Positional Number Systems

Data in the computer's memory is represented using units that can each be in one of two states (0 or 1) This is known as binary code and can be thought of as a switch, 0 = off and 1 = on

All the data contained in a computer system is thus represented in binary, meaning text, images, video, etc. are all transformed into binary numbers

> We count by tens, but computers count by twos, so data is being converted between different number systems (Base 10 ↔ Base 2)

To understand number systems, you have to understand that digits like 10 or 13 are just symbols that we use to represent a number of objects. These symbols can mean different things in different systems depending on how that system groups objects together.

Counting in Base 10

Let's start with the most common number system, base 10 or the decimal number system, which is what we naturally use to count.

In base 10 there are ten digits we use to represent the quantity of objects

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

After 9, we start to group objects together.

Numbers are written in a positional notation.

$$\underset{\uparrow}{2}\ \underset{\uparrow}{3}\ \underset{\uparrow}{0} \cdot \underset{\uparrow}{0}\ \underset{\uparrow}{0}\ \underset{\uparrow}{0}$$

100s 10s 1s 1s 10s 100s

So writing 230 is like saying two groups of 100, three groups of 10, and zero additional ones. 100 is just ten groups of 10 and zero additional tens and ones. 10 is one group of ten and zero additional ones hence $\underset{\uparrow}{1}\ \underset{\uparrow}{0}$ (10s 1s)

Thus, 0-9 are used to represent all the different values you can have in each position.

Counting in Base 5

We are grouping five objects together so the digits we use are 0, 1, 2, 3, 4 only, so 5 in base 10 would be represented as 10 in base 5.

Base 10	Base 5
10	20
35	120
243	1433

Counting in Base 2

Since our computer system uses binary, meaning it only uses 0s and 1s, it is counting in Base 2.

Digits: 0, 1

Let's count in Binary:

0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001 ...

We are grouping much more often, so we get "larger" numbers much faster.

Counting in Base 16

Base 16 or the hexadecimal number system is also important to computer systems.

Digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f

Since we don't begin to group until sixteen objects, we borrow symbols from the alphabet to represent the other numbers leading up to sixteen.

Let's count:

10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1a, 1b, 1c, 1d, 1e, 1f, 20 ...

What comes before 100? ff which represents 15 tens and 15 ones

Equivalent Representations

$(13)_{10} = (15)_8 = (23)_5 = (1101)_2 = (D)_{16}$

Representation of number N in base b_1 → Representation of number N in base b_2

Converting Between Bases

There are two methods of converting between bases.

1) N in base b to N in decimal

2) N in decimal to N in base b

These methods use decimal or base 10 as a stepping stone for conversions.

N in Base b to N in Decimal

$$(a_n \ldots a_2\, a_1\, a_0)_b = a_0 \cdot b^0 + a_1 \cdot b^1 + a_2 \cdot b^2 + \ldots + a_n \cdot b^n$$

We should think of numbers as a weighted sum of positional digits

$$(1\,2\,5)_8 = (5 \cdot 8^0) + (2 \cdot 8^1) + (1 \cdot 8^2) = (85)_{10}$$

$1 \to 8^2 = 64$, $2 \to 8^1 = 8$, $5 \to 8^0 = 1$

$$(1\,0\,1\,1)_2 = (1 \cdot 2^0) + (1 \cdot 2^1) + (0 \cdot 2^2) + (1 \cdot 2^3) = (11)_{10}$$

$1 \to 2^3 = 8$, $0 \to 2^2 = 4$, $1 \to 2^1 = 2$, $1 \to 2^0 = 1$

N in Decimal to N in Base b

Geometrical Progression:

$$1+2+4+8+\ldots+2^K = 2^{K+1}-1$$

$$(75)_{10} = \underset{2^8}{0}\ \underset{2^7}{0}\ \underset{2^6}{1}\ \underset{2^5}{0}\ \underset{2^4}{0}\ \underset{2^3}{1}\ \underset{2^2}{0}\ \underset{2^1}{1}\ \underset{2^0}{1}$$

256 128 64 32 16 8 4 2 1

these are automatically zero because they're > 75

$= 75$

thus the digits in these positions must sum to 75, but how do we know where the 0s and 1s are?

To ensure that we aren't placing digits in positions arbitrarily, we start with the largest position that's less than the number N. In this case it's 2^6/64

$$\begin{array}{r} 75 \\ -64 \\ \hline 11 \end{array}$$

we used 2^6, so it's a 1, there is a remainder or additional objects to deal with

$$\begin{array}{r} 11 \\ -8 \\ \hline 3 \end{array}$$

Here we skip over 2^5 and 2^4 because they're > 11, so we label them 0s and use 2^3 which gets noted as 1

$$\begin{array}{r} 3 \\ -2 \\ \hline 1 \end{array}$$

2^2 is too large, so we use 2^1 and label the positions accordingly

$$\begin{array}{r} 1 \\ -1 \\ \hline 0 \end{array}$$

We're done! there are no remainders so all objects are accounted for

Thus, $(75)_{10} = (1001011)_2$ and we understand that the ones note positions that sum to the original number

The key to this method is to keep cutting down the remainder and note the positions that hold ones

Binary to Hexadecimal Conversion

This conversion is unique because we treat digits independently, meaning we translate them individually and not in relation to each other.

To go from binary to hex, we go right to left and group the digits into groups of four. Then, we can use a hex to 4 bit binary chart to translate the groups.

$$(\underbrace{0011}_{3}\ \underbrace{1011}_{b}\ \underbrace{1001}_{9})_2 = (3b9)_{16}$$

Let's breakdown why this works.

$$(0011\,1011\,1001)_2 = 1\cdot2^0+1\cdot2^1+0\cdot2^2+0\cdot2^3+1\cdot2^4+0\cdot2^5+1\cdot2^6+ 1\cdot2^7+0\cdot2^8+1\cdot2^9+1\cdot2^{10}+0\cdot2^{11}$$

If we take this expression and break it into groups of four we can understand the relationship to the 4 bit binary chart.

$$\underbrace{0\cdot2^{11}+1\cdot2^{10}+1\cdot2^9+0\cdot2^8}+\underbrace{1\cdot2^7+1\cdot2^6+0\cdot2^5+1\cdot2^4}+\underbrace{0\cdot2^3+0\cdot2^2+1\cdot2^1+1\cdot2^0}$$

* factor these * groups

$$\underbrace{0\cdot2^0+1\cdot2^1+1\cdot2^2+0\cdot2^3}+\underbrace{1\cdot2^0+0\cdot2^1+1\cdot2^2+1\cdot2^3}$$

2^8 → $6\cdot2^8$ → $2^8 = 64 = (6)\cdot16^2$

2^4 → $13\cdot2^4$ → $2^4 = 16 = (13)\cdot16^1$

2^0 → $3\cdot2^0$

Addition

Base 10

$$\begin{array}{r} {\scriptstyle 1\ \ \ \ } \\ 325 \\ +692 \\ \hline 1017 \end{array}$$

Base 8

$$\begin{array}{r} {\scriptstyle 1\ 1\ \ \ } \\ 365 \\ +243 \\ \hline 630 \end{array}$$

Base 2

$$\begin{array}{r} {\scriptstyle 1\ \ \ 1\ \ \ \ \ \ \ \ } \\ 10011100 \\ +11011001 \\ \hline 101110101 \end{array}$$

Subtraction

Base 10

$$\begin{array}{r} {\scriptstyle 12\ \ \ \ } \\ 3427 \\ -192 \\ \hline 235 \end{array}$$

Base 8

$$\begin{array}{r} {\scriptstyle 13\ \ \ \ } \\ 4536 \\ -351 \\ \hline 165 \end{array}$$

Base 5

$$\begin{array}{r} {\scriptstyle 13\ \ \ \ } \\ 3432 \\ -141 \\ \hline 241 \end{array}$$

Signed Numbers

Approaches to representing signed numbers using only 0s and 1s:

Sign and Magnitude

The first digit represents the sign and the trailing digits or magnitude represent the number

$$-2 = \underbrace{1}_{\text{sign}}\underbrace{000\ldots011010}_{\text{magnitude}}$$

Two's Complement

This is the most common representation method. In a k-bit two's complement representation of a number:

A positive integer is represented in its $(k-1)$ bit unsigned binary representation, padded with a 0 to its left

The sum of a number and its additive inverse must equal 2^k

8-bit two's complement examples:

$$(26)_{10} = (\underbrace{0}_{\text{Padding}}\underbrace{0011010}_{\text{number}})_2$$

$(-26)_{10} = (1100110)_2$ because

$00011010 + x$ must equal 100000000,

thus $x = 1100110 = (-26)_{10}$

$$\begin{array}{r} {\scriptstyle 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1} \\ 00011010 \\ +\ 11100110 \\ \hline 100000000 \end{array}$$