Computer Science 1000: Part #3

Binary Numbers

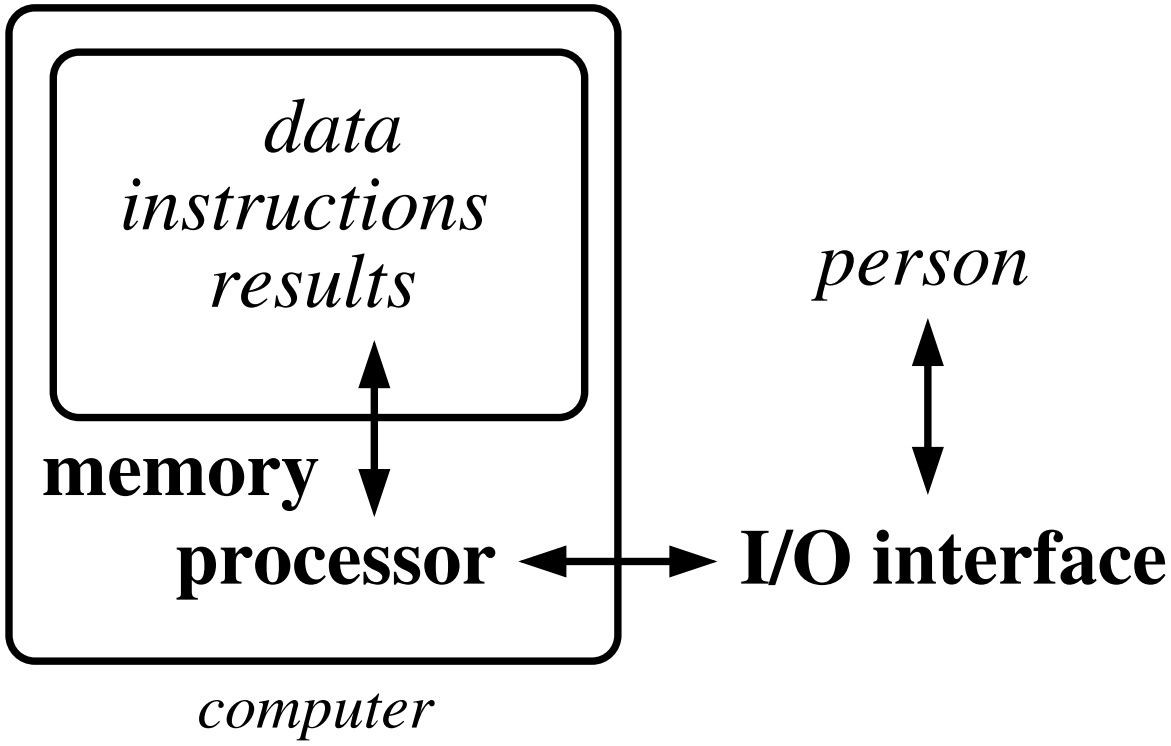
Computer Organization: An Overview

An Historical Interlude

Representing Numbers in Binary

Representing Text, Sound, and Pictures in Binary

Computer Organization: An Overview
The Von Neumann Architecture (1945)



Also known as the stored-program architecture

Computer Organization: An Overview (Cont'd)
What is a Computer (Really)?

A computer is a machine that

(1) stores a very, very large number of numbers and
(2) performs very, very long specified sequences of very simple operations on these numbers
(3) very, very fast.

Computer Organization: An Overview (Cont'd)
Guiding Principles

There are two main principles of computer organization:

1. **Levels of Abstraction**: A complex system can be described as a hierarchy of levels, where collections of interacting entities at one level are encapsulated in a single entity at a higher level (see Textbook, Figure 5.1, p. 223).
2. **Internal vs. External Representation**: Within a complex system, the representations used internally by an entity to perform its functions need not be those with which it interacts externally with other entities.

Computer Organization: An Overview (Cont'd)
Guiding Principles (Cont'd)

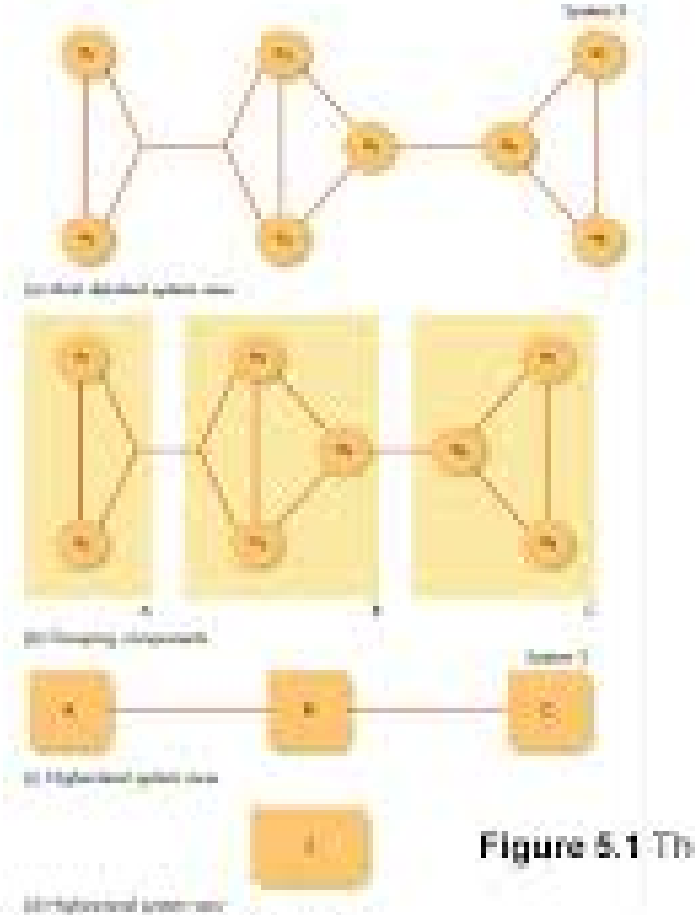


Figure 5.1 The Concept of Abstraction

A Historical Interlude: The Glory of Decimal Numbers

- Decimal numbers (**base-10 positional notation**) were developed around 1200 years ago in the Middle East.
- Rapidly replaced older non-positional systems, e.g.,

MCMXXXVII *(Roman)*

$$(1000 + (1000 - 100) + 10 + 10 + 10 + 5 + 1 + 1)$$

vs.

1937_{10} *(Hindu-Arabic)*

$$((1 \times 10^3) + (9 \times 10^2) + (3 \times 10^1) + (7 \times 10^0))$$

A Historical Interlude: (Cont'd)
The Glory of Decimal Numbers (Cont'd)

Position	Decimal
1	$10^0 \rightarrow 1$
2	$10^1 \rightarrow 10$
3	$10^2 \rightarrow 100$
4	$10^3 \rightarrow 1000$
5	$10^4 \rightarrow 10,000$
6	$10^5 \rightarrow 100,000$
7	$10^6 \rightarrow 1,000,000$
8	$10^7 \rightarrow 10,000,000$
9	$10^8 \rightarrow 100,000,000$
10	$10^9 \rightarrow 1,000,000,000$

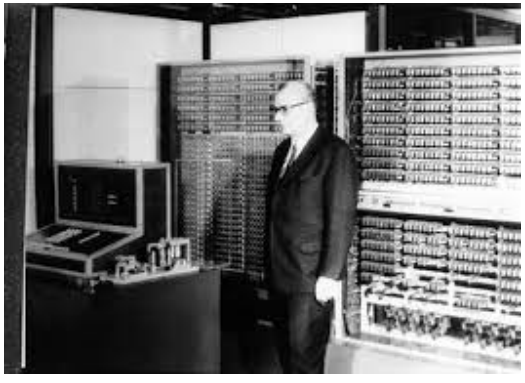
A Historical Interlude: (Cont'd)
The Tyranny of Decimal Numbers

Centrality of decimal numbers in mathematics led to their being both the internal and external representations of numbers in the earliest computing machines, e.g., the 1642 addition machine of Blaise Pascal (1623–1662).

A Historical Interlude: (Cont'd)
The Tyranny of Decimal Numbers (Cont'd)

This reliance on decimal internal representations continued into the early days of electromechanical computing.

Harvard Mark I
(1944)



Zuse Z3
(1941)

Colossus
(1944)

A Historical Interlude: (Cont'd)
The Glory of Binary Numbers

The emergence of fully electronic computing in the late 1940s led to binary (**base-2 positional notation**) internal representations of numbers.

$$167_{10}$$

$$((1 \times 10^2) + (6 \times 10^1) + (7 \times 10^0))$$

vs.

$$10100111_2$$

$$((1 \times 2^7) + (0 \times 2^6) + (1 \times 2^5) + (0 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0))$$

A Historical Interlude: (Cont'd)
The Glory of Binary Numbers (Cont'd)

Position	Decimal	Binary
1	$10^0 \rightarrow 1$	$2^0 \rightarrow 1$
2	$10^1 \rightarrow 10$	$2^1 \rightarrow 2$
3	$10^2 \rightarrow 100$	$2^2 \rightarrow 4$
4	$10^3 \rightarrow 1000$	$2^3 \rightarrow 8$
5	$10^4 \rightarrow 10,000$	$2^4 \rightarrow 16$
6	$10^5 \rightarrow 100,000$	$2^5 \rightarrow 32$
7	$10^6 \rightarrow 1,000,000$	$2^6 \rightarrow 64$
8	$10^7 \rightarrow 10,000,000$	$2^7 \rightarrow 128$
9	$10^8 \rightarrow 100,000,000$	$2^8 \rightarrow 256$
10	$10^9 \rightarrow 1,000,000,000$	$2^9 \rightarrow 512$

In binary, need 10 digits to represent a factor of ≈ 1000.

A Historical Interlude: (Cont'd)
The Glory of Binary Numbers (Cont'd)

Done for reliability, e.g., distinguishing between and maintaining two voltage levels is **much** easier to do than distinguishing between and maintaining ten voltage levels.

Decimal		**Binary**
(0) +0		+0 (0)
(1) +5		
(2) +10		
(3) +15		
(4) +20	vs.	
(5) +25		
(6) +30		
(7) +35		
(8) +40		
(9) +45		+45 (1)

A Historical Interlude: (Cont'd)
The Glory of Binary Numbers (Cont'd)

To convert a binary number to its decimal equivalent, add up the powers of two corresponding to the 1's in the number, e.g.,

$$\begin{aligned} 110101_2 &= 2^5 + 2^4 + 2^2 + 2^0 \\ &= 32 + 16 + 4 + 1 \\ &= 53_{10} \end{aligned}$$

$$\begin{aligned} 10101100_2 &= 2^7 + 2^5 + 2^3 + 2^2 \\ &= 128 + 32 + 8 + 4 \\ &= 172_{10} \end{aligned}$$

A Historical Interlude: (Cont'd)
The Glory of Binary Numbers (Cont'd)

To convert a decimal number to its binary equivalent, repeatedly divide by two and read the remainder digits in reverse (from last to first), e.g.,

	Quotient	Remainder	
$53/2 \Rightarrow$	26	1	
$26/2 \Rightarrow$	13	0	
$13/2 \Rightarrow$	6	1	$\Longrightarrow 110101_2$
$6/2 \Rightarrow$	3	0	
$3/2 \Rightarrow$	1	1	
$1/2 \Rightarrow$	0	1	

A Historical Interlude: (Cont'd)
The Glory of Binary Numbers (Cont'd)

	Quotient	Remainder	
$23/2 \Rightarrow$	11	1	
$11/2 \Rightarrow$	5	1	
$5/2 \Rightarrow$	2	1	$\Longrightarrow 10111_2$
$2/2 \Rightarrow$	1	0	
$1/2 \Rightarrow$	0	1	

	Quotient	Remainder	
$8/2 \Rightarrow$	4	0	
$4/2 \Rightarrow$	2	0	$\Longrightarrow 1000_2$
$2/2 \Rightarrow$	1	0	
$1/2 \Rightarrow$	0	1	

Representing Numbers in Binary

Basic base-2 positional notation (**unsigned binary**) can handle the representation and addition of positive integers , e.g.,

$$
\begin{array}{cccccc}
 & 1 & 1 & 1 & 1 & 1 \\
 & & 1 & 1 & 0 & 1 & 1_2 \\
+ & & & 1 & 1 & 0 & 1_2 \\
\hline
 & 1 & 0 & 1 & 0 & 0 & 0_2
\end{array}
\quad \Longleftrightarrow \quad
\begin{array}{cccc}
 & 1 & & \leftarrow \text{carry} \\
 & 2 & 7_{10} & \\
+ & 1 & 3_{10} & \\
\hline
 & 4 & 0_{10} &
\end{array}
$$

. . . But what about negative integers? . . .
. . . Or fractional numbers? . . .

Representing Numbers in Binary: (Cont'd)
Sign/Magnitude Representation

- Use leftmost bit to encode sign (+ve as 0, -ve as 1), e.g.,

Negative		**Positive**	
-0_{10}	100_2	000_2	$+0_{10}$
-1_{10}	101_2	001_2	$+1_{10}$
-2_{10}	110_2	010_2	$+2_{10}$
-3_{10}	111_2	011_2	$+3_{10}$

- Has two representations of zero, and both can arise during arithmetic; this causes problems for computer designers.

Representing Numbers in Binary: (Cont'd)
Sign/Magnitude Representation (Cont'd)

Negative		**Positive**	
-0_{10}	1000_2	0000_2	$+0_{10}$
-1_{10}	1001_2	0001_2	$+1_{10}$
-2_{10}	1010_2	0010_2	$+2_{10}$
-3_{10}	1011_2	0011_2	$+3_{10}$
-4_{10}	1100_2	0100_2	$+4_{10}$
-5_{10}	1101_2	0101_2	$+5_{10}$
-6_{10}	1110_2	0110_2	$+6_{10}$
-7_{10}	1111_2	0111_2	$+7_{10}$

Representing Numbers in Binary: (Cont'd)
Two's Complement Representation

- Represent negative number by taking unsigned binary positive version and starting after rightmost 1, complement every bit to the left (see also Textbook, p. 160), e.g.,

Negative		**Positive**	
—		000_2	$+0_{10}$
-1_{10}	111_2	001_2	$+1_{10}$
-2_{10}	110_2	010_2	$+2_{10}$
-3_{10}	101_2	011_2	$+3_{10}$
-4_{10}	100_2	—	

- Has one zero but more negative than positive numbers.

Representing Numbers in Binary: (Cont'd)
Two's Complement Representation (Cont'd)

Negative		Positive	
—		0000_2	$+0_{10}$
-1_{10}	1111_2	0001_2	$+1_{10}$
-2_{10}	1110_2	0010_2	$+2_{10}$
-3_{10}	1101_2	0011_2	$+3_{10}$
-4_{10}	1100_2	0100_2	$+4_{10}$
-5_{10}	1011_2	0101_2	$+5_{10}$
-6_{10}	1010_2	0110_2	$+6_{10}$
-7_{10}	1001_2	0111_2	$+7_{10}$
-8_{10}	1000_2	—	

Representing Numbers in Binary: (Cont'd)
Two's Complement Representation (Cont'd)

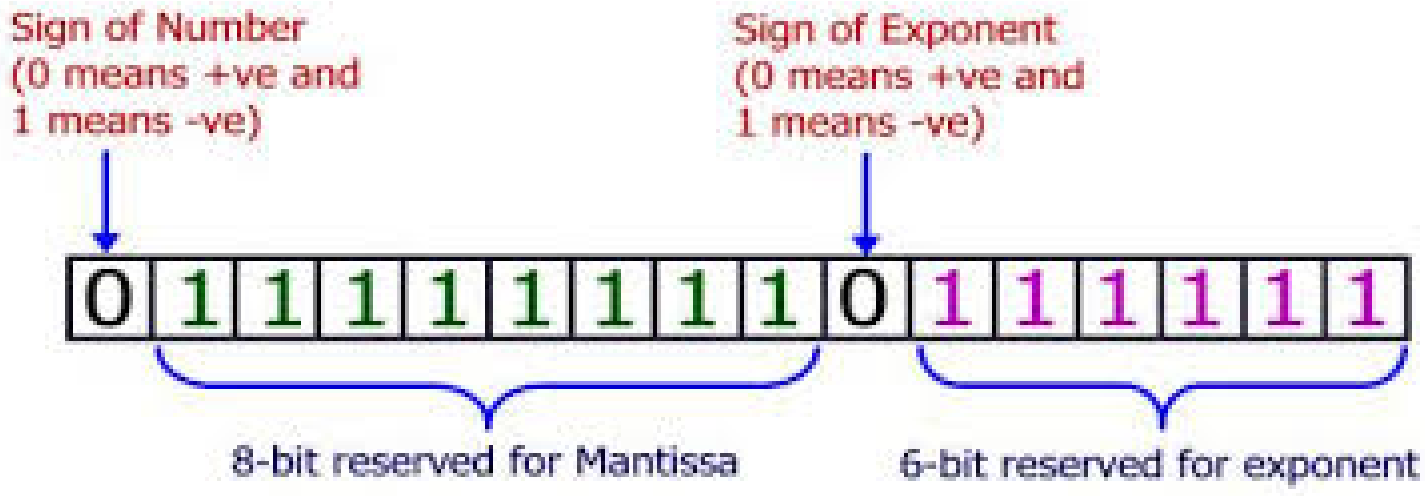
- Another way to get the negative version of a number x in two's complement is to toggle all the bits of the positive binary version of that number and add 1, e.g.,

	0	1	1	1	7_{10}
		$\Downarrow$			
	1	0	0	0	7_{10} (toggled)
+	0	0	0	1	Add 1
	1	0	0	1	-7_{10}

- Note that in two's complement representation (unlike sign magnitude), when you add x and $-x$, you get zero.

Representing Numbers in Binary: (Cont'd) Fractional Numbers

- Store in scientific notation ($M \times B^E$) where base $B = 2$, mantissas (M) and exponents (E) are stored in signed binary notation, and mantissas are normalized, e.g.,



Representing Numbers in Binary: (Cont'd)
Fractional Numbers (Cont'd)

Example: Representing 5.75_{10} as a fractional number

- $5_{10} = 4 + 1 = 2^2 + 2^0 = 101_2$ and
  $.75_{10} = 1/2 + 1/4 = 2^{-1} + 2^{-2} = .11_2$;
  Therefore, $5.75_{10} = 101.11_2$
- Normalize 101.11_2, e.g.,

$$\begin{aligned} 101.11_2 &= 101.11_2 \times 2^0 \\ &= 10.111_2 \times 2^1 \\ &= 1.0111_2 \times 2^2 \\ &= .10111_2 \times 2^3 \end{aligned}$$

- Encode $M = +.10111_2$ and $E = +3_{10} = +11_2$, i.e.,

0 + 00010111 + 0 + 000011

Representing Numbers in Binary: (Cont'd)
More Compact Representations

Long binary numbers represented more compactly in hexadecimal (16 digit; $0, 1, \ldots, 9, A(10), B(11), C(12), D(13), E(14), F(15)$), e.g.,

				A (10)				0				$\Rightarrow A0_{16}$
1	0	1	0		0	0	0		0_2			
	F (15)				C (12)				9			$\Rightarrow FC9_{16}$
1	1	1	1	1	1	0	0	1	0	0	1_2	
		2				9			D (13)			$\Rightarrow 29D_{16}$
[0	0]	1	0	1	0	0	1	1	1	0	1_2	

Representing Numbers in Binary: (Cont'd)
More Compact Representations (Cont'd)

To convert a hexadecimal number to its decimal equivalent, add up the powers of sixteen multiplied by non-zero digits, e.g.,

$$\begin{aligned} A9_{16} &= (10 \times 16^1) + (9 \times 16^0) \\ &= 160 + 9 \\ &= 169_{10} \end{aligned}$$

$$\begin{aligned} F4B_{16} &= (15 \times 16^2) + (4 \times 16^1) + (11 \times 16^0) \\ &= 3840 + 64 + 11 \\ &= 3915_{10} \end{aligned}$$

Representing Numbers in Binary: (Cont'd)
More Compact Representations (Cont'd)

To convert a decimal number to its hexadecimal equivalent, repeatedly divide by sixteen and read the remainder digits in reverse (from last to first), e.g.,

	Quotient	Remainder	
$169/16 \Rightarrow$	10	9	
$10/16 \Rightarrow$	0	10	$\Longrightarrow A9_{16}$

	Quotient	Remainder	
$3915/16 \Rightarrow$	244	11	
$244/16 \Rightarrow$	15	4	
$15/16 \Rightarrow$	0	15	$\Longrightarrow F4B_{16}$

Binary Computer Memory

What does

01000001010110001110001111

mean?

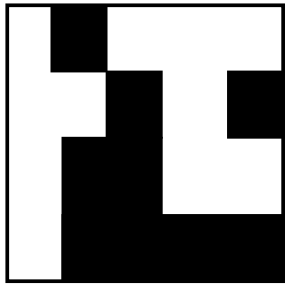
Binary Computer Memory (Cont'd)

Things that binary computer memory does well:

Sounds

01000#00101#01100#01100#01111

Pictures



Numbers

8, 5, 12, 12, 15

Text

"HELLO"

Representing Text in Binary

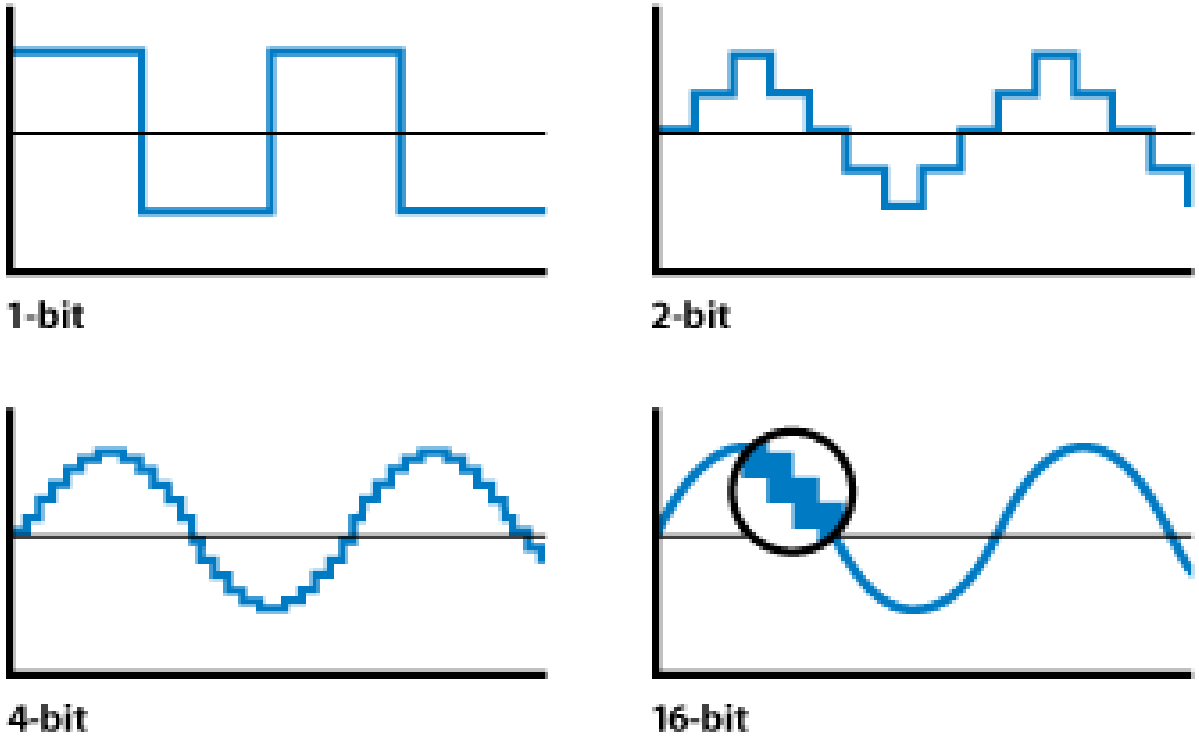
- Associate individual symbols with unsigned binary numbers; these symbols may not be printable but rather instructions to I/O interface devices, e.g., control characters.
- Original ASCII and EBCDIC standards used 7 and 8 bits to represent 128 and 256 symbols (see Table 4.3 in textbook for part of ASCII standard).
- Original UNICODE standard used 16 bits to represent $\approx$ 65,000 symbols; embedded original ASCII standard in lowest 128 codes.
- UNICODE subsequently extended to 32 bits; can accommodate $\approx$ two billion symbols.

Representing Text in Binary: ASCII

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

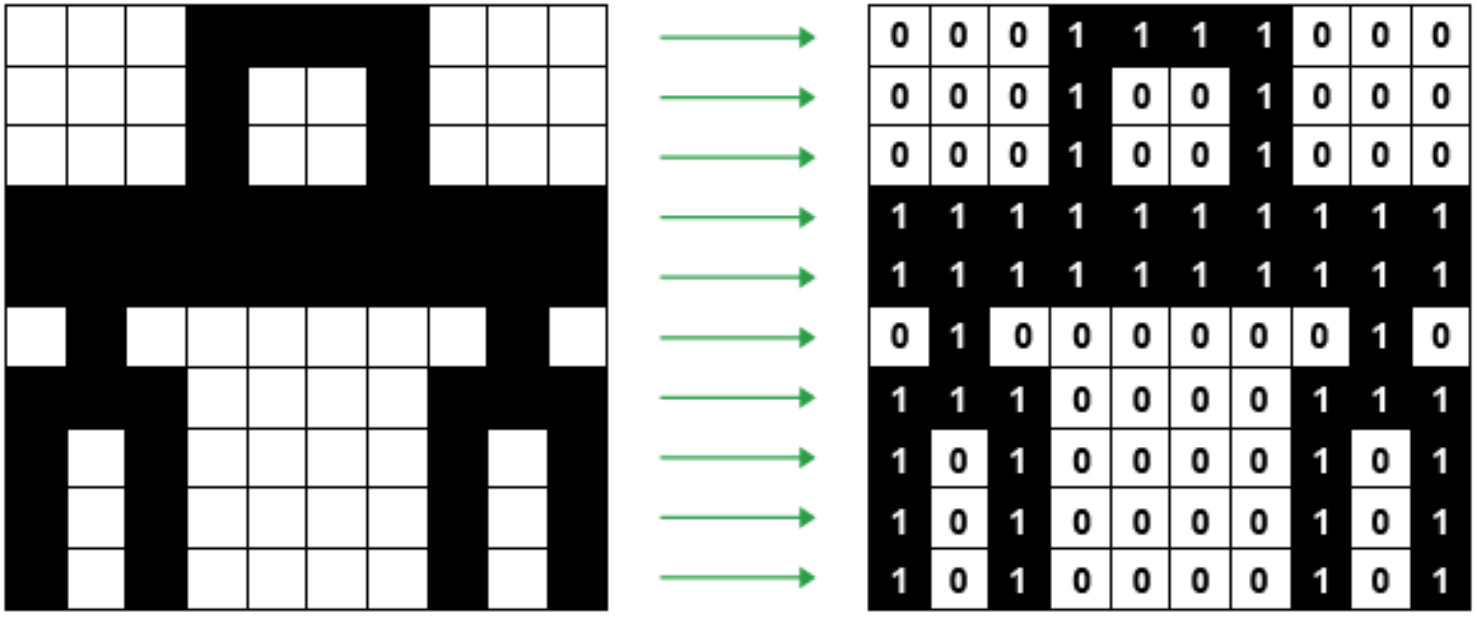
Representing Sound in Binary

- Encode periodic audio signal samples as integers.
- 40,000 samples/sec suffices for human hearing.



Representing Pictures in Binary

- Reduce picture to grid of picture elements (**pixels**).
- Encode pixel values as one or more integers, e.g., single bits (B/W (see below)), 8-value gray scale (B/W), triplets of 256-value red / green / blue intensities (color).



Binary Computer Memory Redux

Things that binary computer memory doesn't do so well:

Arbitrary Fractional Numbers

$1/3 \longrightarrow 0.33$

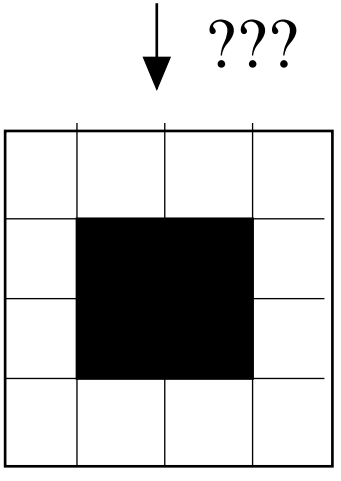
$1/3 \longrightarrow 0.33$

$+\ 1/3 \longrightarrow 0.33$

$1 \neq 0.99$

Detailed Pictures

Earth



. . . And If You Liked This . . .

- MUN Computer Science courses on this area:
  - COMP 2003: Computer Architecture
  - COMP 3731: Introduction to Scientific Computing
  - COMP 4734: Matrix Computations and Applications
- MUN Computer Science professors teaching courses / doing research in in this area:
  - Sharene Bungay
  - Vinicius Prado da Fonseca
  - George Miminis