Number Systems

In this document, as well as attached hand written notes, I will be discussing number systems. This will entail what numbers sytems are, the different types of number systems, uses of number systems and operations of number systems.

There are different number systems used in lots of different areas, but in IT we focus mainly on the use of 4 main number systems. These being hexadecimal, binary, octal and binary. Throughout this document I'll be presenting uses for these numbers sytems, but first I'll explain how they work and then demonstrate conversion between them.

Binary Numbers is a number system built from only two digits. These are 1's and 0's. Also known as true or false values. This is the most basic number system and is used at the most basic levels of computing such as physical hardware and chipsets. Binary is also used in everyday situations such as electronic gates and electricity circuits.

Denary Numbers are the most commonly used number system as it is used by us in everyday life. They are used in standard accounting, finances and measurement systems. The benefits to the denary number system is that it is much easier to understand compared to the other number systems as it is taught from birth rather than the other systems which would only ever be learned by those specifically interested in IT. It is also thought to have come about due to humans having 10 fingers and at one point depending on them to count.

Octal numbers are the least common of the 4 number systems described in this document. They are mainly used in computing graphics, text and UNIX, which uses octal numbers for it's file protection system (I'll cover this more later on). Octal numbers are formed of 8 different units, 0 through 7. Once 0 through 7 has been counted, the system restarts with a new value in the tens and will then work up to 77 where it will continue with 100 and 101. Octal numbers can often be mistaken for denary numbers as there's no particular distinction in the way they're written. To be clear users of different systems will note what base the number system uses. You'll see what base octal is on the table below.

Hexadecimal numbers are used most commonly in computing to represent memory locations. Memory locations are found via memory addresses, these are a data concept used at many levels of software and hardware to access a computer's primary storage memory. Memory addresses are composed of a fixed length sequence of digits displayed as unsigned (non-negative) integers. Hexadecimal numbers are one of the more difficult number systems to read and follow because they incorporate letters after 10 numerals which is unlike the other four number systems we've discussed.

This is a conversion table, it shows some basic conversions between the four number systems. We are all familiar with the denary number system, the rest of which have to be figured out by applying conversion methods.

Denary (Base 10)	Binary (Base 2)	Octal (Base 8)	Hexadecimal (Base 16)
0	00000000	00	0
1	00000001	01	1
2	00000010	02	2
3	00000011	03	3
4	00000100	04	4
5	00000101	05	5
6	00000110	06	6
7	00000111	07	7
8	00001000	10	8
9	00001001	11	9
10	00001010	12	A
11	00001011	13	B
12	00001100	14	C
13	00001101	15	D
14	00001110	16	E
15	00001111	17	F

Applications of number systems vary between them. The most used of the four is without doubt the denary system, this is used in things like currency, mathematics and measurements. Other number systems are also rather common, like sexagesimal which is used for timekeeping in hours, or duodecimal which was once as common as denary, used for measuring hours in a day, feet and inches, pound and ounce. These applications are well known by almost everyone, below I will be detailing the more technical applications of number systems used by IT practitioners.

ASCII (American Standard Code for Information Interchange) is a character encoding standard. This is a standardised set of binary values assigned to english characters on a keyboard. Essentially meaning that each key selected by a user's finger inputs a string of binary which then outputs the english letter printed on the key, on the screen. For example the phrase “Hello” with an uppercase ‘H’ and lowercase ‘ello’ would translate to “01001000 01100101 01101100 01101100 01101111”. They can also be translated into denary numbers, which would like like, “72 101 108 108 111”.
The ASCII character set ranges from 0 to 127 with upper and lower case letters, numerals and special characters. Some computers will extend the range to the full 256 characters

available in a byte of data, this is done to allow for use of common foreign accented characters. Below is an ASCII character table for a visual look at the encoding standard.

ASCII Alphabet			
A	1000001	N	1001110
B	1000010	O	1001111
C	1000011	P	1010000
D	1000100	Q	1010001
E	1000101	R	1010010
F	1000110	S	1010011
G	1000111	T	1010100
H	1001000	U	1010101
I	1001001	V	1010110
J	1001010	W	1010111
K	1001011	X	1011000
L	1001100	Y	1011001
M	1001101	Z	1011010

As previously stated, octal number systems are employed for file protection in UNIX. Every file and folder in UNIX has what's called an access permission. Meaning what a user is able to do with the file or folder. The three levels of permissions are; read access, the user may only read the contents of the file or folder; write access, the user may both read and edit the contents of the file or folder; execute access, the user may read, edit and execute the file, meaning it could be a VBA or shell script that had the potential to be executed.

There are three types of users as far as UNIX permissions are concerned. The owner, the group the owner belongs to and other users. Meaning that in UNIX protection, there are 3 levels of permissions, and 3 types of users, allowing for 9 different variations. The means then that UNIX file permissions are nine bits of information, some might only contain one or two values; grant or deny access.

This specifies for each file who can read or write from/to the file. In the case of scripts or executable files, it can also set whether or not they may be executed.

The creation of these characters are formed using the following rules.The first of 10 characters (0) is represented by either a 'd' if the item is a directory (folder), 'l' if it is a link or '-' if the item is a file. Characters in positions 1, 2 and 3 ('rwx') represent permissions for the owner of the file. 4, 5 and 6 ('r--') represent permissions for the group. 7, 8 and 9 ('r--') represent permissions for other users. The character 'r' means read access, 'w' is write access and 'x' is execute access, whilst '-' indicates access type denied in lieu of r,w or x.

So far there doesn't appear to be any usage of octal, but this is where is comes into UNIX. The '-rw-rw-rw-' we've discussed is called the symbolic notation. Octal is what we need this notation in, numeric notation. The example I just gave is represented in octal as 0666. Below is a table of other examples of symbolic and octal UNIX notations.

Symbolic Notation	Numeric Notation	English
----------	0000	No permissions.
-rwx------	0700	Read, write & execute only for owner.
-rwxrwx---	0770	Read, write & execute for owner and group.
-rwxrwxrwx	0777	Read, write & execute for owner, group and others.
---x--x--x	0111	Execute.
--w--w--w-	0222	Write.
--wx-wx-wx	0333	Write & execute.
-r--r--r--	0444	Read.
-r-xr-xr-x	0555	Read & execute.
-rw-rw-rw-	0666	Read & write.
-rwxr-----	0740	User may read, write & execute; group can only read; others have no permissions.

Hexadecimal is a slightly trickier number system which, as previously mentioned, can be used for addressing memory. Memory addresses are set out as two hex numbers. For example, D900:3 could be a memory address. The left hex is called the segment address, the right is called the offset. The real address is found by adding a zero to the segment address and then pulsing the offset. So in the prior example, the actual address would be D9003, this is called the linear address of the memory.

The splitting of the segment and offset by a colon was introduced at the time when the largest register in a CPU was 16-bit, only able to process 65,536 bytes of memory directly. This decision was made in an attempt to create more ways for the 16-bit registers to access more memory, by expanding the instruction set, more programs could command the CPU to group 16-bit registers together when they needed to refer to a linear address of a location beyond 64KB. The alternative would've been to produce CPU's with larger register sizes, but would've increased the cost of the CPU's.

IP Addressing (of V4) is formed of binary, 32 bits in total, split into 4 octets of 8 bits each. 109.156.16.51 is my public IPv4 address, this would therefore be stored as 01101101 10011100 00010000 00110011. This is fairly easy to understand and involves the simple conversion of denary to binary.

IP Addressing (of V6) is first formed of binary, 128 bits in total, split into 16 bit boundaries. My public IPv6 address is “8193:0:24309:31227:13538:15895:37475:61388” in denary. In binary it would become “0010000000000001 0000000000000000 0101111011110101 0111100111111011 0011010011100010 0011111000010111 1001001001100011 1110111111001100”. Finally to produce an IPv6 address most would be more familiar with, it is translated then into hexadecimal which would look like this, “2001:0:5ef5:79fb:34e2:3e17:9263:efcc”. You might notice that this looks rather short, that’s because you can further simplify IPv6 addresses by removing leading zeros within each block, besides the final digit as all blocks must contain at least one digit.

Binary is also linked to IP Addressing with classes of IP addresses. IP addresses have classes to determine which part belongs to the network address and which part belongs to the node address. All nodes on any single network share identical network prefixes but must have a unique host number.

In a Class A Network, binary addresses start with 0, leaving the decimal range between 1 and 126. The first octet identifies the network whilst the remaining 24 bits indicate the host within a network. A Class A IP address could look like 109.156.16.51 where 109 identifies the network and 156.16.51 indentifies the host on the network.

In Class B Networks, binary addresses start with 10, so the decimal range is from 128 to 191. (127 is reserved for a process called loopback and is used solely for internal testing on the local machine.) The first two octets of a Class B IP address identify the network and the second two octets identify the host within the network. A Class B IP address could look like 128.222.56.83 where 128.222 identifies the network and 56.83 identifies the host on the network.

In Class C Networks, binary addresses start with 110, so the decimal range must be between 192 and 223. The first three octets if a Class C IP address identify the network and the remaining octet identifies the host within the network. A Class C IP address could look like 200.145.215.155 where 200.145.215 identifies the network and 155 identifies the host on the network.

Classless Inter Domain Routing (CIDR) was invented in 1993 by the Internet Engineering Task Force to replace the previous addressing model of classful network design on the internet. Its purpose was to slow down the rapid exhaustion of IPv4 addresses as the current classful design was very wasteful, some business were given addresses with 16 million host addresses and would use no more than 100. CIDR specifies an IP address range using a combination of IP addresses and network masks.

CIDR notation uses this notation:

“xxx.xxx.xxx.xxx/n”

‘n’ is the number of ‘1’ bits in the mask, such as 192.168.12.0/23, there are 23 ‘1’ bits in this address.
Network Masks might look like this; 255.255.254.0, applied to the prior IP address, represents the address range 192.168.12.0 to 192.168.12.255.