Representing Information

Writing Words and Numbers

The First Software?

The Sumerians were ancient inhabitants of the southern part of present-day Iraq. Until about 150 years ago, no one knew that they had ever existed. Unlike the Babylonian civilization that succeeded them in Mesopotamia, they are not mentioned either in the Bible or in the writings of the ancient Greeks. Only after a series of archaeological excavations, beginning in the late nineteenth century, unearthed the remnants of a rich culture, did we come to realize that these people, lost to history for many centuries, created what may have been the world's first great civilization.

Some 5000 years ago the Sumerians had one of the greatest ideas that anyone has ever had. They invented a way to represent the language they spoke with a fixed inventory of graphic symbols. The Sumerians were not the only people to independently come up with the idea of writing, but as far as we know, they were the first.

Sumerian Tablet, ca. 3000 BC

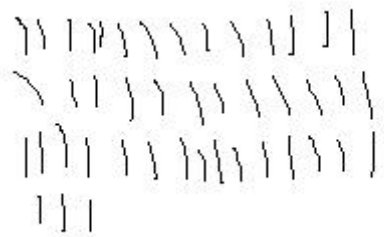
While this course is about computers, part of its purpose is to get you to think about technology and the role it has played in human history. Think now about writing: It is an odd sort of technological advance, if you compare it to other great inventions like the wheel, metallurgy, agriculture, and electric lighting. The Sumerians, who had little in the way of stone or metal, wrote by pressing their characters with a stick into soft clay tablets that were then baked in the sun. But whether writing is pressed into clay, carved on a stone wall, written with pen and ink on paper, traced in puffs of smoke by a skywriting airplane, or made up of glowing dots on a computer screen, anyone who knows the code can decipher the message. The invention is not the physical thing, but the coding system itself.

Writing is not a *hardware* invention—it's *software*. Of course, terms like "software" and "information technology" are used exclusively to refer to something having to do with computers. But schemes for representing and manipulating *information*, inventions that are entirely *logical*, rather than *physical*, have been with us for a very long time.

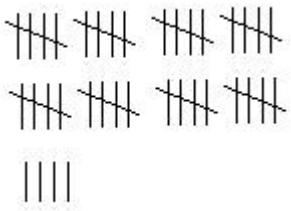
Number Systems

Even if you don't have to come up with the idea of writing all by yourself---let's say that someone tells you, "Here is how we Sumerians remember our kings' names, and how to make beer, and what to buy at the grocery store"—devising a useful writing system for your own language is a difficult task. Compared to inventing a good system for writing *words*, inventing a reasonable way to write *numbers* is a snap.

That's four. No problem.



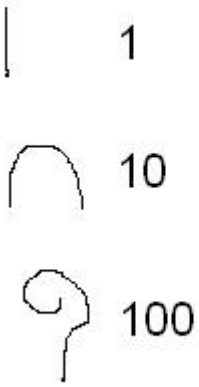
That's forty-four, and that *is* a problem, since the reader has to count all the marks the writer made. A reader who knows how to count by 5's has an easier time with this:



But the burden is still on the writer to make all those strokes. And if you have a lot more strokes it can become hard to read as well.

Big numbers can be hard to get your head around, but people do get the hang of it. We learn the habit of thinking in terms of larger groups, and groups of groups, etc. Sooner or later someone will get the idea of taking one of those blocks of five strokes and replacing it by a single symbol, and then representing five of those symbols by yet another symbol, etc.

The hieroglyphic number-writing system of ancient Egypt works exactly this way, except the basic quantity for grouping is ten rather than five. The first few symbols are:

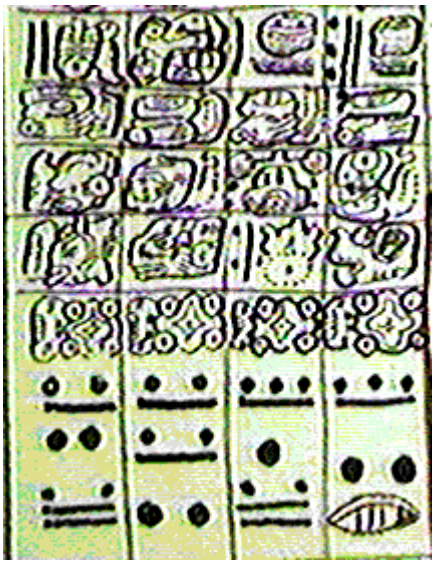


You wrote a number, say two hundred seventeen, simply by grouping the appropriate numbers of 1 symbols, 10 symbols, 100 symbols, etc. It doesn't much matter in this system what order you group the symbols in---you can put the tens above the ones, or below them, or to the right or the left.

Look closely at this Egyptian hieroglyphic inscription and you can see the numbers 23 and 15 in the top row, and 18 in the bottom row.

It is interesting to compare this to the system we presently use for writing numbers. The string of symbols "459" is interpreted to mean 4x100 + 5x10 + 9x1. Our number system is *positional*, which means that a single symbol, like 4, can represent 4, or 40, or 400,… depending upon its position. This is in marked contrast to the *additive* Egyptian system, in which the positions are irrelevant. Thus a positional system uses a fixed collection of symbols— in our case the digits 0 through 9--to represent arbitrarily large quantities, while in Egyptian you need to introduce a new symbol for each successive power of ten. A positional system pretty much forces you to include a symbol that does not add to the value of the number but simply holds a position---a zero.

Both our system and the Egyptian one are *base ten*, or *radix ten,* or *decimal* systems: the symbols in the Egyptian system are all powers of ten, and the symbols in our system are *weighted* by powers of ten, depending upon the position. The reason for using powers of ten is surely because people counted on their fingers, but there's nothing special about ten in the mathematical sense; any whole number larger than 1 would do just as well as a base. Positional number systems were invented independently in several different cultures. The earliest appearance was in Babylonian culture in Mesopotamia around 1900 BC, and there was an independent development in Mesoamerica, among the Olmec and Maya peoples. The Babylonians, incidentally, used a base 60 system, and the Mesoamericans base 20. Our own positional system originated in India around the 8th century AD and traveled to Europe via the Arab world. (That's why we call them Arabic numerals, or sometimes Hindu-Arabic numerals.)



Detail from the "Dresden Codex", showing four numbers written in Maya positional number system

Bits and Bytes

Attorney: OK, let's prepare you for your testimony. Do you know what color my dress is?
Client: It's blue.
Arrorney: Wrong! My dress is *blue, but the right answer is "yes".*

The smallest amount of information that you can give, and still give *some* information, is the answer to a single yes-or-no question, to distinguish between two alternatives. This amount of information is called a *bit.* You can represent the two possible values of a bit of information by the words "yes" and "no", or "true" and "false", or "on" and"off", but they are usually denoted by the digits 1 and 0. In fact the word "bit", which was coined in the early 1940's by John Tukey, comes in part from the meaning "just a little bit of information", but is also a contraction of "binary digit". (Tukey, who must have had quite a flair for inventing new words, also coined "software".)

Bits, Bytes, Kilobytes, and Such

There are two possible values of a single bit, 4 possible values for a sequence of two bits (that is, 00, 01, 10 or 11), 8 possible values for a sequence of 3 bits, ...In general, there are 2^k different sequences of k bits.

Information stored in computers is usually given in chunks of 8 bits. A chunk of 8 bits is called a **byte**. (The etymology of the term is uncertain; it first surfaced in IBM technical reports in the 1950's.) There are thus 2^8=256 possible different values for a byte.

Powers of two come up so often in this subject that it is useful to have a table of them around:

n	**2^n**	**n**	**2^n**
0	1	11	2048
1	2	12	4096
2	4	13	8192
3	8	14	16384
4	16	15	32768
5	32	16	65536
6	64	17	131072
7	128	18	262144
8	256	19	524288
9	512	20	1048576
10	1024	21	2097152

A *kilobyte*, by rights, ought to mean one thousand bytes, but everything with computers goes by powers of two In practice one is much more likely to encounter a chunk of 2^{10} = 1024 bytes than a chunk of 1000 bytes, so a kilobyte (KB) is 1024 bytes. If you look at the "Properties" of a Windows file that contains between 1024 and one million bytes of data, you'll see the size given in both KB and bytes, so you can verify this.

Likewise, a megabyte (MB) is 2^{20} = 1048576 bytes, and a gigabyte (GB) is 2^{30} , or about 1.07 billion bytes. The text in this document, which I'm preparing in Microsoft Word, would ordinarily take up about 20KB, but what with all the pasted-in pictures and such it requires about 200KB. The now nearly obsolete floppy diskettes everyone used to carry around held about 1MB, a CD about 700 MB, a DVD about 5 GB, the hard disk on the computer on which I'm typing this document about 40GB.

If you have a dialup Internet connection, then your modem transfers, at best, 56 kilo*bits* per second. (I confess that I am uncertain whether a kilobit means exactly 1000 bits or 1024 bits; I'm pretty sure it's 1000 bits.) My home DSL connection will get up to about 70 KB/second on a good day.

Hexadecimal Notation

It's hard to read a sequence of bits like 11011001 and tell at a glance that it is different from 11010001. So people who have to look at bit patterns a lot use a human-readable scheme called "hexadecimal" , which represents each block of 4 bits by a single symbol:

Block of four bits	Hexadecimal equivalent
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Note that in the left-hand column of the table above, the sixteen different bit patterns are listed in alphabetical order (under the assumption that 0 precedes 1 in the alphabet). We want to use a single conventional symbol for each pattern, so we switch from digits to letters when we run out of digits. This might lead you to wonder why we just didn't start with A and use only letters—but there's a good reason, as you'll see shortly.

A single byte is made up of two blocks of four bits, and thus can be represented by two hexadecimal symbols---two "hex digits", in the lingo. The two bytes at the beginning of this section would be represented as D9 and D1.

Writing Numbers with Bits

As mentioned above, any whole number greater than 1 can be used as the base of a positional number system. In our decimal positional system, there are ten digits: 0,1,2,…,9. In a base five system, there are only the five digits 0,1,2,3,4. The positions in a string of digits are weighted by powers of five. So the string

4013

is the base five representation of the number

$4x5^3+0x5^2+1x5^1+3x5^0=4x125+0x25+1x5+3x1,$

which is five hundred and eight. This is annoying and confusing, since when you see "4013" you really, really want to say "four thousand thirteen", but we are giving an entirely different interpretation to such strings of digits. To make things absolutely clear, we will sometimes write

4013_5

to mean "the number whose base five representation is the string 4013". (This theme of different coding systems giving different interpretations to the same sequence of symbols comes up again and again.)

The smallest possible base of a positional number system is two. A base two system is said to be a *binary* system. There are only two digits, 0 and 1, which explains the term "binary digit" as the meaning of "bit". The positions are weighted by the powers of two: 1,2,4,8,16,etc. So, for example,

11001001

has 1's in the positions corresponding to 1,8,64 and 128, and thus represents the number

128+64+8+1,

or two hundred and one. Again, we could write something like

$11001001_2 = 201_{10}$

to be absolutely clear about how the symbols are meant to be interpreted.

Base Conversion

In general, if you need to find the decimal representation of a number given in binary, you can proceed as above, starting at the right, and writing the successive powers of two under the bits of the number:

1	0	1	1	0	1	1	0	1
256	128	**64**	**32**	16	**8**	**4**	2	**1**

256+64+32+8+4+1=365

Then just add up the powers of two that correspond to positions containing a 1. This shows $101101101_2=365_{10}$

Here is another method (another *algorithm*). Write down the value 0. Then begin at the leftmost bit and read from left to right. At each successive bit, either double the value

you've written (if the bit is zero) or double it and add 1 (if the bit is 1). The table below illustrates this algorithm.

	1	0	1	1	0	1	1	0	1
0	1=2x0+1	2=2x1	5=2x2+1	11=2x5+1	22=2x11	45	91	182	365

What if you want to go in the other direction? If you have the number in some familiar format, like decimal notation, how do you compute its binary representation?

Algorithm 1. (Left-to-right.)Make a table with three rows. In the first row, write the powers of two from right to left, until you exceed the value of the number N you are trying to represent. Then, write the number N in the first column of the second row. Beginning at the leftmost column, repeatedly perform the following operation: Subtract the value in the first row of the next column from the value in the second row of the current column. If the result is zero or more, write the difference in the second row of the next column and 1 in the third row. Otherwise, just copy the value in the second row of the current column to the second row of the next column, and write '0' below it. The binary representation of N appears in the last row of the table. The example below illustrates the computation of the binary representation of four hundred eighty-three.

512	256	128	64	32	16	8	4	2	1
483	227=483-256	99=227-128	35	3	3	3	3	1	
	1	1	1	1	0	0	0	1	1

Algorithm 2. (Right-to-left.) Make a table with two rows. Write N in the rightmost column of the first row. At each step, divide the value in the first row by 2, writing the quotient to its left, and the remainder below it. Repeat until you get a quotient of 0. The binary representation appears in the last row.

0	1	3	7	15	30	60	120	241	483
	1	1	1	1	0	0	0	1	1

Hexadecimal Notation, Again

Observe that the block 1001 of four bits corresponding to the hex digit 9 really is the binary representation of nine. Similarly, the block 0101 of four bits corresponding to hex 5 really does represent five in binary, since the zero at the left doesn't change the value. And the bit strings corresponding to the hex digits A,B,C,D,E,F are just the binary representations of the numbers ten through fifteen.

This means that you can think of the hex digits as the numbers 0 through fifteen, and of a string of hex digits as the *base sixteen* representation of a number. With this interpretation, a string like

D3A

Represents $13x16^2+3x16+10 = 3386$

Now remember that hexadecimal notation was just supposed to be a convenient way of writing strings of bits, so that D3A is just shorthand for

0110100111010.

This string of bits is itself the binary representation of a number, and that number is 3386. There's not much difference between base two and base sixteen representations of numbers: Each hex digit is just an abbreviation for a block of four bits. It's a simple matter to translate between the two representations (just remember that if you are translating from binary to hex to first add enough bits at the left so that the number of bits is a multiple of four).

Arithmetic

Suppose you want to add the two numbers

11011001_2

and

11010001_2,

and express the answer in binary. It's tempting to begin by translating these into some more familiar representation, like decimal, add them the usual way, and then convert back into binary.

But there's no reason to do this! The very same algorithms that you learned in elementary school for adding and multiplying work in any base. For addition, you just have to keep in mind that in binary 1+1=10 and 1+1+1=11, so you have to carry whenever you add two or more 1's:

```
   1     1           1
   1 1 0 1 1 0 0 1
  +1 1 0 1 0 0 0 1
-----------------
1 1 0 1 0 1 0 1 0
```

The little red bits in the example show where carries were performed.

Now there's an important point to be made here, one that explains why we spend our time learning about this stuff: It is *easy* to build a machine that carries out addition as illustrated above, quite a lot easier than trying to manipulate the decimal digits of the numbers directly. A machine that has to perform many such calculations can do them all in binary. Base conversion only has to occur at the end of the process, when the results are shown to humans.

Multiplication is no harder. In fact multiplication and long division are much *easier* to do in binary than in decimal, because there are no multiplication tables to memorize, and absolutely no guesswork involved in determining which digits to write in the quotient in a division problem.

```
          1 1 0 1
        X 1 0 1 1
        ---------
          1 1 0 1
        1 1 0 1
    + 1 1 0 1_____
    ---------------
    1 0 0 0 1 1 1 1
```

Writing Text with Bits

One of the themes of this course is that all information manipulated by computers is represented by bits. That includes ordinary text. Let's see how this is done.

A computer keyboard (at least the one I'm working on now) has 47 keys for typing characters, and two characters per key, giving a total of 94 different characters. An ordinary text document will also have to include information about spacing and line breaks, so provisions need to be made for special "whitespace characters", but the total number of characters is about a round one hundred.

Now you can't represent one hundred different values with 6 bits, since that gives you only 2^6=64 possible patterns, but you can get one hundred different patterns with 7 bits, and still have some space left over.

The standard method for representing characters by bit strings actually uses *eight* bits--- that is, one byte--- to represent each character, but the highest-order (the leftmost) bit is always zero, so it really is a seven-bit representation. This coding scheme is called ASCII, which stands for "American Standard Code for Information Interchage" and is pronounced "asky". The character codes are given in the table below.

Dec	Hex	Sym	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	NUL	32	20		64	40	@	96	60	`
1	1	SOH	33	21	!	65	41	A	97	61	a
2	2	STX	34	22	"	66	42	B	98	62	b
3	3	ETX	35	23	#	67	43	C	99	63	c
4	4	EOT	36	24	$	68	44	D	100	64	d
5	5	ENQ	37	25	%	69	45	E	101	65	e
6	6	ACK	38	26	&	70	46	F	102	66	f
7	7	BEL	39	27	'	71	47	G	103	67	g
8	8	BS	40	28	(	72	48	H	104	68	h
9	9	TAB	41	29	)	73	49	I	105	69	i
10	A	LF	42	2A	*	74	4A	J	106	6A	j
11	B	VT	43	2B	+	75	4B	K	107	6B	k
12	C	FF	44	2C	,	76	4C	L	108	6C	l
13	D	CR	45	2D	-	77	4D	M	109	6D	m
14	E	SO	46	2E	.	78	4E	N	110	6E	n
15	F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	

Here are a few things to note about this coding scheme:

You might want to encode a digit character, like '7', by the number 7 itself. But in fact, as you can see, the encoding of '7' doesn't have much to do with the value 7, and certainly the encoding of a letter like 'B' has nothing to do with the value eleven represented by the hex digit 'B'. On the other hand, the digits are encoded in order, so for example the encoding of '7' is indeed three more than the encoding of '4'. Likewise, the upper-case letters are represented in alphabetical order by consecutive code values, and so are the lower-case letters. You'll notice that a few characters were inserted between the upper- and lower-case letters. This is actually to make the process of conversion between lower and upper case simpler---you'll see what I mean if you write down the encodings of 'B', 'C', 'b' and 'c' in binary.

The first thirty-three values encode non-printing characters. Some of these are familiar, like SP (encoded by 32), which stands for "space", and HT (encoded by 9), which stands for "horizontal tab", but most of them are obscure. They refer to functions (like "line feed" and "form feed") of printing equipment that was in common use when the ASCII code was invented.

Since we've used eight bits to encode what we could have encoded in seven, there is a significant amount of waste. We could fit 14% more text on a compact disk if we eliminated that extra bit. Now there is a good reason to have an 8-bit encoding---as we shall see, computers are built in such a way as to manipulate data in byte-sized chunks, and it would actually take longer to extract characters if you had two parts of different characters stored in a single byte. But the point is still an important one---in situations where we care more about storage resources than computation time, it makes sense to try to squeeze the information into fewer bits. We will take up this matter when we talk about data compression.

This system is heavily biased towards the English language. While many other languages use the same Roman alphabet as English, most of them use special accent marks on the letters (like ñ in Spanish, and ç in French). Turkish has the letter ı (a dotless i) as well as İ (dotted upper-case I). And, of course, many widely used languages use different scripts entirely. Some, like Chinese, use thousands of different characters. In an effort to internationalize the representation of characters by bits, a new standard has been developed, called **Unicode.** Unicode uses 16 bits (two bytes) to represent each character. The Unicode representation of '0' is 0030_{16}, and similarly for all the standard ASCII characters: The leftmost eight bits are all zero and the rightmost eight bits are the usual ASCII representation. The Unicode representations of that Spanish ñ (00F1) and French ç (00E7) as well as many other variant Roman characters likewise have the leftmost eight bits all zero, but have the next bit equal to 1. The code for the Turkish ı (0131) does not fit into a single byte, and the basic Chinese character set in Unicode uses all the values 4E00 and 9FBF.

Endnotes

1. *Sumerians and the invention of writing.* True writing is the representation of spoken language by graphic symbols. It need not necessarily represent the words phonetically--although all writing systems, including Chinese, do contain at least some phonetic elements. The earliest tablets from Sumer seem to be in a kind of picture-writing in which a symbol resembling, say, an eye might signify an eye, or seeing, or some related concept, but does not stand for an actual spoken word. The transition from such proto-writing to true writing took several centuries. However, once the *idea* of true writing spreads to other language communities, writing systems for new languages can be developed fairly quickly, without having to go through the proto-writing stage. Probably most of the scripts used throughout the world were developed in this way.

The question of who first invented writing does not have a completely clear answer—it's uncertain when the transition from picture-writing to true writing in Sumer actually took place, and whether inscriptions from Egypt or the Indus Valley in India might be earlier examples of true writing.

2. Mesopotamian Number Systems. The Sumerians used a base sixty system, but it was an additive number system, like that used by the Egyptians, rather than a positional one. The original positional number system of the Babylonians actually did not have a zero: Zeros in the least significant positions (for us, at the right-hand end of the number) could be dropped---this would be like representing fourteen, one hundred forty, one thousand four hundred, etc., all by the string 14---it was assumed the reader could infer the correct order of magnitude from the context. Zeros in the interior of a number were more of a problem. With a base sixty system you don't run across them too often, since there's only a one in sixty chance that a randomly chosen three-digit number will require a zero in the middle digit. On at least one tablet, the scribe left a little extra space between the digits to indicate where a zero would ordinarily go.

3. *Mesoamerican Number Systems.* Since the Maya system is a base 20 positional system, each digit represents a number between zero and nineteen. The symbol that looks like an eye or a shell is zero; the other digits are made from a combination of lines and dots, with a dot representing 1 and a line 5. The multi-digit numbers in the illustration are written with the most significant digit at the top. If you ever find yourself in Cancun on Spring Break and are sober enough to look out the window of the buses that drive along the main road, you'll see Mayan numerals on the highway kilometer sings.

Most extant samples of Mayan writing are engraved on stone. Almost all of the books on paper were destroyed by Spanish missionaries; the Dresden codex illustrated in the notes is one of the few that survive.

The great intellectual achievements of Mesoamerican civilization are traditionally attributed to the Maya, but there is a recent current of thought suggesting that the Olmec people may have been the "mother civilization", and the originators of both writing and the positional number system. The jury still seems to be out on this.

4. The conversation between the attorney and client really did take place. (Reported to me by the client.)

5. “A machine that has to perform many such calculations can do them all in binary. Base conversion only has to occur at the end of the process, when the results are shown to humans.” We’ll say a bit more about this later when we talk about the history of computing machines. Modern computer experts take binary for granted, but this point was not obvious to the designers of the earliest electrical and electronic computers, who preferred to build machines that did all the work in decimal.

Sources for the historical information contained in these notes:

The Sumerians, by Samuel Noah Kramer

A Universal History of Numbers, by Georges Ifrah

“Oldest New World Writing Suggests Olmec Innovation”, Erik Stokstad in *Science* Magazine, vol. 298, Dec. 2002.

Also, a very good History of Mathematics website at St. Andrews University in Scotland, with many links and references to printed literature: www-history.mcs.st-andrews.ac.uk