

CSC 121 Computers and Scientific Thinking

Fall 2005

Data Representation

1

Analog vs. Digital



there are two ways data can be stored electronically

1. *analog* signals represent data in a way that is analogous to real life
 - signals can vary continuously across an infinite range of values
 - e.g., frequencies on an old-fashioned radio with a dial
2. *digital* signals utilize only a finite set of values
 - e.g., frequencies on a modern radio with digital display

the major tradeoff between analog and digital is variability vs. reproducibility

- analog allows for a (potentially) infinite number of unique signals, but they are harder to reproduce
 - good for storing data that is highly variable but does not need to be reproduced exactly
- digital signals limit the number of representable signals, but they are easily remembered and reproduced
 - good for storing data when reproducibility is paramount

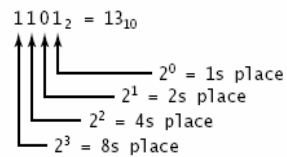
2

Binary Numbers

- modern computers save and manipulate data as discrete (digital) values
- the most effective systems use two distinct binary states for data representation
 - in essence, all data is stored as *binary numbers*

in the binary number system, all values are represented using only the two binary digits 0 and 1, which are called *bits*

binary representation



converting binary to decimal

Binary Number	Decimal Equivalent
$11_2 \rightarrow$	$1 \cdot 2 + 1 \cdot 1 = 3$
$1101_2 \rightarrow$	$1 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 13$
$10011_2 \rightarrow$	$1 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 1 \cdot 1 = 19$
$100110_2 \rightarrow$	$1 \cdot 32 + 0 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = 38$

3

Decimal \rightarrow Binary

algorithm for converting from decimal (D) to binary (B):

- Initialize B to be empty.
- As long as D is nonzero, repeatedly:
 - If D is even, add 0 to the left of B.
 - If D is odd, add 1 to the left of B.
 - Divide D by 2 and round down (i.e., $D = \text{Math.floor}(D/2)$)

Converting 19 to binary:

19 is odd \rightarrow B = 1, D = $\text{Math.floor}(19/2) = 9$
 9 is odd \rightarrow B = 11, D = $\text{Math.floor}(9/2) = 4$
 4 is even \rightarrow B = 011, D = $\text{Math.floor}(4/2) = 2$
 2 is even \rightarrow B = 0011, D = $\text{Math.floor}(2/2) = 1$
 1 is odd \rightarrow B = 10011, D = $\text{Math.floor}(1/2) = 0$

Converting 116 to binary:

116 is even \rightarrow B = 0, D = $\text{Math.floor}(116/2) = 58$
 58 is even \rightarrow B = 00, D = $\text{Math.floor}(58/2) = 29$
 29 is odd \rightarrow B = 100, D = $\text{Math.floor}(29/2) = 14$
 14 is even \rightarrow B = 0100, D = $\text{Math.floor}(14/2) = 7$
 7 is odd \rightarrow B = 10100, D = $\text{Math.floor}(7/2) = 3$
 3 is odd \rightarrow B = 110100, D = $\text{Math.floor}(3/2) = 1$
 1 is odd \rightarrow B = 1110100, D = $\text{Math.floor}(1/2) = 0$

4

Representing Integers

when an integer value must be saved on a computer, its binary equivalent can be encoded as a bit pattern and stored digitally

usually, a fixed size (e.g., 32 bits) is used for each integer so that the computer knows where one integer ends and another begins

- the initial bit in each pattern acts as the *sign bit* (0=positive, 1=negative)
- negative numbers are represented in *two's complement notation*
 - the "largest" bit pattern corresponds to the smallest absolute value (-1)

Bit Pattern	Decimal Value
10000000000000000000000000000000	$(-2^{31} = -2,147,483,648)$
10000000000000000000000000000001	$(-2^{31}-1 = -2,147,483,647)$
10000000000000000000000000000010	$(-2^{31}-2 = -2,147,483,646)$
.	
.	
1111111111111111111111111111111101	(-3)
1111111111111111111111111111111110	(-2)
1111111111111111111111111111111111	(-1)
00000000000000000000000000000000	(0)
00000000000000000000000000000001	(1)
00000000000000000000000000000010	(2)
00000000000000000000000000000011	(3)
.	
.	
0111111111111111111111111111111101	$(2^{31}-3 = 2,147,483,645)$
0111111111111111111111111111111110	$(2^{31}-2 = 2,147,483,646)$
0111111111111111111111111111111111	$(2^{31}-1 = 2,147,483,647)$

5

Representing Real Numbers

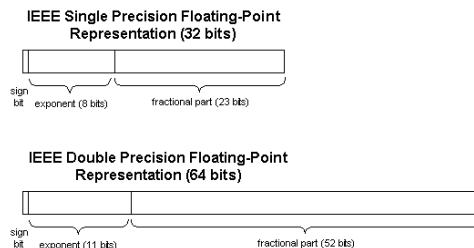
a real number can be uniquely identified by the two components of its scientific notation (fractional part and the exponent)

$$123.45 = 12345 \times 10^{-2} \quad .000042 = 42 \times 10^{-6}$$

thus, any real number can be stored as a pair of integers

- real numbers stored in this format are known as *floating point numbers*, since the decimal point moves (floats) to normalize the fraction

standard formats exist for storing real numbers, using either 32 bits (single precision) or 64 bits (double precision)



most programming languages represent integers and reals differently
JavaScript simplifies things by using IEEE double-precision floating point for all numbers

6

Representing Characters

characters have no natural correspondence to binary numbers

- computer scientists devised an arbitrary system for representing characters as bit patterns
- ASCII (American Standard Code for Information Interchange)
 - maps each character to a specific 8-bit pattern
 - note that all digits are contiguous, as are all lower-case and all upper-case letters

'0' < '1' < ... < '9'
 'A' < 'B' < ... < 'Z'
 'a' < 'b' < ... < 'z'

code	char	code	char	code	char
00100000	space	01000000	@	01100000	'
00100001	!	01000001	A	01100001	a
00100010	"	01000010	B	01100010	b
00100011	#	01000011	C	01100011	c
00100100	\$	01000100	D	01100100	d
00100101	%	01000101	E	01100101	e
00100110	&	01000110	F	01100110	f
00100111	'	01000111	G	01100111	g
00101000	(01001000	H	01101000	h
00101001)	01001001	I	01101001	i
00101010	*	01001010	J	01101010	j
00101011	+	01001011	K	01101011	k
00101100	,	01001100	L	01101100	l
00101101	-	01001101	M	01101101	m
00101110	.	01001110	N	01101110	n
00101111	/	01001111	O	01101111	o
00110000	0	01010000	P	01110000	p
00110001	1	01010001	Q	01110001	q
00110010	2	01010010	R	01110010	r
00110011	3	01010011	S	01110011	s
00110100	4	01010100	T	01110100	t
00110101	5	01010101	U	01110101	u
00110110	6	01010110	V	01110110	v
00110111	7	01010111	W	01110111	w
00111000	8	01011000	X	01111000	x
00111001	9	01011001	Y	01111001	y
00111010	:	01011010	Z	01111010	z
00111011	;	01011011	[01111011	{
00111100	<	01011100	\	01111100	
00111101	=	01011101]	01111101	}
00111110	>	01011110	^	01111110	~
00111111	?	01011111		01111111	delete

7

Representing Text

strings can be represented as sequences of ASCII codes, one for each character in the string



specific programs may store additional information along with the ASCII codes

- e.g. programming languages will often store the number of characters along with the ASCII codes
- e.g., word processing programs will insert special character symbols to denote formatting (analogous to HTML tags in a Web page)

8

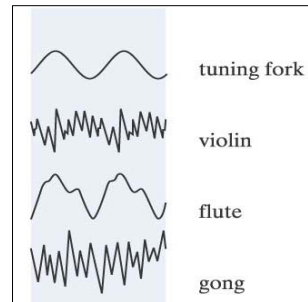
Representing Sounds

computers are capable of representing much more than numbers and text

- complex data requires additional techniques and algorithms

EXAMPLE: representing sounds

- sounds are inherently analog signals with a specific amplitudes and frequencies
- when sound waves reach your ear, they cause your eardrum to vibrate, and your brain interprets the vibration as sound
- e.g. telephones translate a waveform into electrical signals, which are then sent over a wire and converted back to sound
- e.g. phonographs interpret waveforms stored on on grooves of a disk (similar to audio cassettes)
- analog signals cannot be reproduced exactly, but this is not usually a problem since the human ear is unlikely to notice small inconsistencies



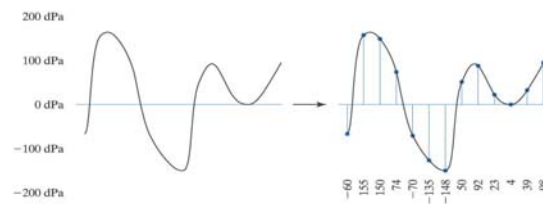
9

Representing Sounds (cont.)

when analog recordings are repeatedly duplicated, small errors that were originally unnoticed begin to propagate

digital recordings can be reproduced exactly without any deterioration in sound quality

- analog waveforms must be converted to a sequence of discrete values
- *digital sampling* is the process in which the amplitude of a wave is measured at regular intervals, and stored as discrete measurements



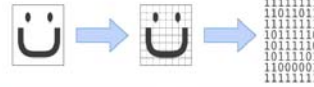
- frequent measurements must be taken to ensure high quality (e.g., 44,100 readings per second for a CD)
 - this results in massive amounts of storage
 - techniques are used to compress the data and reduce file sizes (e.g., MP3, WAV)

10

Representing Images

EXAMPLE: representing images

- images are stored using a variety of formats and compression techniques
- the simplest representation is a *bitmap*
- bitmaps partition an image into a grid of picture elements, called *pixels*, and then convert each pixel into a bit pattern



1. To generate a bitmap, the image is first partitioned into a grid of pixels, here 8×8 .

2. For a black-and-white bitmap, a black pixel is represented with a 0 bit, a white pixel with a 1 bit.

resolution refers to the sharpness or clarity of an image

- bitmaps that are divided into smaller pixels will yield higher resolution images
- the left image is stored using 96 pixels per square inch, and the right image is stored using 48 pixels per square inch
 - the left image appears sharp, but has twice the storage requirements



11

Representing Images (cont.)

when creating a bitmap of a color image, more than one bit is required to represent each pixel

- the most common system is to translate each pixel into a 24 bit code, known as its *RGB value*: 8 bits to represent the intensity of each red/green/blue component

Common HTML colors					
color	(R, G, B)	color	(R, G, B)	color	(R, G, B)
red	(255, 0, 0)	green	(0, 128, 0)	blue	(0, 0, 255)
darkred	(139, 0, 0)	darkgreen	(0, 100, 0)	darkblue	(0, 0, 139)
maroon	(128, 0, 0)	forestgreen	(34, 139, 34)	royalblue	(65, 105, 225)
crimson	(220, 20, 60)	olive	(128, 128, 0)	lightblue	(173, 216, 230)
pink	(255, 192, 203)	lightgreen	(144, 238, 144)	purple	(128, 0, 128)
violet	(238, 130, 238)	brown	(165, 42, 42)	gray	(128, 128, 128)
orange	(255, 165, 0)	white	(255, 255, 255)	black	(0, 0, 0)

common image formats implement various compression techniques to reduce storage size

- GIF (Graphics Interchange Format)
 - a *lossless* format, meaning no information is lost in the compression
 - commonly used for precise pictures, such as line drawings
- JPEG (Joint Photographic Experts Group)
 - a *lossy* format, so the compression is not fully reversible (but more efficient)
 - commonly used for photographs

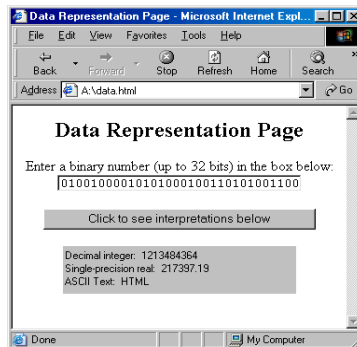
12

Disti ngui shi ng Data Types



how does a computer know what type of value is stored in a particular piece of memory?

- short answer: it doesn't
- when a program stores data in memory, it must store additional information as to what type of data the bit pattern represents
- thus, the same bit pattern might represent different values in different contexts



13