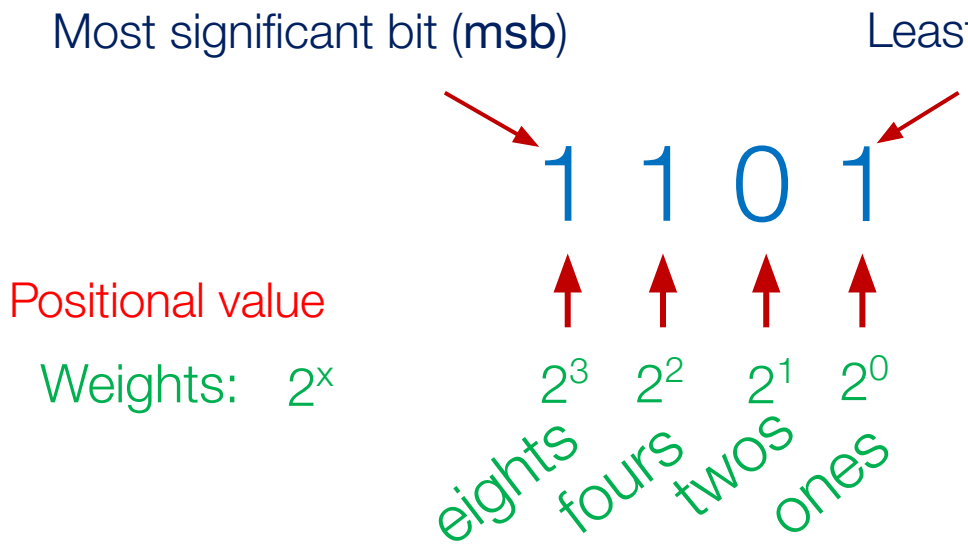
$2745 = 2*10^3 + 7*10^2 + 4*10^1 + 5*10^0$ (Polynomial representation)

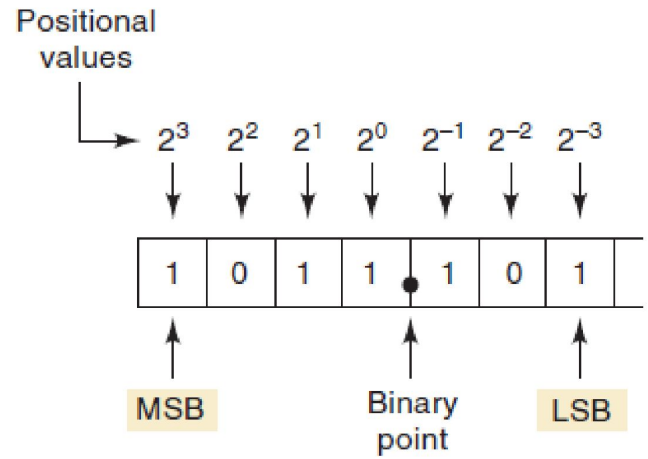
$2745.214 = 2*10^3 + 7*10^2 + 4*10^1 + 5*10^0 + 2*10^{-1} + 1*10^{-2} + 4*10^{-3}$

The 2 carries the most weight; it is referred to as the most significant digit (MSD).The 5 carries the least weight and is called the least significant digit (LSD).

Binary system: Base 2

- The Binary Number System uses base (radix) 2.
- It includes the digits 0 and 1.
- For example:





$$(1101)_2 = 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 8 + 4 + 0 + 1 = 13_{10}$$

$$(1011.101)_2 = 2^3 + 2^1 + 2^0 + 2^{-1} + 2^{-3}$$

Number system: generalization

- Number X in radix r system :

$$(x)_r = x_{n-1}x_{n-2} \ldots x_1x_0 \cdot x_{-1}\, x_{-2} \ldots x_{-(m-1)}\, x_{-m}$$

- Decimal Value of X :

$$Val_{10} = x_{n-1} \times r^{n-1} + x_{n-2} \times r^{n-2} + \ldots + x_1 \times r + x_0 + x_{-1} \times r^{-1} + x_{-2} \times r^{-2} + \ldots + x_{-m} \times r^{-m}$$

position

radix

- Examples:
  - $(11011.01)_2 = 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2 + 1 + 0\times2^{-1} + 1\times2^{-2} = (27.25)_{10}$
  - $(2103)_4 = 2\times4^3 + 1\times4^2 + 0\times4 + 3 = (147)_{10}$
  - $(156.7)_8 = (1x8^2 + 5x8^1 + 6x8^0 + 7x8^{-1})_{10} = (64 + 40 + 6 + 0.875)_{10} = (110.875)_{10}$
  - $(55)_9 = (45 + 5)_{10} = 50_{10}$
  - $(12)_{12} = (12 + 2)_{10} = 14_{10}$

Octal and hexadecimal systems (1)

- When working with bits, oftentimes we have strings with 32 or 64 bits (a large number of bits!!).
- So, it is more convenient and less error-prone to write the binary numbers in intermediate base systems such as **Octal** (base 8) or **Hexadecimal** systems (base 16),
- Octal and Hex are often used in a digital system as sort of a "shorthand" way, more compact, to represent strings of bits.
- However, it is important to keep in mind that digital circuits **all work in binary**.
- Octal and Hex are simply used as a convenience for the humans involved.

Octal and hexadecimal systems (2)

- Base 8 (octal) numbers : 0,1,2,3,4,5,6,7
  - one octal digit is formed by grouping 3 bits.

$$100001010110_2 = 100\ 001\ 010\ 110$$
$$= 4\ 1\ 2\ 6_8$$

- Base 16 (Hexadecimal) is most useful as a “shorthand” notation for binary numbers.
- Hex numbers : 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
  - one hexadecimal digit is formed by grouping 4 bits and start with 0x

$$100001010110_2 = 1000\ 0101\ 0110$$
$$= 8\ 5\ 6_{16}$$

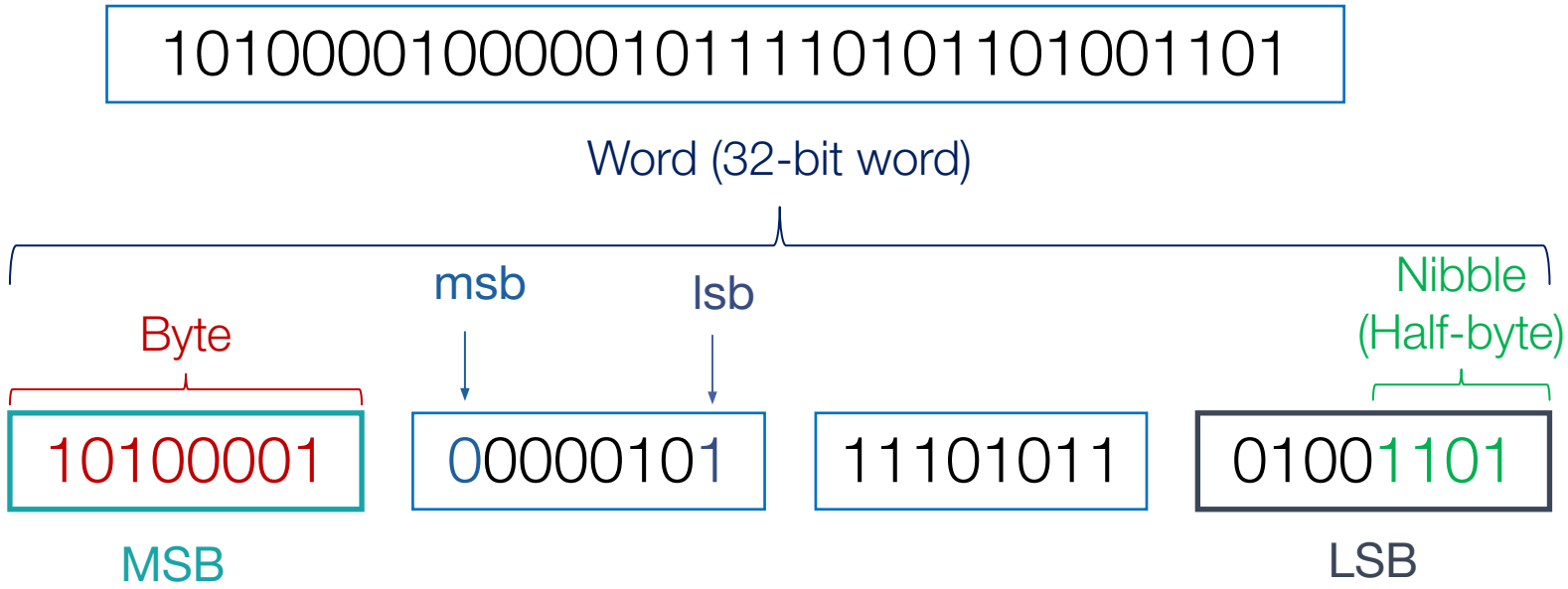
$$= 0x\ 856$$

Counting . . . 2, 8, 10, 16

Decimal	Binary	Octal	Hexadecimal
0	00000	000	00
1	00001	001	01
2	00010	002	02
3	00011	003	03
4	00100	004	04
5	00101	005	05
6	00110	006	06
7	00111	007	07
8	01000	010	08
9	01001	011	09
10	01010	012	0A
11	01011	013	0B
12	01100	014	0C
13	01101	015	0D
14	01110	016	0E
15	01111	017	0F
16	10000	20	10

Bytes, Nibbles, Words, MSB, LSB, …

- It's quite tedious to work with only binary numbers…



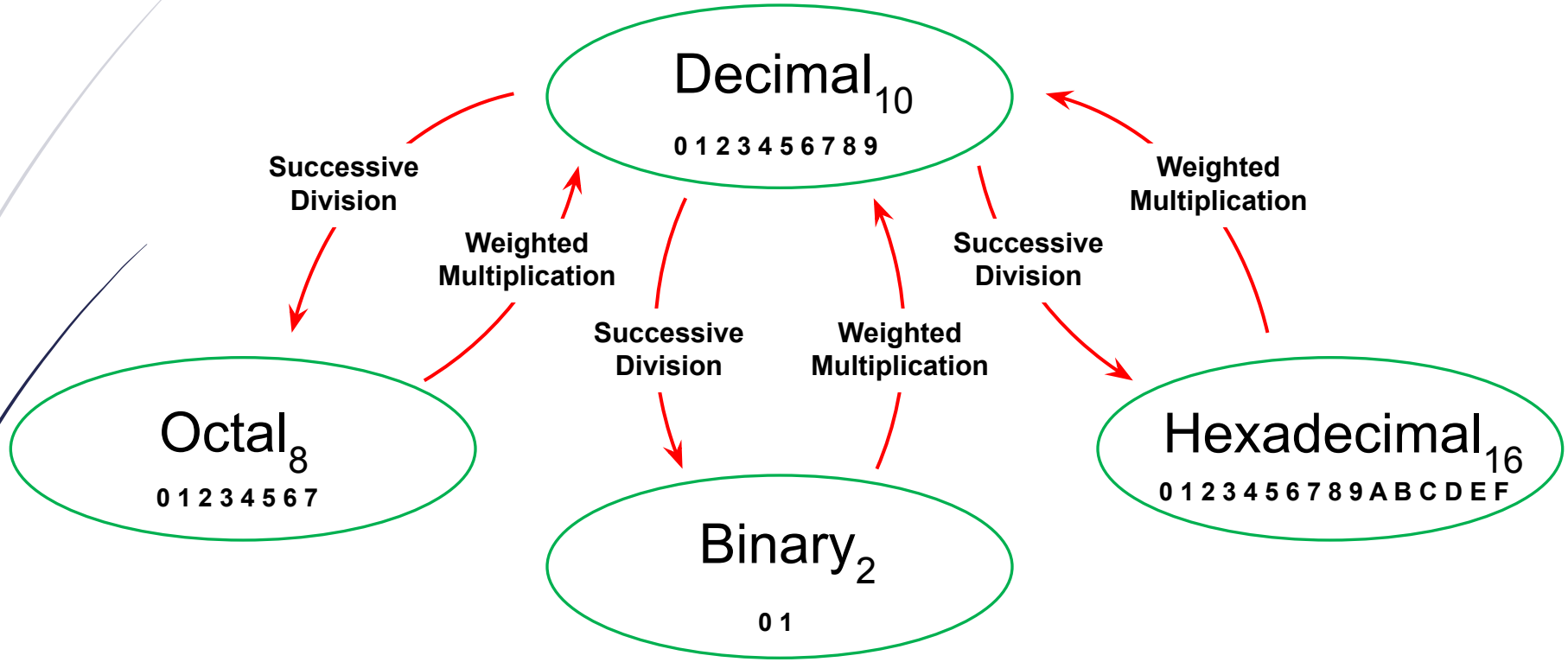
- In a group of bits, the least significant bit (lsb) is to the right. The most significant bit (msb) is to the left.
- Within a word, the terms are the most significant byte (MSB) and the least significant byte (LSB).

Range of values

- In digital computers, the range of numbers that can be represented is based on the number of bits available in the hardware structures that store and process information.
- What is the total range of decimal values that can be represented in a single byte (8 bits)?
  - minimum = 0000 0000 = 0
  - maximum = 1111 1111 = 255
    - Method 1: $1*2^7 + 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = 255$
    - Method 2: $2^8 - 1 = 255$
- The maximum for n bits is $2^n - 1$

Conversion from/to base x

Converting To and From Decimal



base X to Decimal Conversion

- Any number in base 2, 8 or 16 can be converted to its decimal equivalent by **multiplying each digit by its positional weight**.

$$(11010)_2 = 1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 = (26)_{10}$$

$$(127.4)_8 = 1 * 8^2 + 2 * 8^1 + 7 * 8^0 + 4 * 8^{-1} = (87.5)_{10}$$

$$(B65F)_{16} = 11 * 16^3 + 6 * 16^2 + 5 * 16^1 + 15 * 16^0 = (46687)_{10}$$

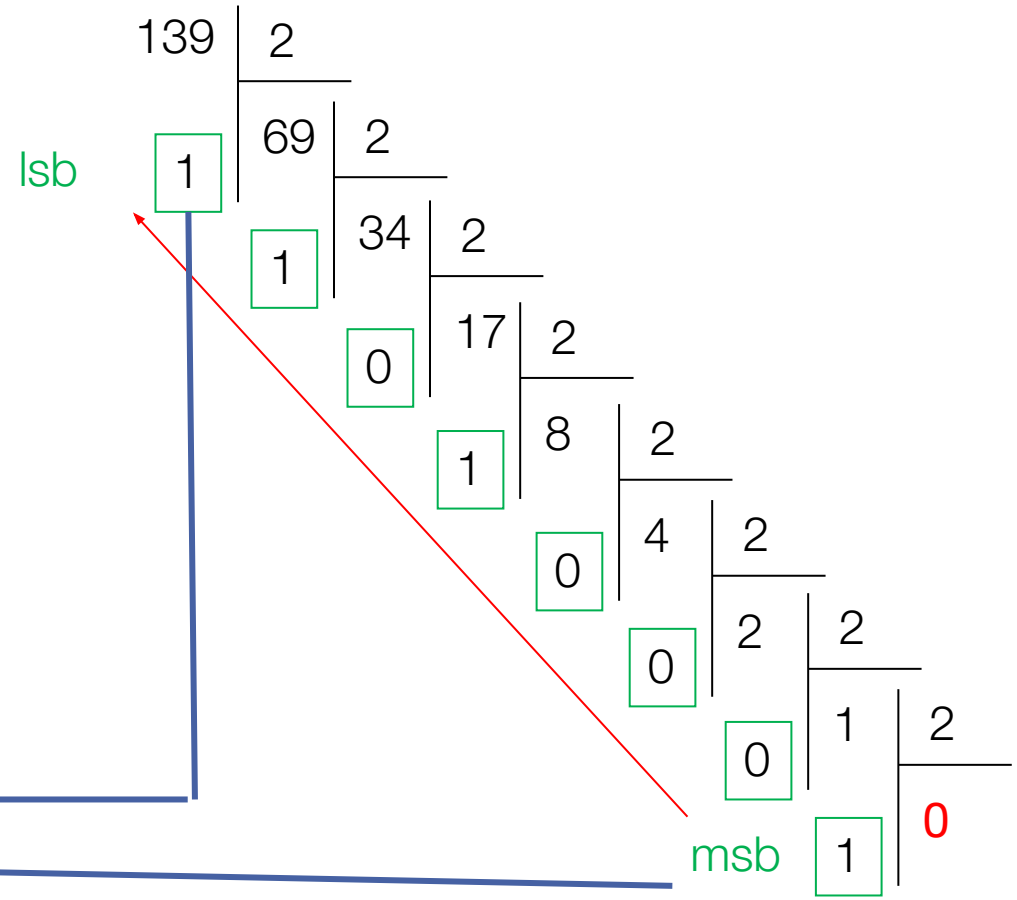
Decimal to Binary Conversion : Integer

Repeated Division Method

- Divide the decimal number by 2.
- Write the remainder after each division until a quotient of zero is obtained.
- The first remainder is the lsb.
- The last is the msb.

Example

- Convert to binary $(139)_{10}$



$(139)_{10}=(10001011)_2$

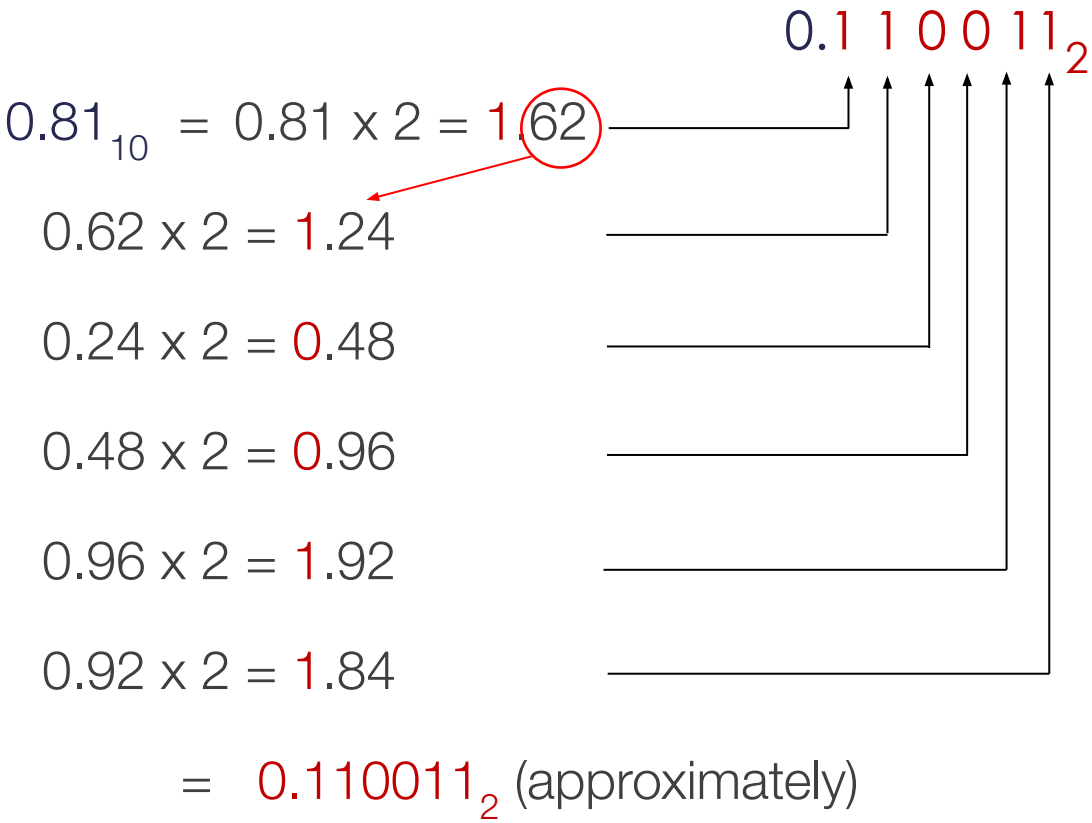
Decimal to Binary Conversion : Fraction

Repeated Multiplication Method

- Multiply the fractional part by 2.
- The digit to the left of the decimal point in the product will be 0 or 1 and contributes to the binary representation,
- The fractional part of the product is used as the multiplicand in the next step.

Example

- Convert to the binary 0.81_{10}



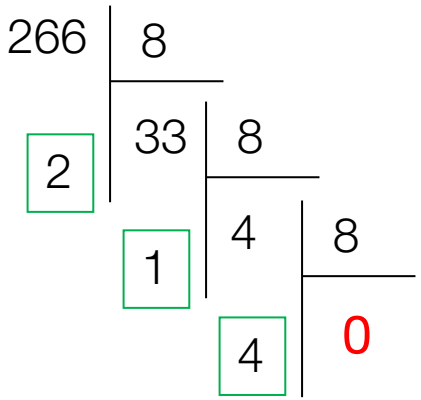
$= 0.110011_2$ (approximately)

Decimal to Octal/Hex Conversion

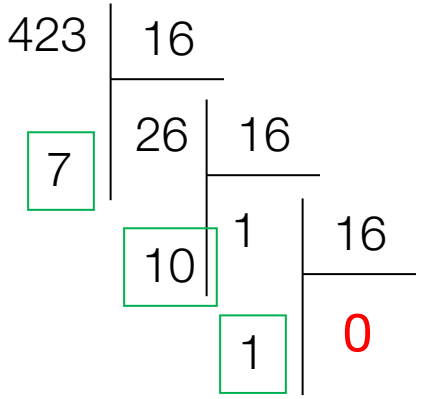
A decimal can be converted to octal (resp. hexadecimal) by using the same repeated-division method that used in binary conversion, but with a division factor of 8 (resp. 16) instead 2,

- Example

$266 = (412)_8$



$423 = (1A7)_{16} = 0x1A7$



Binary and Octal conversions

Octal to Binary

- Replace each octal digit with its equivalent 3-bit binary sequence.
- Example:

261.35_8 = 2 6 1 . 3 5 $_8$
= 010 110 001 . 011 101 $_2$

Binary to Octal

- Make groups of 3 bits, starting from the binary point.
- Add 0s to the ends of the number if needed.
- Convert each bit group to its corresponding octal digit.
- Example :

10110100.001011_2 = 010 110 100. 001 011 $_2$
= 2 6 4 . 1 3 $_8$

Binary and Hex conversions

Hexadecimal to Binary

- Replace each hex digit with its equivalent 4-bit binary sequence.
- Example:

261.35_{16} = 2 6 1 . 3 5_{16}

= 0010 0110 0001 . 0011 0101_2

Binary to Hexadecimal

- Make groups of 4 bits, starting from the binary point.
- Add 0s to the ends of the number if needed.
- Convert each bit group to its corresponding hex digit.
- Example :

10110100.001011_2 = 1011 0100. 0010 1100_2

= B 4 . 2 C_{16}

Hex	Binary
0	0000
1	0001
2	0010
3	0011

Hex	Binary
4	0100
5	0101
6	0110
7	0111

Hex	Binary
8	1000
9	1001
A	1010
B	1011

Hex	Binary
C	1100
D	1101
E	1110
F	1111

Binary arithmetic

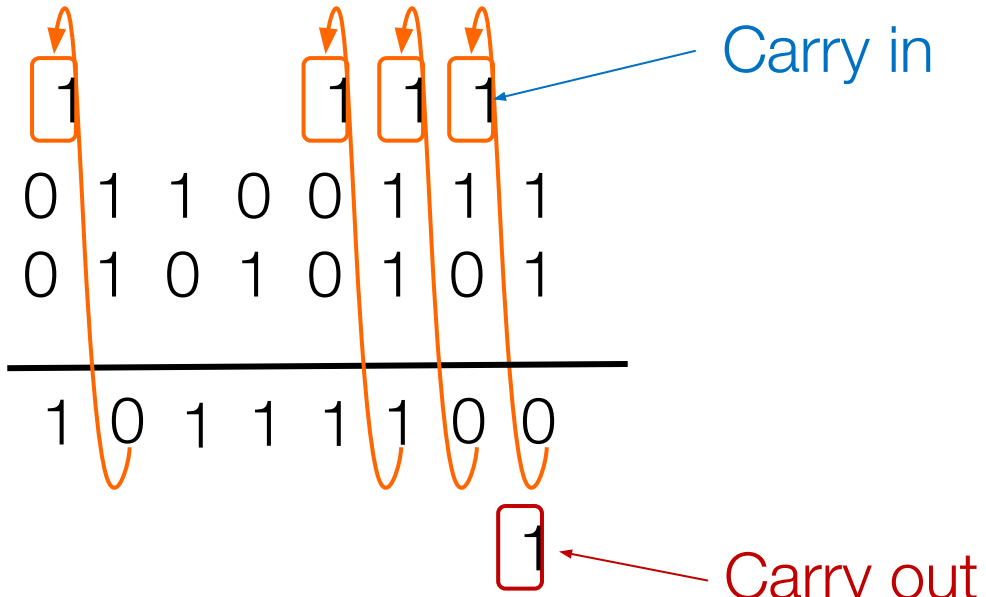
Binary Addition

- The elementary operations are:

```
  0      0      1      1      1
+ 0    + 1    + 0    + 1    + 1
---    ---    ---    ---    ---
  0      1      1     10     11
```

- Compute 103+ 85

```
  103   ⟶
+  85   ⟶ +
-----
  188   ⟶
```



Octal and Hexadecimal Addition

- Compute

$$\begin{array}{r} 25_8 \\ +\ 12_8 \\ \hline 37_8 \end{array} \qquad \begin{array}{r} 55_8 \\ +\ 66_8 \\ \hline 143_8 \end{array} \qquad \begin{array}{r} 3_8 \\ +\ 7_8 \\ \hline 12_8 \end{array}$$

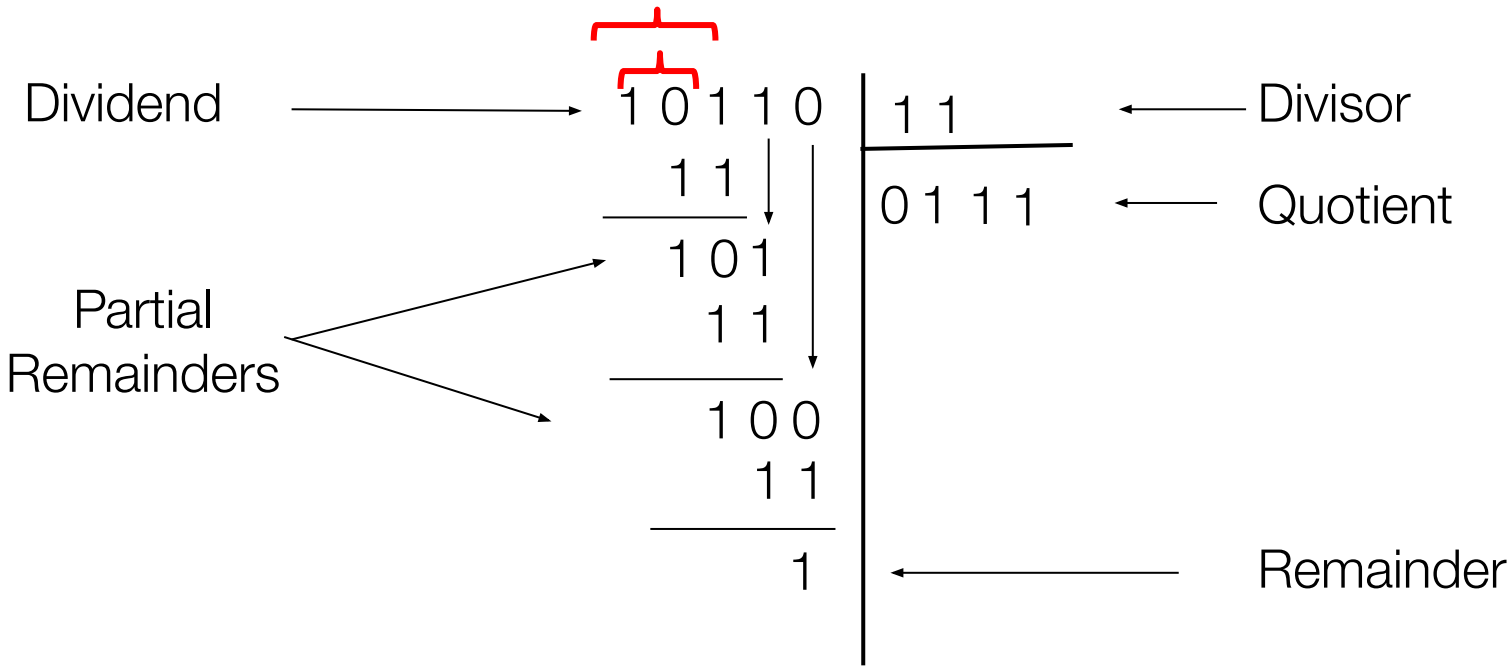
$$\begin{array}{r} 15_{16} \\ +\ 0A_{16} \\ \hline 1F_{16} \end{array} \qquad \begin{array}{r} 2D_{16} \\ +\ 36_{16} \\ \hline 63_{16} \end{array} \qquad \begin{array}{r} 3_{16} \\ +\ 7_{16} \\ \hline A_{16} \end{array}$$

Binary Multiplication

- Notes:
  - if multiplier bit is 1 copy multiplicand (place value)
    - otherwise zero
  - Note: need double length result

```
      1 0 1 1      Multiplicand
    x 1 1 0 1      Multiplier
-------------------
      1 0 1 1
    0 0 0 0 •
  1 0 1 1 • •
1 0 1 1 • • •
-------------------
1 0 0 0 1 1 1 1
```

Binary Division



Binary Substraction

How do we subtract two binary numbers?

$5 - 3 = ?$

$101 - 11 = ?$ ✗ materially

Equivalent to adding with a negative number

$5 + (-3) = ?$

$101 + ? = ?$

How can we represent negative numbers?

Integer

Unsigned : 0, 1, 2, 3,…

$(139)_{10}=(10001011)_2$

Signed : ..,-2, -1, 0, 1, 2,..

Negative

$(-139)_{10}=(?)_2$

Positive

$(+139)_{10}=(?)_2$

Sign/Magnitude

One's complement

Two's complement

Presentation including the sign bit

Sign/Magnitude Representation

- Let msb represent the sign (0=positive, 1=negative)
- and the remaining N-1 bits for magnitude
  - 0111 = 7
  - 1111 = -7
- Problems
  1. Two zero's: $+0_{10}$ (0000_2) different than -0_{10} (1000_2)
  2. Addition does not work

$$\begin{array}{r} 2_{10} \\ +\ -5_{10} \\ \hline -3_{10} \end{array} \longrightarrow + \begin{array}{r} 0\ 0\ 1\ 0_2 \\ 1\ 1\ 0\ 1_2 \\ \hline 1\ 1\ 1\ 1_2 \end{array} = -7_{10}$$

One's complement

- Leading 0's for positive and 1's for negative
- Negative numbers: complement the positive number
  - Example:
    - $+7_{10}$: 0111_2
    - -7_{10} : 1000_2
- Problem
  - Two zero's still: $+0_{10}$ (0000_2) different than -0_{10} (1111_2)

Two's complement

The 2's-complement system for representing signed numbers works like this:

- If the number is positive, the magnitude is represented in its true binary form, and a sign bit of 0 is placed in front of the MSB.

$$+45 = 0\ 1\ 0\ 1\ 1\ 0\ 1$$

- If the number is negative, apply the 2's complement form :

1. Obtain the bit pattern of positive counterpart,
2. Invert the bit pattern (1's complement),
3. Add one to it.

$$\begin{array}{rl} & 0\ 1\ 0\ 1\ 1\ 0\ 1 = +45 \\ & \downarrow\ \downarrow\ \downarrow\ \downarrow\ \downarrow\ \downarrow\ \downarrow \\ & 1\ 0\ 1\ 0\ 0\ 1\ 0 \\ + & \qquad\qquad\qquad 1 \\ \hline & 1\ 0\ 1\ 0\ 0\ 1\ 1 = -45 \end{array}$$

Two's complement : examples

- $(-9)_{10} = (?)_2$ represented in 5 bits

$(-9)_{10}$ = 2's complement (+9) = 2's complement (01001)

= 10110 + 1 = 10111_2

- $(10111)_2 = (?)_{10}$

$(10111)_2$ = - (2's complement (10111))

negative

= - (01000 + 1)

= - (01001)

= $-\ 9_{10}$

- $(00111)_2 = (?)_{10}$

$(00111)_2$ = + $(1*2^2 + 1*2^1 + 1*2^0)$

positive

= $+\ 7_{10}$

Range of values

0000 = 0
0001 = 1
...
1111 = 15

Integer Decimal

(n bits)

4 bits

Unsigned integer

0, 1, 2, 3..

Presentation : true binary form

Range values : $[0, 2^n - 1]$

[0, 15]

Signed integer

.., -2, -1, 0, 1, 2,..

Presentation : 2's complement

Range values : $[- 2^{n-1}, 2^{n-1} - 1]$

[- 8, 7]

4 bits Two's complement signed integer

Only one number represent 0

Binary	Decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

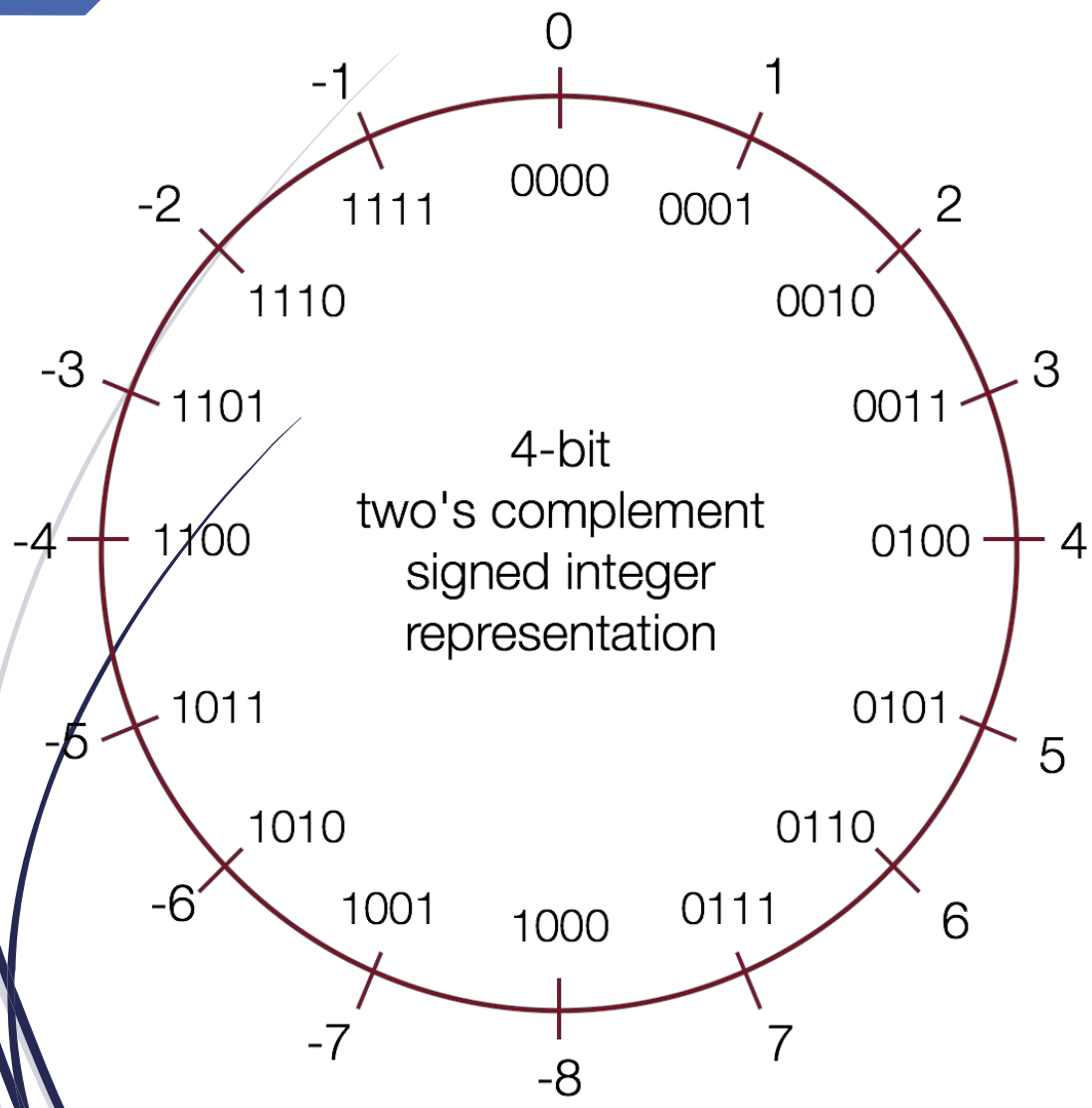
Positive numbers are represented as usual.

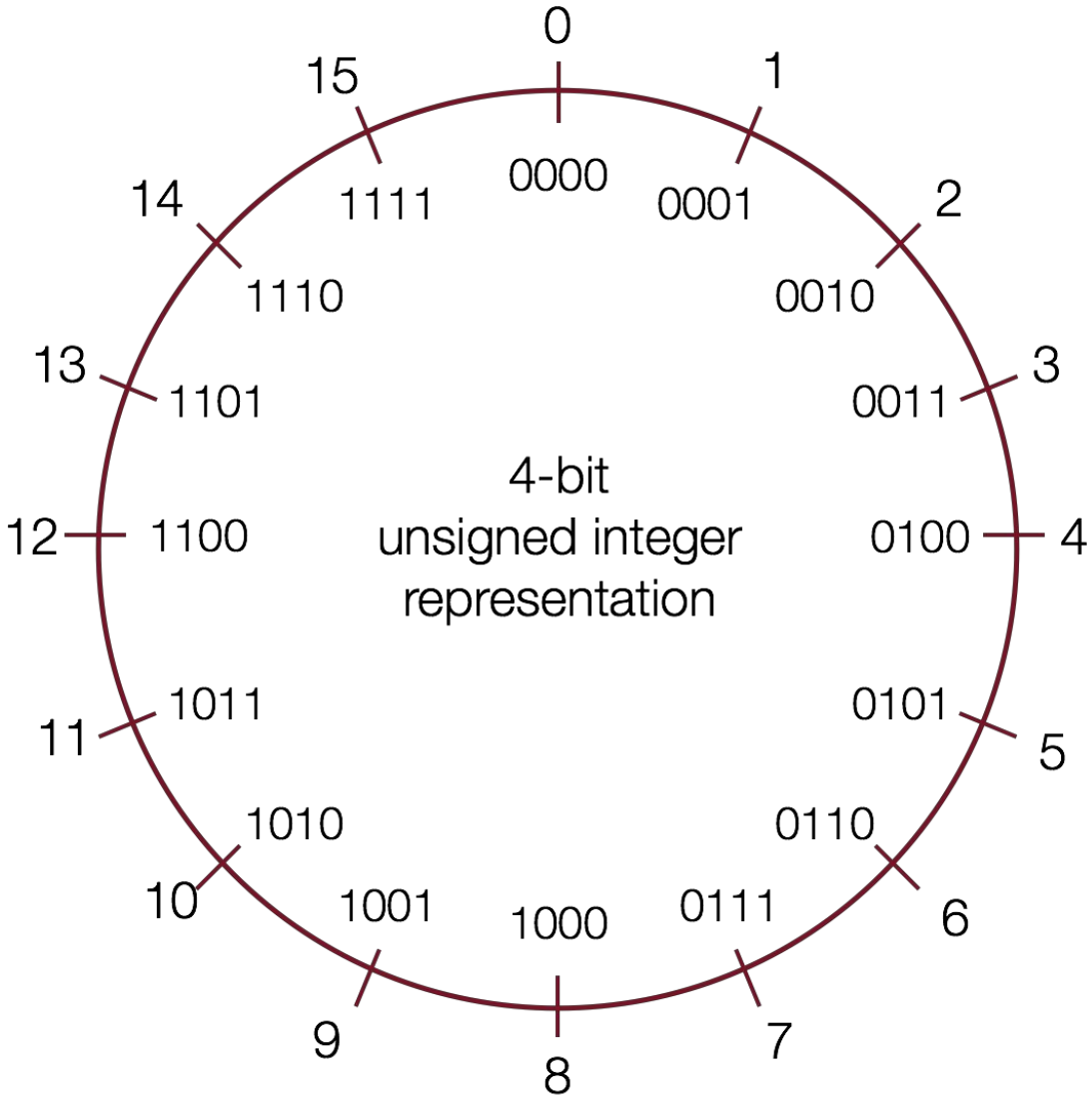
Can represent one more negative number than positive numbers.

Negative numbers are listed in reverse order.

All negative numbers have msb set. This is called the *sign bit*.

Signed vs Unsigned Number Wheels





Arithmetic Operations with Signed Numbers

- Rules for addition
  - Use the signed number notation with negative numbers in 2's complement form
  - Add the two signed numbers.
  - Discard any final carries.
  - The result is in signed form.

```
00011110 = +30
00001111 = +15
--------------
00101101 = +45
```

```
00001110 = +14
11101111 = -17
--------------
11111101 =  -3
```

```
11111111 =  -1
11111000 =  -8
--------------
111110111 =  -9
```

Discard carry

Overflow

- Sometimes result can't be represented with given number of bits
- Either too large magnitude of positive or negative
- Oveflow occurs only if both numbers have the same sign.
- Example 1 : For 4-bit representation
  - 0111+0001 = 1000 WRONG
  - 1000 in two's complement is -8, not +8
- Example 2: For 8-bit representation

```
 01000000 = +64
 01000001 = +65
----------------
 10000001 = -127
```

```
 10000001 = -127
 10000001 = -127
----------------
 100000010 = +2
```

Discard carry

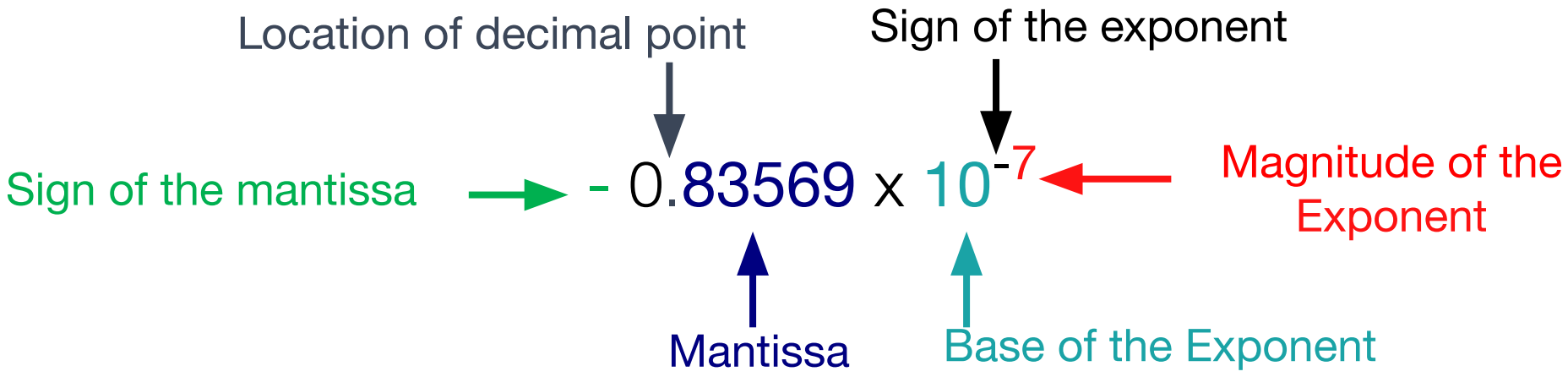
- The overflow ⇔ Sign bit of the result different than the sign bit of the two added numbers.

Real numbers

- Our decimal system handles Two types of non-integer *real* numbers
  - fixed point
    - Using the decimal point (.) to make a notation:
    - $3.78 = 3.10^{3} + 7.10^{-1} + 8.10^{-2}$
  - floating point (called also scientific notation)
    - Used to represent very large and very small numbers (integer or real), with the precision needed
    - Unit of electric charge e = $1.602\ 176\ 462 \times 10^{-19}$ Coulomb
    - Volume of universe = $1 \times 10^{85}\ cm^{3}$

Floating point numbers

- 4 specifications required for a number
  1. Sign
  2. Magnitude or mantissa or Significand
  3. Sign of the exponent
  4. Magnitude of the exponent
- Plus
  5. Base of the exponent
  6. Location of decimal point (or other base) radix point



Floating-point numbers normalization

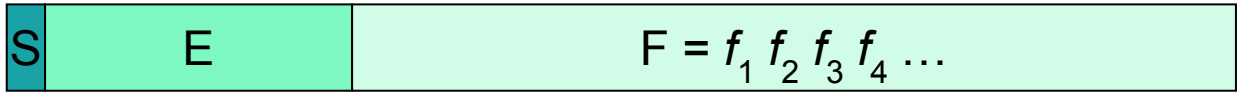
- The following representations are all the same in value
  - $+ 0.110 \times 2^{5}$
  - $+ 0.00110 \times 2^{7}$
  - $+ 11000000 \times 2^{(-3)}$
- Floating-point numbers should be normalized
- Idea
  - Exactly one non-zero digit should appear before the point
    - In a binary number, this digit should be 1
  - Normalized: -1.101101×2^{-3}
  - Not Normalized: 1101.101×2^{-6}

Real numbers in binary

- We mimic the decimal floating point notation to create a “hybrid” binary floating point number:
  1. We first use a “binary point” to separate whole numbers from fractional numbers to make a fixed point notation:
     - e.g. 00011001.110
  2. We then “float” the binary point:
     - 00011001.110 => 1.1001110×2^4 (mantissa = 1.1001110, exponent_value = 4)
  3. Now we have to express this without the extra symbols (x, 2, .)
     - by convention, we divide the available bits into three fields: sign, mantissa, exponent

Normalized Floating Point Numbers

- For a normalized floating point number (S, E, F)



- Significand is equal to $(1.F)_2 = (1.f_1f_2f_3f_4\ldots)_2$
  - IEEE 754 assumes hidden 1. (not stored) for normalized numbers
  - Significand is 1 bit longer than fraction
- Value of a Normalized Floating Point Number:

$$\pm (1.F)_2 \times 2^{exponent\_value}$$

$$\pm (1.f_1f_2f_3f_4\ldots)_2 \times 2^{exponent\_value}$$

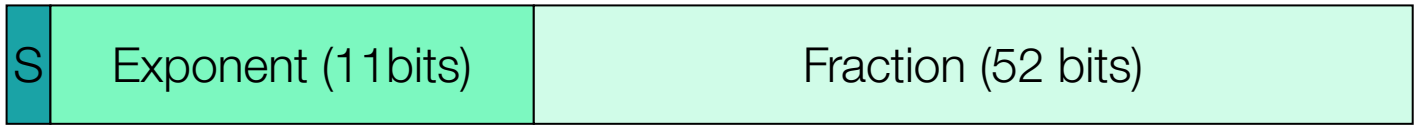
$$\pm (1 + f_1\times 2^{-1} + f_2\times 2^{-2} + f_3\times 2^{-3} + f_4\times 2^{-4}\ \ldots)_2 \times 2^{exponent\_value}$$

  - S = 0 is positive, S = 1 is negative

IEEE 754 Floating-Point Standard

- Found in virtually every computer invented since 1980
- Single Precision Floating Point Numbers (32 bits)
  - 1-bit sign + 8-bit exponent + 23-bit fraction

- Double Precision Floating Point Numbers (64 bits)
  - 1-bit sign + 11-bit exponent + 52-bit fraction



Exponent Representation

- How to represent a signed exponent?
  - Sign + magnitude representation for the exponent,
  - Two's complement representation,
  - Biased representation
- IEEE 754 uses biased representation for the exponent
  - Exponent Value = E– Bias (Bias is a constant)
- Value of a Normalized Floating Point Number is

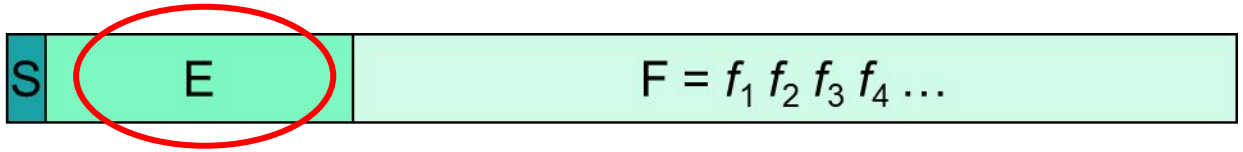
$$\pm (1.F)_2 \times 2^{(E\text{-}Bias)}$$

$$\pm (1.f_1f_2f_3f_4\ldots)_2 \times 2^{(E\text{-}Bias)}$$

$$\pm (1 + f_1\times 2^{-1} + f_2\times 2^{-2} + f_3\times 2^{-3} + f_4\times 2^{-4}\ldots)_2 \times 2^{(E\text{-}Bias)}$$

S = 0 is positive, S = 1 is negative

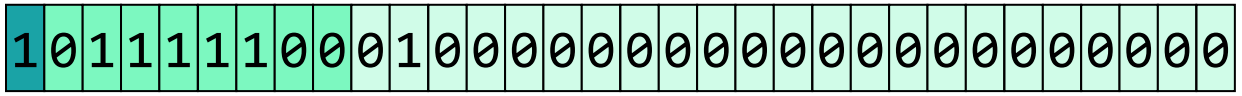
Biased Exponent



- Single precision
  - The exponent field is 8 bits
  - E = 0 and E = 255 are reserved for special use
  - E = 1 to 254 are used for normalized floating point numbers
  - Bias = 127 (half of 254)
  - Exponent value = E – 127 Range: -126 to +127
- Double precision
  - The exponent field is 11 bits
  - E = 0 and E = 2047 are reserved for special use
  - E = 1 to 2046 are used for normalized floating point numbers
  - Bias = 1023 (half of 2046)
  - Exponent value = E – 1023 Range: -1022 to +1023

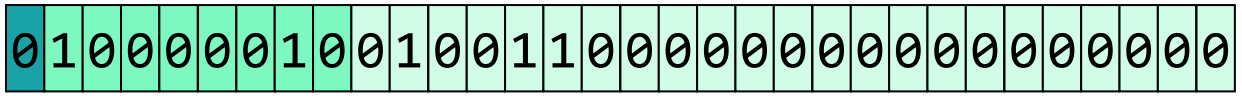
Examples of Single Precision Float

- What is the decimal value of this Single Precision float?



- Solution:
  - Sign = 1 is negative
  - E = $(01111100)_2$ = 124, E – bias = 124 – 127 = –3
  - Significand = $(1.0100 \ldots 0)_2 = 1 + 2^{-2} = 1.25$ (1. is implicit)
  - Value in decimal = $-1.25 \times 2^{-3} = -0.15625$

- What is the decimal value of?



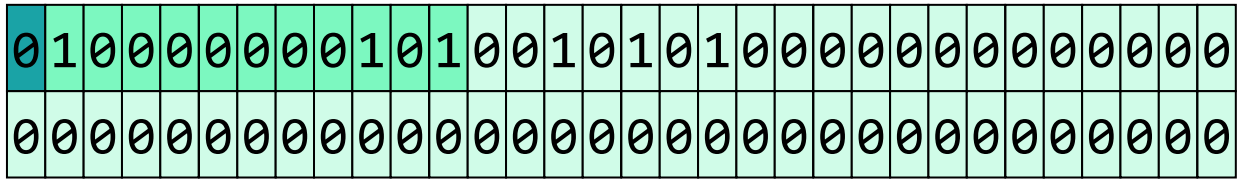
- Solution:

implicit

  - Value in decimal = $+(1.01001100 \ldots 0)_2 \times 2^{130-127} =$ $(1.01001100 \ldots 0)_2 \times 2^3 = (1010.01100 \ldots 0)_2 =$ 10.375

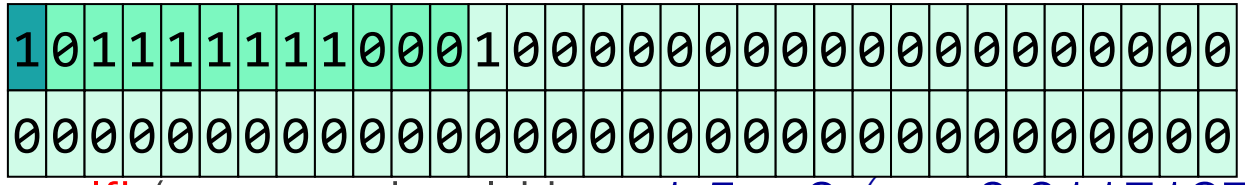
Examples of Double Precision Float

- What is the decimal value of this Double Precision float ?



- Solution:
  - Value of exponent = $(10000000101)_2$ – Bias = 1029 – 1023 = 6
  - Value of double = $(1.00101010 \ldots 0)_2 \times 2^6$ (1. is implicit) = $(1001010.10 \ldots 0)_2$ = 74.5

- What is the decimal value of ?



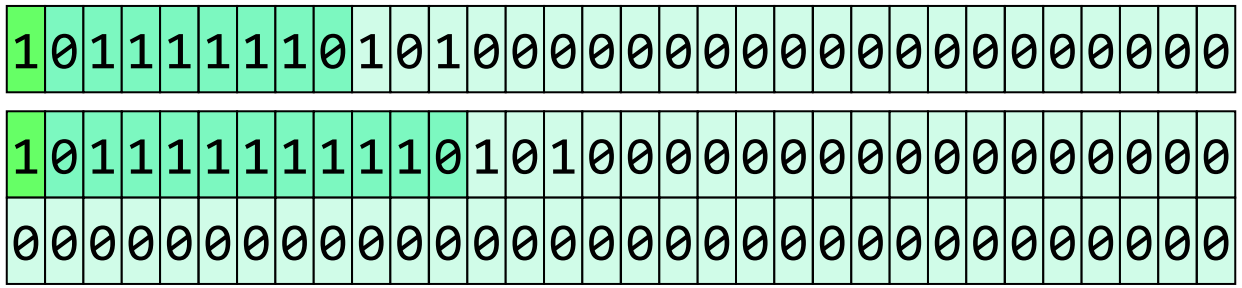
- Do it yourself! (answer should be $-1.5 \times 2^{-7} = -0.01171875$)

Decimal to Binary Floating-Point

- Convert - 0.8125 to single and double-precision floating-point
- Solution:
  - Fraction bits can be obtained using multiplication by 2
    - $0.8125 \times 2 = 1.625$
    - $0.625 \times 2 = 1.25$
    - $0.25 \times 2 = 0.5$
    - $0.5 \times 2 = 1.0$
    - Stop when fractional part is 0, or after computing all required fraction bits

$$0.8125 = (0.1101)_2$$

  - Fraction = $(0.1101)_2 = (1.101)_2 \times 2^{-1}$ (Normalized)
  - E = –1 + Bias = 126 (single precision) and 1022 (double)



Single Precision

Double Precision

Summary of IEEE 754 Encoding

Single-Precision	Exponent = 8	Fraction = 23	Value
Normalized Number	1 to 254	Anything	$\pm (1.F)_2 \times 2^{E-127}$
Denormalized Number	0	nonzero	$\pm (0.F)_2 \times 2^{-126}$
Zero	0	0	± 0
Infinity	255	0	$\pm \infty$
NaN (not a Number)	255	nonzero	NaN

Double-Precision	Exponent = 11	Fraction = 52	Value
Normalized Number	1 to 2046	Anything	$\pm (1.F)_2 \times 2^{E-1023}$
Denormalized Number	0	nonzero	$\pm (0.F)_2 \times 2^{-1022}$
Zero	0	0	± 0
Infinity	2047	0	$\pm \infty$
NaN	2047	nonzero	NaN

Other codes

Binary Coded Decimal (BCD)

- In the Binary Coded Decimal (BCD) representation each decimal digit is converted to its 4-bit pure binary equivalent
- This code is the simplest, most intuitive binary code for decimal digits and uses the same powers of 2 as a binary number,
  - but only encodes the first ten values from 0 to 9.
    - For example: $(57)_{dec} \rightarrow (?)_{bcd}$

$$(\ 5\ \ \ 7\ )_{dec} = (0101\ \ 0111)_{bcd}$$

Binary Coded Decimal (BCD)

- You can think of BCD in terms of column weights in groups of four bits. For an 8-bit BCD number, the column weights are:

80 40 20 10 8 4 2 1

- What are the column weights for the BCD number : 1000001101011001?

1000 0011 0101 1001

8000 4000 2000 1000 800 400 200 100 80 40 20 10 8 4 2 1

- Note that you could add the column weights where there is a 1 to obtain the decimal number. For this case:

$1000001101011001 = 8000 + 200 + 100 + 40 + 10 + 8 + 1 = 8359_{10}$

BCD addition

- Add the following BCD numbers :
  - 0011 + 0100
  - 010001010000 + 010000010111
  - 1001 + 0100

- Solution
  - 0011+0100=0111
  - 010001010000+010000010111=100 0110 0111
  - 1001+0100=0001 0011 (Add 6 to the invalid BCD number which is >9)

Gray code

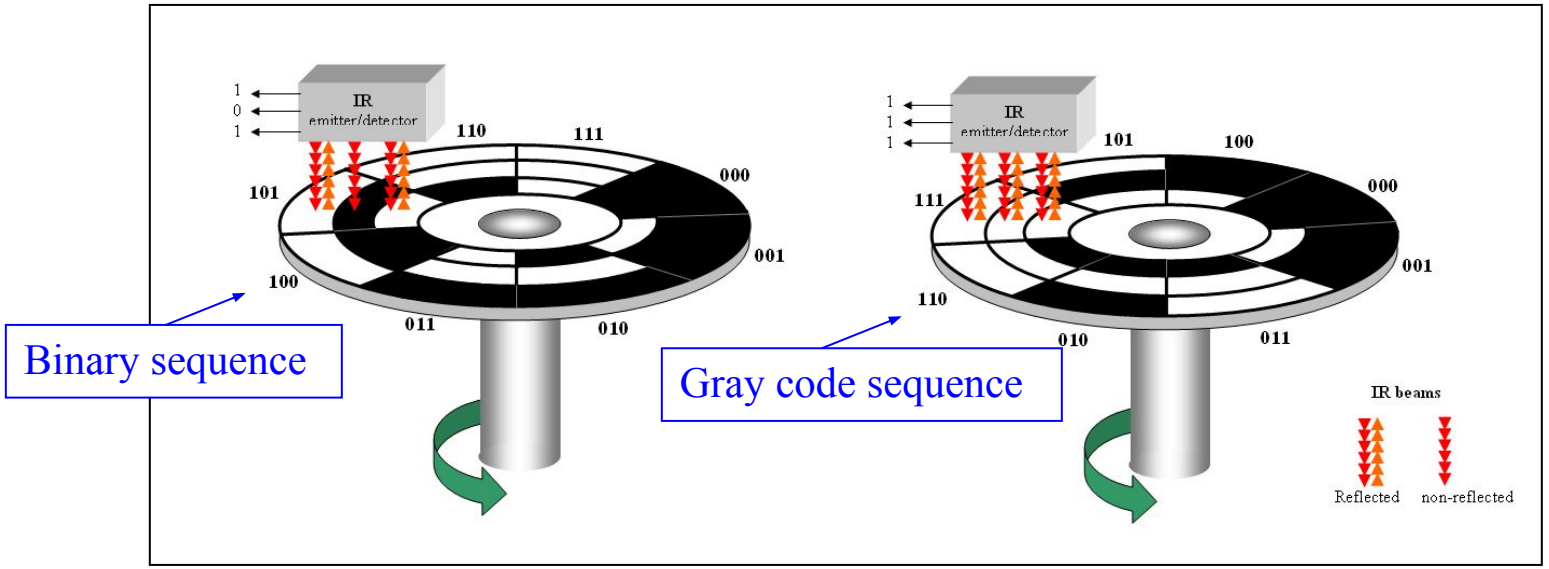
- The Gray code is used in applications where numbers change rapidly.
  - Only one bit changes from each value to the next.
  - This allows to avoid problems in systems where an error can occur if more than one bit changes at a time

Three bit binary and Gray code equivalents.

B_2	B_1	B_0	G_2	G_1	G_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

Gray code : Example of application: A shaft encoder

- Three IR emitter/detectors are used to encode the position of the shaft.
- The encoder on the left uses binary and can have three bits change together, creating a potential error.
- The encoder on the right uses gray code and only 1-bit changes, eliminating potential errors.



Decimal-Binary-Hex-BCD-Gray

Decimal	Binary	Hexadecimal	BCD	GRAY
0	0	0	0000	0000
1	1	1	0001	0001
2	10	2	0010	0011
3	11	3	0011	0010
4	100	4	0100	0110
5	101	5	0101	0111
6	110	6	0110	0101
7	111	7	0111	0100
8	1000	8	1000	1100
9	1001	9	1001	1101
10	1010	A	0001 0000	1111
11	1011	B	0001 0001	1110
12	1100	C	0001 0010	1010
13	1101	D	0001 0011	1011
14	1110	E	0001 0100	1001
15	1111	F	0001 0101	1000

ASCII (American Standard Code for Information Interchange)

- The ASCII code represents characters and functions on a computer keyboard
  - 26 lowercase
  - 26 uppercase letters
  - 10 digits
  - 7 punctuation marks
  - 20 to 40 other characters.
- Seven bit code: $2^7 = 128$ possible codes
- use: transfer information between computers; computers & printers;
- Extension: In 1981, IBM introduced extended ASCII, which is an 8-bit code and increased the character set to 256

ASCII code

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	Y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~

ASCII code

128	Ç	144	É	160	á	176	▒	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

UNICODE

- UNICODE extends ASCII to 65,536 universal characters codes
  - For encoding characters in world languages
  - Available in many modern applications
  - 2 byte (16-bit) code words

Conversion or Coding?

- Do NOT mix up "conversion of a decimal number to a binary number" with "coding a decimal number with a binary code".
- $13_{10} = 1101_2$
  - This is conversion
- 13 ⇔ $0001\ 0011_{BCD}$
  - This is coding