Bits, bytes, and number base system

Base 10

If you have house address plates that have space to write only two digits (where a digit is from 0 to 9), how many different houses can you address?

So, the address plates look like the following:

___ ___

Base 10

- Each of those two spaces can hold any digit from 0 to 9.

- [0 thru 9]   [0 thru 9]

- This means the first space can have 10 different digits, and the second space can also have 10 different digits.
- So, how many possible combinations is that ….   It would be 10 X 10 = 100

- Meaning we can represent 100 different values with two spaces…. From 0 to 99

- If we had three spaces, we would be able to represent 1000 values, from 0 to 999

Base 10

- In other words, the number of values we can represent is:

- 10^n

- When n is 2, we get $10^2 = 100$
- When n is 3, we get $10^3 = 1000$
- When n is 4, we get $10^4 = 10000$ (from 0 to 9999)

Base 2

- In base 2, the only digits we have are 0 and 1

  - In general, in base n, you will have digits from 0 to n-1
  - For example, in decimal system (base 10), we have digits from 0 to 9
  - And in hexadecimal system (base 16), digits are from 0 to 15 (represented as F)
  - In octal (base 8), digits are from 0 to 7.

- So, in binary system, if we have two spaces for digits, how many values can we represent

- ___ ___

Base 2

- [0 or 1]   [0 or 1]

- This means the first space can have 2 different digits, and the second space can also have 2 different digits.
- So, how many possible combinations is that ….   It would be 2 X 2 = 4

- Meaning we can represent 4 different values with two spaces…. From 00 to 11 in binary, which is from 0 to 3 in decimal

- With 3 spaces, we can represent 8 values, from 000 to 111 binary, which is from 0 to 7 in decimal

- With 10 spaces, we can represent 1024 values, from 00 to 1111111111 in binary, which is from 0 to 1023 in decimal

Base 2

- So, in general, with n digits, we can represent 2^n values.
  - And the range of those values is from 0 to $2^n - 1$
  - Note, we are talking only about unsigned integers

- If the size of int data type is 32 bits:
  - Number of values is 2^{32}
  - Range of values is from 0 to $2^{32} – 1$
  - 2^{32} is a little over 4 billion.

- If the size of int data type is 64 bits:
  - Number of values is 2^{64}
  - Range of values (unsigned) is from 0 to $2^{64} – 1$
  - $2^{64} = 2^{32} * 2^{32}$ OR approximately 4 billion X 4 billion.
  - Also written as: 0xFFFFFFFFFFFFFFFF in hexadecimal (base 16)
  - where F is 15 in base 10, or 1111 in base 2.
  - A = 10, B = 11, C = 12, D = 13, E = 14, F =15 (hexadecimal goes from 0 to 15)

Base 2

- Note some interesting facts

- **Multiplying a number by 2** is the same as shifting its binary representation **left by 1**
- E.g.:
  - 4 x 2 = 8
  - In binary: 4 is 0100 : shift left by one position, u get : binary 1000, which is 8 decimal

  - Shifting left by 2 positions will multiply by 2^2, so u get: binary 10000, which is 16 decimal

  - This works in decimal system as well.
    - Shift 77 to left by one position, u get: 770 (multiplied by 10)
    - Shift 77 to left by two positions, u get: 7700 (multiplied by 100, or 10^2)

Base 2

- **Dividing a number by 2** is the same as shifting its binary representation **right by 1**
- E.g.:
  - 4 / 2 = 2
  - In binary: 0100 : shift right by one position, u get : 010, which is 2

  - Shifting right by 2 positions will divide by 2^2, so u get: 01, which is 1

  - This works in decimal system as well.
    - Shift 777 to right by one position, u get: 77 (integer division by 10)
    - Shift 777 to right by two positions, u get: 7 (integer division by 100, or 10^2)

- So we looked at decimal as well as binary number systems.
- Computers only understand the binary number system.

- Why do you think computers were made to understand binary, but not the decimal system… after all, most people are familiar with decimal system and not binary?

Bits and bytes

Bit is the smallest value that can be represented.

One bit can hold a value of either 1 or 0. That means one bit can hold 2 values.

Byte is 8 bits.

1000 bytes = 1KB

1000 KB = 1MB

1000 MB = 1GB (or 10^9 bytes)

1000 GB = 1TB (or 10^{12} bytes)

1000 TB = 1 PB (peta) (or 10^{15} bytes)

1000 PB = 1 EB (exa byte) (or 10^{18} bytes)

1000 EB = 1 ZB (zetta byte) (or 10^{21} bytes)

1000 ZB = 1 YB (yotta bytes) (or 10^{24} bytes)

1000 YB = 1 Xenottabyte (XB ?) (or 10^{27} bytes)

Shilentnobyte, then Domegemegrottebyte, then Icosebyte and then Monoicosebyte

https://en.wikipedia.org/wiki/Talk%3AYottabyte#Xenottabyte?_Shilentnobyte?_Domegemegrottebyte?

LAB

- Write a function that takes an integer value and returns its binary representation as a string.
- So, something like:
  - ConvertToBinary(4) would return "100" (or "0000000100")
  - ConvertToBinary(10) would return "0001010"
  - ConvertToBinary(15) would return "01111" ("0000001111")

- 00000000000000000000000000010101 is binary for decimal 21
- 00000000000000000000000000010100 is binary for decimal 20
- Note that in binary representation:
  - Rightmost bit is also known as the Least Significant Bit, or LSB.
  - Leftmost bit is also known as the Most Significant Bit, or MSB.
  - LSB == 0 implies the number is even,
  - LSB == 1 implies the number is odd

  - So, my evenness check could be
    - boolean IsEven = (n & 0x1) == 0;
    - I am "*bit anding*" the number *n* with 0x1 (which is hexadecimal 1, and same as decimal 1).
    - So, 0x1 is a bit mask with LSB as 1, and rest all 0s. (Since 1 is 0000…1 i.e., 31 0s followed by 1)
  - BTW, I could use this bit masking check along with *bit shifting* to figure out the binary representation of a number.

LAB

- Write a function Convert:

  - takes an integer as 1st argument – this represents the value to be converted
  - takes an integer as 2nd argument – this represents the base to be used.
  - Returns a string that has the converted value.

  - Examples:
    - Convert (3, 2) returns “11” // of “0000011”
    - Convert (7, 2) returns “111”
    - Convert (32, 16) returns “20”

URL shortening

- The number base conversion is used in url shortening … these are the shortened urls that you may have seen.
- Example: http://bit.ly/alg-ds (just made that up) or something like that
- Benefit is that these urls are easy to type or tell someone over the phone (or remember), rather than trying to type really long ones and getting them wrong because you missed one character.

- Lets look at the number base theory behind this.
- For representing the shortened urls, we can use any of the 26 letters from A to Z.

- Taking both upper and lower case gives us 52 letters.
  - A B C D E F …W X Y Z a b c d e f … w x y z

- Using digits from 0 to 9 gives us a total of 62.
  - 0 1 2 3 4 5 6 7 8 9 A B C D E F …W X Y Z a b c d e f … w x y z

Base 62

- So, we have a total of 62 letters / symbols we can use.

- So, if our shortened url were to be only of length 1, how many different values could it represent?
- What about with length of 2?

- Now, lets say our shortened urls are going to be 7 characters long.
- Each space can be any of ‘A’ to ‘Z’, or ‘a’ to ‘z’ or ‘0’ to ‘9’.
- So, we have a total of 62 x 62 x 62 x 62 x 62 x 62 x 62 = 62^7 possible combinations !!!
- That’s more than 3.5 trillion urls that we can represent using shortened urls of length 7.

- If we use 8 spaces, this becomes more than 200 trillion (3.5 trillion X 62)

Base 62

- Lets look at a simple example:

- If the shortened url is:
  - aAx97
  - What value does this represent in base 10 (decimal system)?

Base 62

- We have: aAx97

a	A	x	9	7	
___	___	___	___	___	
62^4	62^3	62^2	62^1	62^0	← Place values

a = 36 x 62^4 = 531948096

A = 10 x 62^3 = 2383280

x = 59 x 62^2 = 226796

9 = 9 x 62 = 558

7 = 7 x 1 = 7

0 thru 9. ‘A’ thru ‘Z’ ‘a’ thru ‘b’

Sum of these: 534,558,737

Base 62

- What is the largest number we can represent in base 62 if it can be 5 spaces long?

Base 62

Base 62: zzzzz

Decimal value: 916,132, 831

Largest number we can represent in base 62 if it can be 7 spaces long:

Base 62: zzzzzzz

Decimal value: 2,020,598,544,545 (more than 2 trillion)