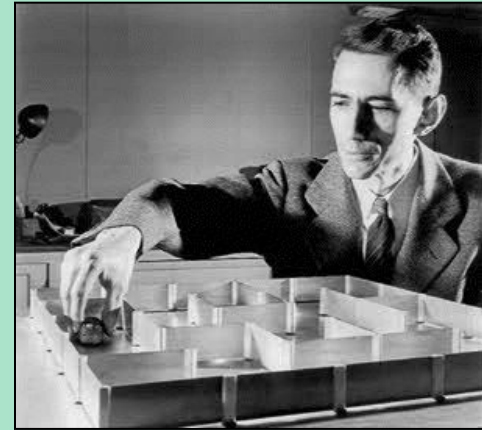# Binary Representation

Abraham Gebrekidan
EIP 01
August 16, 2024

Once upon a time . . .

Claude Shannon

Claude Shannon was one of the pioneers who shaped computer science in its early years. In his master's thesis, Shannon showed how it was possible to use Boolean logic and switching circuits to perform arithmetic calculations. That work led directly to the use of binary arithmetic inside computers—an idea that initially generated considerable resistance among more established scientists.

Claude Shannon (1916-2001)

Shannon is best known as the inventor of *information theory,* which provides the mathematical foundation for understanding data communication. To many, however, Shannon's great contribution was that he never lost his sense of fun throughout his long career.

Binary Representation

The Power of Bits

- The fundamental unit of memory inside a computer is called a *bit*—a term introduced in a paper by Claude Shannon as a contraction of the words *binary digit.*

- An individual bit exists in one of two states, usually denoted as **0** and **1**.

- More sophisticated data can be represented by combining larger numbers of bits:
  - Two bits can represent four (2 × 2) values.
  - Three bits can represent eight (2 × 2 × 2) values.
  - Four bits can represent 16 (2^4) values, and so on.

- This laptop has 16GB of main memory and can therefore exist in $2^{549,755,813,888}$ states. If you were to write that number out, it would contain more than fifty billion digits.

Leibniz and Binary Notation

- Binary notation is an old idea. It was described back in 1703 by the German mathematician Gottfried Wilhelm von Leibniz.

- Writing in the proceedings of the French Royal Academy of Science, Leibniz describes his use of binary notation in a simple, easy-to-follow style.

- Leibniz's paper further suggests that the Chinese were clearly familiar with binary arithmetic 2000 years earlier, as evidenced by the patterns of lines found in the *I Ching*.
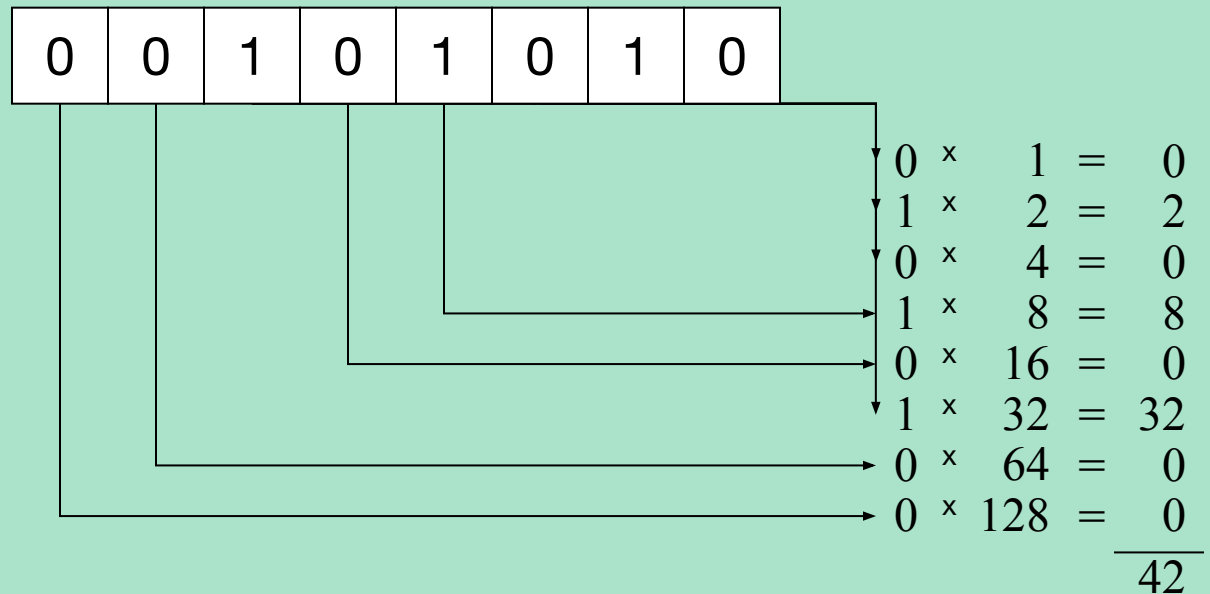

O N

U E

·lle donne le

ohy.

ait suivant la
le dix carac-
qui signifient
·uf inclusive-
:, & on écrit

dix ; par 10 ; & dix fois dix, ou *cent*, par 100; & dix fois cent, ou *mille*, par 1000 ; & dix fois mille, par 10000. Et ainsi de suite.

Mais au lieu de la progression de dix en dix, j'ai employé depuis plusieurs années la progression la plus simple de toutes, qui va de deux en deux ; ayant trouvé qu'elle sert à la perfection de la science des Nombres. Ainsi je n'y employe point d'autres caractères que o & 1, & puis allant à deux, je recommence. C'est pourquoi *deux* s'écrit ici par 10, & deux fois deux ou *quatre* par 100; & deux fois quatre ou *huit* par 1000; & deux fois huit ou *seize* par 10000, & ainsi de suite. Voici *la Table des Nombres* de cette façon, qu'on peut continuer tant que l'on voudra.

Using Bits to Represent Integers

- Binary notation is similar to decimal notation but uses a different *base*. Decimal numbers use 10 as their base, which means that each digit counts for ten times as much as the digit to its right. Binary notation uses base 2, which means that each position counts for twice as much, as follows:

0	0	1	0	1	0	1	0

$$0 \times 1 = 0$$
$$1 \times 2 = 2$$
$$0 \times 4 = 0$$
$$1 \times 8 = 8$$
$$0 \times 16 = 0$$
$$1 \times 32 = 32$$
$$0 \times 64 = 0$$
$$0 \times 128 = 0$$
$$\overline{42}$$

Numbers and Bases

- The calculation at the end of the preceding slide makes it clear that the binary representation 00101010 is equivalent to the number 42. When it is important to distinguish the base, the text uses a small subscript, like this:

$$00101010_2 = 42_{10}$$

- Although it is useful to be able to convert a number from one base to another, it is important to remember that the number remains the same. What changes is how you write it down.

- The number 42 is what you get if you count how many stars are in the pattern at the right. The number is the same whether you write it in English as *forty-two,* in decimal as 42, or in binary as 00101010.

- Numbers do not have bases; representations do.

Octal and Hexadecimal Notation

- Because binary notation tends to get rather long, computer scientists often prefer ***octal*** (base 8) or ***hexadecimal*** (base 16) notation instead. Octal notation uses eight digits: 0 to 7. Hexadecimal notation uses sixteen digits: 0 to 9, followed by the letters A through F to indicate the values 10 to 15.

- The following diagrams show how the number forty-two appears in both octal and hexadecimal notation:

octal

5	2

$$2 \times 1 = 2$$
$$5 \times 8 = 40$$
$$\overline{42}$$

hexadecimal

2	A

$$10 \times 1 = 10$$
$$2 \times 16 = 32$$
$$\overline{42}$$

- The advantage of using either octal or hexadecimal notation is that doing so makes it easy to translate the number back to individual bits because you can convert each digit separately.

Exercises: Number Bases

- What is the decimal value for each of the following numbers?

$$10001_2 \qquad\qquad 177_8 \qquad\qquad AD_{16}$$

17   127   173

- As part of a code to identify the file type, every Java class file begins with the following sixteen bits:

1	1	0	0	1	0	1	0	1	1	1	1	1	1	1	0

How would you express that number in hexadecimal notation?

1	1	0	0	1	0	1	0	1	1	1	1	1	1	1	0

$CAFE_{16}$

Exercise: Perfect Numbers in Binary

- Greek mathematicians took a special interest in numbers that are equal to the sum of their ***proper divisors,*** which are simply those divisors less than the number itself. They called such numbers ***perfect numbers***. For example, 6 is a perfect number because it is the sum of 1, 2, and 3, which are the integers less than 6 that divide evenly into 6. The next three perfect numbers—all of which were known to the Greeks—are 28, 496, and 8128.

- Convert each of these numbers into its binary representation:

$$6 = 110_2$$
$$28 = 11100_2$$
$$496 = 111110000_2$$
$$8128 = 1111111000000_2$$

Bits and Representation

- Sequences of bits have no intrinsic meaning except for the representation that we assign to them, both by convention and by building particular operations into the hardware.

- As an example, a 32-bit word represents an integer only because we have designed hardware that can manipulate those words arithmetically, applying operations such as addition, subtraction, and comparison.

- By choosing an appropriate representation, you can use bits to represent any value you can imagine:
  - Characters are represented using numeric character codes.
  - Floating-point representation supports real numbers.
  - Two-dimensional arrays of bits represent images.
  - Sequences of images represent video.
  - And so on . . .

Bits Are Everywhere
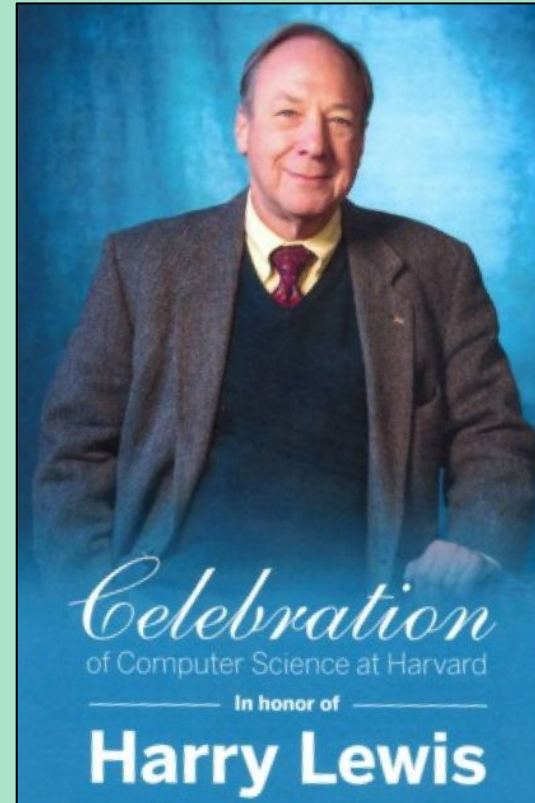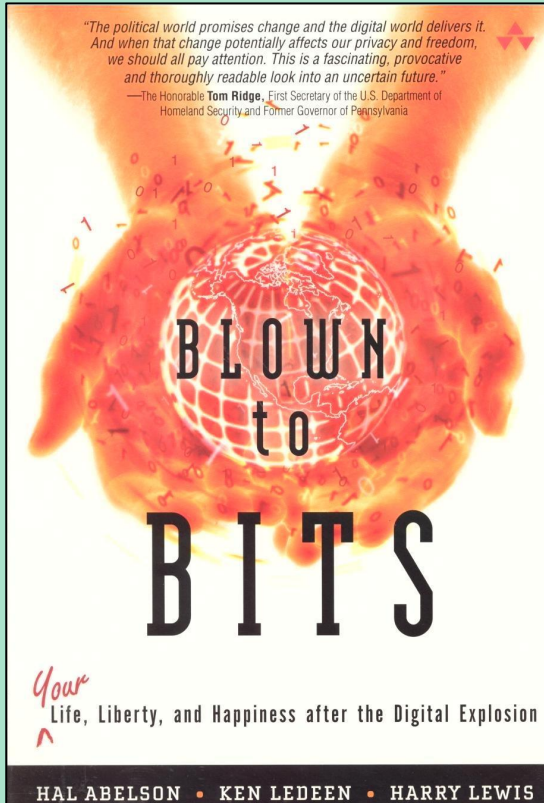
Representing Characters

- Computers use numeric encodings to represent character data inside the memory of the machine, in which each character is assigned an integral value.

- Character codes, however, are not very useful unless they are standardized. When different computer manufacturers use different coding sequence (as was indeed the case in the early years), it is harder to share such data across machines.

- The first widely adopted character encoding was ASCII (*American Standard Code for Information Interchange*).

- With only 256 possible characters, the ASCII system proved inadequate to represent the many alphabets in use throughout the world. It has therefore been superseded by Unicode, which allows for a much larger number of characters.

The ASCII Subset of Unicode

The following table shows the first 128 characters in the Unicode character set, which are the same as in the older ASCII scheme:

	0	1	2	3	4	5	6	7
$00x$	\000	\001	\002	\003	\004	\005	\006	\007
$01x$	\b	\t	\n	\011	\f	\r	\016	\017
$02x$	\020	\021	\022	\023	\024	\025	\026	\027
$03x$	\030	\031	\032	\033	\034	\035	\036	\037
$04x$	*space*	!	"	#	$	%	&	'
$05x$	(	)	*	+	,	-	.	/
$06x$	0	1	2	3	4	5	6	7
$07x$	8	9	:	;	<	=	>	?
$10x$	@	A	B	C	D	E	F	G
$11x$	H	I	J	K	L	M	N	O
$12x$	P	Q	R	S	T	U	V	W
$13x$	X	Y	Z	[	\	]	^	_
$14x$	`	a	b	c	d	e	f	g
$15x$	h	i	j	k	l	m	n	o
$16x$	p	q	r	s	t	u	v	w
$17x$	x	y	z	{			}	~	\177

Hardware Support for Characters
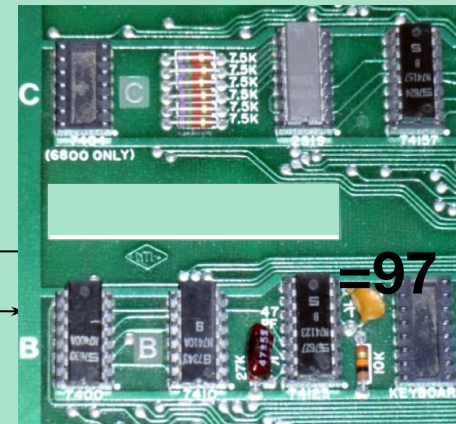
Strings as an Abstract Idea

- Characters are most often used in programming when they are combined to form collections of consecutive characters called **_strings_**.

- As you will discover when you have a chance to look more closely at the internal structure of memory, strings are stored internally as a sequence of characters in sequential memory addresses.

- The internal representation, however, is really just an implementation detail. For most applications, it is best to think of a string as an abstract conceptual unit rather than as the characters it contains.

- JavaScript emphasizes the abstract view by defining a built-in string type that defines high-level operations on string values.

The End

Tom Lehrer on "The New Math"

$$3\ 4\ 2_{8}$$
$$-1\ 7\ 3_{8}$$

Tom Lehrer (1928–)