Using Binary

Coding Information

Remember!

Computers do not understand words or decimal numbers and they do not understand natural languages such as English. Every data item and every instruction must be converted into binary digits (bits) to be understood by the machine.

Bit = 0 or 1
Byte = the number of bits used to represent letters, numbers and special characters such as $ # , / &.
Word = the number of bytes a computer can process at one time by the CPU.
A number of coding schemes have been developed to translate characters into a series of bits. Taken together these bits form a byte, so that one character is stored as a single byte in memory.

EBCDIC for Extended **B**inary **C**oded **D**ecimal **I**nterchange **C**ode

ASCII means **A**merican Standard **C**ode for **I**nformation **I**nterchange

EBCDIC (8 bits)	**Character**	**ASCII** (7 bits)
1100 0001	A	100 0001
1100 0010	B	100 0010
1100 0011	C	100 0011
1100 0100	D	100 0100
1100 0101	E	100 0101
1100 0110	F	100 0110
1100 0111	G	100 0111
1100 1000	H	100 1000
1100 1001	I	100 1001
1101 0001	J	100 1010
1101 0010	K	100 1011
1101 0011	L	100 1100
1101 0100	M	100 1101
1101 0101	N	100 1110
1101 0110	O	100 1111
1101 0111	P	**101 0000**
1101 1000	Q	101 0001
1101 1001	R	101 0010
1110 0010	S	101 0011
1110 0011	T	101 0100
1110 0100	U	101 0101
1110 0101	V	101 0110
1110 0110	W	101 0111
1110 0111	X	101 1000
1110 1000	Y	101 1001
1110 1001	Z	101 1010

Using Binary

EBCDIC (8 bits)	**Character**	**ASCII** (7 bits)
1111 0000	0	011 0000
1111 0001	1	011 0001
1111 0010	2	011 0010
1111 0011	3	011 0011
1111 0100	4	011 0100
1111 0101	5	011 0101
1111 0110	6	011 0110
1111 0111	7	011 0111
1111 1000	8	011 1000
1111 1001	9	011 1001

Computer professionals often prefer ASCII because the pattern is easier to understand.

Notice:

Look at the ASCII representation for the numbers 5 through 8:

5 = 011 0101
6 = 011 0110
7 = 011 0111
8 = 011 1000

Just as 5 < 6 < 7 < 8, the same is true for their binary equivalent.

```
     5     <       7
011 0101 < 011 0111

And  7     <       8
011 0111 < 011 1000
```

In the same way, letters also take on an ordering.

```
A =      100 0001
B =                      100 0010        A < B < C
C =      100 0011

X =      101 1000
Y =                      101 1001        X < Y < Z
Z =      101 1010

     A      <       C      <       Z
100 0001 < 100 0011 < 101 1010
```

Because of the mathematical ordering of letters, tasks such as alphabetizing are possible since computers are really comparing numbers rather than letters.

Since both ASCII (and EBCDIC) were based on English over time their value as a world wide coding system has decreased dramatically. At most there are 256 possible different codes in

Using Binary

ASCII based on it's 7-bit makeup. This is barely sufficient to encode English and most European languages.

Unicode

The world of computers is not limited to European languages. In 1980 the Chinese Character Code for Information Interchange (CCCII) was created in Taiwan to code characters from Chinese, Taiwanese and Japanese.

Interesting reading

http://unicode.org/

Unicode provides a single code for every character regardless of the natural language from which it comes. This is critical in using the Web. Unicode can code information in two directions so it can deal with languages like Hebrew and Arabic.

Errors Happen

- When bytes/characters are sent from place to place, transmission errors can happen.
- Computers are only as reliable as the data and information they contain
- Garbage In = Garbage Out (GIGO)

Adding a parity or check bit

Odd Parity

To check for transmission errors, the computer sums the bits in the byte and determines if the number is even or odd.

The computer then adds a 0 or 1 (an extra bit) so that when you examine the entire byte including the parity bit, you have an odd number of 1s. In other words, if the byte has an even number of 1, the parity bit is set to one. If the byte has an odd number of bits, the parity bit is set to zero.

- 0 is an even number and 1 is an odd number.
- In binary adding the digits is the same as counting the number of 1s.
- Odd add a zero(0), even add a one (1) to make the total result odd.

H ⟶ 100 1000 ASCII byte

H with odd parity bit 1 100 1000 ASCII byte

There are two(2) 1-bits in the byte. The byte is even
Add 1 to the byte to make byte odd

Let's look at doing error checking on a telephone number

Using ASCII to translate a phone number 645 3180 into bits (binary digits) would give the following:

Using Binary

6 ➔ 011 0110
4 ➔ 011 0100
5 ➔ 011 0101

3 ➔ 011 0011
1 ➔ 011 0001
8 ➔ 011 1000
0 ➔ 011 0000

Using odd parity, **remember**, the computer counts the **number** of "1"s in the list of bits and decides if the result is odd. If the number of 1-bits is odd, a 0 is placed in the leftmost place keeping the number of 1-bits odd. If the number of 1-bits is even, a 1 is placed in front of the number making the number of 1-bits odd. The receiving computer checks the transmission and then removes (strips off) the extra bit before using the remaining bits as data.

The above binary translation for 645 3180 would look like the following:

6	➔	011 0110	➔	**1** 011 0110
4	➔	011 0100	➔	**0** 011 0100
5	➔	011 0101	➔	**1** 011 0101
3	➔	011 0011	➔	**1** 011 0011
1	➔	011 0001	➔	**0** 011 0001
8	➔	011 1000	➔	**0** 011 1000
0	➔	011 0000	➔	**1** 011 0000

Interesting reading:

http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=97038