Data Storage

Introduction to Computer

Yu-Ting Wu

(with some slides borrowed from Prof. Tian-Li Yu)

1

Outline

- Bits and their storage
- Main memory
- Mass storage
- Representing information as bit patterns
- The binary system
- Data and compression
- Communication errors

2

Outline

- Bits and their storage
- Main memory
- Mass storage
- Representing information as bit patterns
- The binary system
- Data and compression
- Communication errors

3

Binary World

- Digital data is **represented** and **stored** in **binary form**
- **Bit:** a binary digit (0 or 1)
  - Bit patterns are used to represent information, such as numbers, text characters, images, sound, … etc.
- Why binary?
  - Simple
  - Logical (0 means false and 1 means true)
  - Unambiguous

4

Boolean Operations and Gates

- **Boolean Operation**
  - An operation that manipulates one or more true/false values
  - AND, OR, XOR (exclusive or), NOT
- **Gate**
  - A device that computes a Boolean operation
  - Often implemented as small electronic circuits called transistors
  - Provide the **building blocks** from which computers are constructed

Boolean Operations and Gates (cont.)

AND gate

Input A B  Z Output

Input		Output
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table

OR gate

Input A B  Z Output

Input		Output
0	0	0
0	1	1
1	0	1
1	1	1

Boolean Operations and Gates (cont.)

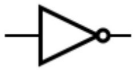
XOR (eXclusive OR) gate

Input A B  Z Output

Input		Output
0	0	0
0	1	1
1	0	1
1	1	0

Truth Table

NOT gate

Input A  Z Output

Input	Output
0	1
1	0

Flip-flops

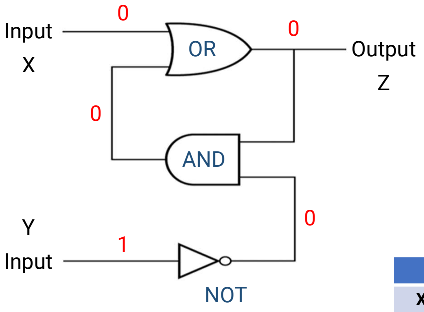
- Circuits built from gates that act as a fundamental unit of **computer memory**
  - Keep the state of output until the next excitement
- Spec: two inputs
  - One input for storing a value to 0
  - The other input for storing a value to 1
  - While both two inputs are not set (0), keep the **most recently** stored value

Input X Y 

Z Output

Input		Output
X	Y	Z
0	0	Unchanged
0	1	0
1	0	1
1	1	undefined

A Simple Flip-flop Circuit

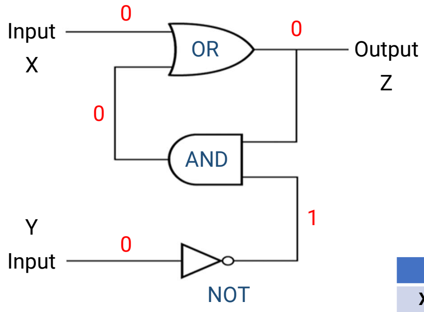


Input (X/Y)
(0/1)
0

Input		Output
X	Y	Z
0	0	Unchanged
0	1	0
1	0	1
1	1	undefined

9

A Simple Flip-flop Circuit

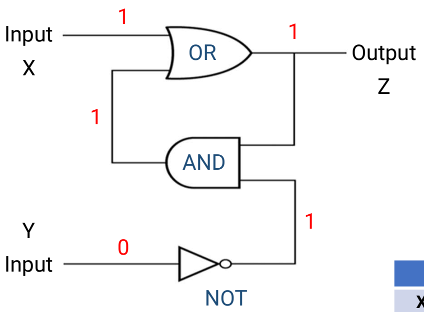


Input (X/Y)
(0/1) → (0/0)
0 0

Input		Output
X	Y	Z
0	0	Unchanged
0	1	0
1	0	1
1	1	undefined

10

A Simple Flip-flop Circuit

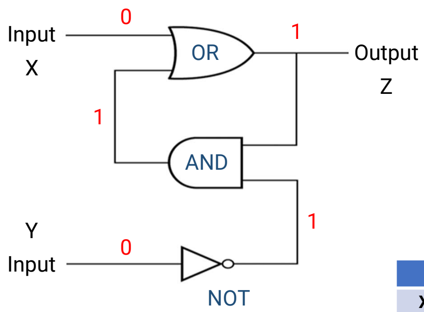


Input (X/Y)
(0/1) → (0/0) → (1/0)
0 0 1

Input		Output
X	Y	Z
0	0	Unchanged
0	1	0
1	0	1
1	1	undefined

11

A Simple Flip-flop Circuit

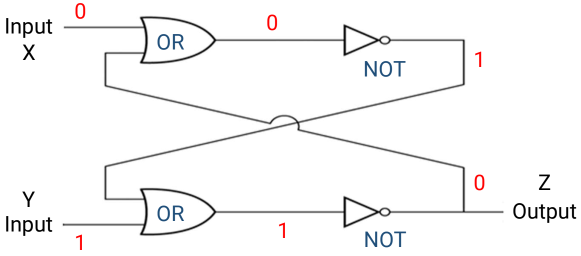


Input (X/Y)
(0/1) → (0/0) → (1/0) → (0/0)
0 0 1 1

Input		Output
X	Y	Z
0	0	Unchanged
0	1	0
1	0	1
1	1	undefined

12

Another Flip-flop circuit

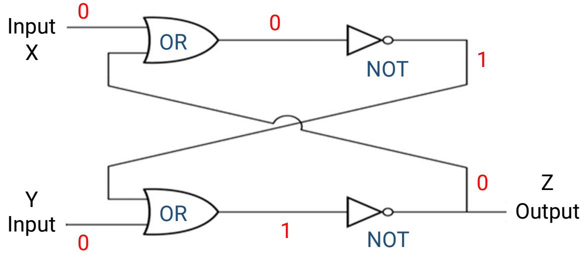


Input		Output
X	Y	Z
0	0	Unchanged
0	1	0
1	0	1
1	1	undefined

Input (X/Y)
(0/1)
0

13

Another Flip-flop circuit

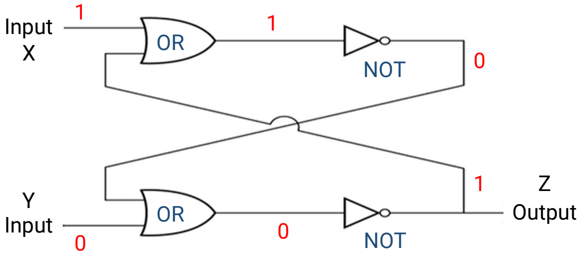


Input		Output
X	Y	Z
0	0	Unchanged
0	1	0
1	0	1
1	1	undefined

Input (X/Y)
(0/1) ➔ (0/0)
0 0

14

Another Flip-flop circuit

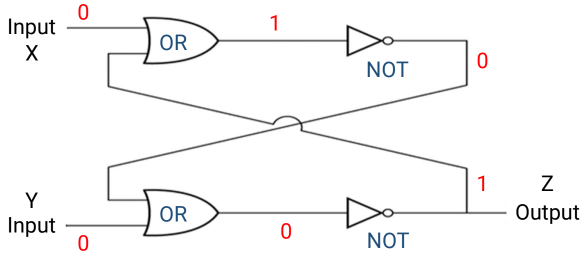


Input		Output
X	Y	Z
0	0	Unchanged
0	1	0
1	0	1
1	1	undefined

Input (X/Y)
(0/1) ➔ (0/0) ➔ (1/0)
0 0 1

15

Another Flip-flop circuit



Input		Output
X	Y	Z
0	0	Unchanged
0	1	0
1	0	1
1	1	undefined

Input (X/Y)
(0/1) ➔ (0/0) ➔ (1/0) ➔ (0/0)
0 0 1 1

16

Outline

- Bits and their storage
- Main memory
- Mass storage
- Representing information as bit patterns
- The binary system
- Data and compression
- Communication errors

Hexadecimal Notation

- A **shorthand notation** for long bit patterns
  - Divide a pattern into groups of **four** bits each
  - Represent each group by a single symbol
- Examples:
  - 10110101 ➔ 0xB5
  - 00011111 ➔ 0x1F
  - 11110000 ➔ 0xF0

Bit pattern	Hexadecimal representation
0000	0x0
0001	0x1
0010	0x2
0011	0x3
0100	0x4
0101	0x5
0110	0x6
0111	0x7
1000	0x8
1001	0x9
1010	0xA
1011	0xB
1100	0xC
1101	0xD
1110	0xE
1111	0xF
10000	0x10

Main Memory

- **Cell**: a unit of main memory (typically 8 bits called 1 **byte**)
  - Most significant bit: the bit at the left (high-order) end
  - Least significant bit: the bit at the right (low-order) end

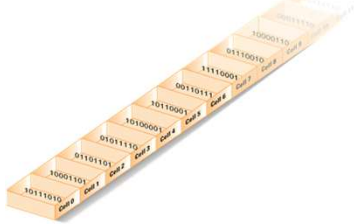
High-order end 0 1 0 1 1 0 1 0 Low-order end

Most significant bit

Least significant bit

Main Memory Addresses

- Address: a **name** that **uniquely identifies one cell** in the computer's main memory
  - The names are actually **numbers**
    - These numbers are assigned consecutively starting at zero
  - Numbering the cells in this manner associates an order with the memory cells



Classification of Main Memory

- **Random Access Memory (RAM)**
  - Memory in which individual cells can be easily accessed **in any order**
- Classification
  - Static memory (SRAM), like flip-flop
  - Dynamic memory (DRAM)
    - RAM composed of volatile memory
  - Synchronous DRAM (SDRAM)
  - Double Data Rate (DDR)
  - Dual/Triple channel

21

Memory Capacity

- **Kilobyte**: 2^{10} bytes = 1024 bytes
  - Example: 3 **KB** = 3 × 1024 bytes
- **Megabyte**: 2^{20} bytes = 1,048,576 bytes
  - Example: 3 **MB** = 3 × 1,048,576 bytes
- **Gigabyte**: 2^{30} bytes = 1,073,741,824 bytes
  - Example: 3 **GB** = 3 × 1,073,741,824 bytes

22

Outline

- Bits and their storage
- Main memory
- Mass storage
- Representing information as bit patterns
- The binary system
- Data and compression
- Communication errors

23

Mass Storage

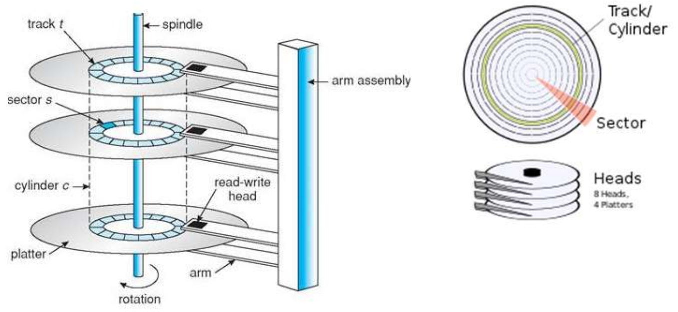
- Additional devices:
  - Magnetic disks
  - CDs
  - Flash drives (e.g., USB)
  - Magnetic tapes
  - DVDs
  - Solid-state drives
- Advantages over main memory
  - Less **volatility**
  - Larger storage capacities (?)
  - Low cost (but much slower)
  - In many cases can be removed

24

Mass Storage Performance

- **Bandwidth:** the total amount of bits that can be transferred in a unit of time
- **Latency:** the total time between the request for data transfer and its arrival

Magnetic Disk Storage System

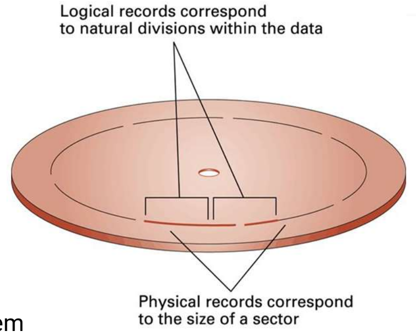


Access time = seek time + rotation delay (latency time)
Transfer rate (SATA 1.5/3/6, etc.)

Magnetic Disk Storage System (cont.)

- **Buffer**
  - To **synchronize** different R/W mechanisms and rates
    - Disk I/O is very slow compared to CPU and memory
  - Buffer is a memory area used for the temporary storage of data
    - Blocks of data compatible with physical records can be transferred between buffers and the mass storage system
    - Data in the buffer can be referenced in terms of logical records

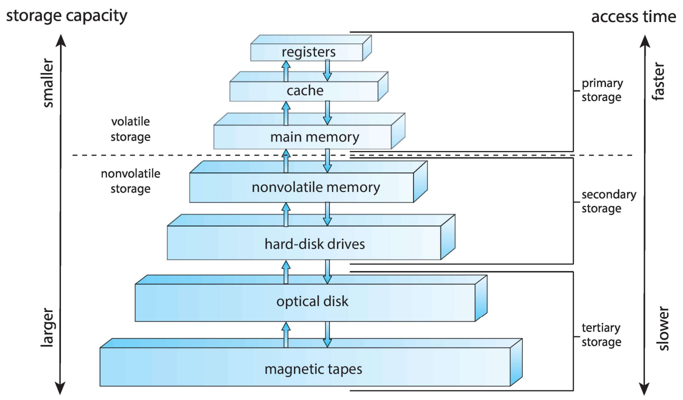
Magnetic Disk Storage System (cont.)



- File and file systems
- Fragmentation problem

We will talk about this later in OS

Storage Structure



Storage Structure (cont.)

Level	1	2	3	4	5
Name	registers	cache	main memory	solid-state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25-0.5	0.5-25	80-250	25,000-50,000	5,000,000
Bandwidth (MB/sec)	20,000-100,000	5,000-10,000	1,000-5,000	500	20-150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Flash Drives

- Flash memory
  - Circuits that trap electrons in tiny silicon dioxide chambers
  - Repeated erasing slowly damages the media
  - SD cards provide GBs of storage
- Commonly used for
  - Digital cameras
  - Smartphones

Outline

- Bits and their storage
- Main memory
- Mass storage
- Representing information as bit patterns
- The binary system
- Data and compression
- Communication errors

Data Representation

- Many different kinds of information can be encoded as **bit patterns**
- Systems for encoding information have been established for
  - Text
  - Numeric Data
  - Images
  - Sound
  - Other data

Representing Text

- Each character (letter, punctuation, etc.) is assigned a unique bit pattern
  - **ASCII** uses patterns of 7-bits (or 8-bits with a leading 0) to represent most symbols used in written English text

1	▯	17	▯	33	!	49	1	65	A	81	Q	97	a	113	q
2		18	▯	34	"	50	2	66	B	82	R	98	b	114	r
3	▯	19	▯	35	#	51	3	67	C	83	S	99	c	115	s
4	▯	20	▯	36	$	52	4	68	D	84	T	100	d	116	t
5	▯	21	▯	37	%	53	5	69	E	85	U	101	e	117	u
6	▯	22	▯	38	&	54	6	70	F	86	V	102	f	118	v
7	▯	23	▯	39	'	55	7	71	G	87	W	103	g	119	w
8	▯	24	▯	40	(	56	8	72	H	88	X	104	h	120	x
9		25	▯	41	)	57	9	73	I	89	Y	105	i	121	y
10		26	▯	42	*	58	:	74	J	90	Z	106	j	122	z
11	▯	27	▯	43	+	59	;	75	K	91	[	107	k	123	{
12	▯	28		44	,	60	<	76	L	92	\	108	l	124	|
13		29		45	-	61	=	77	M	93	]	109	m	125	}
14	▯	30		46	.	62	>	78	N	94	^	110	n	126	~
15	▯	31		47	/	63	?	79	O	95	_	111	o	127	▯
16	▯	32		48	0	64	@	80	P	96	`	112	p	128	€

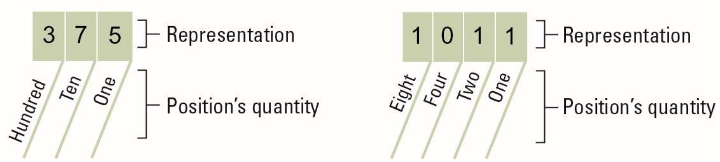
01001000	01100101	01101100	01101100	01101111	00101110
H	e	l	l	o	.

Representing Text

- Each character (letter, punctuation, etc.) is assigned a unique bit pattern
  - **ASCII** uses patterns of 7-bits (or 8-bits with a leading 0) to represent most symbols used in written English text
  - **ISO** developed a number of extensions to ASCII, each designed to accommodate a major language group
    - E.g., Western European language: ä, ö, and ü
  - **Unicode** uses patterns up to 21-bits to represent the symbols used in languages worldwide, 16-bits for the world's commonly used languages

Representing Numeric Values

- Binary notation: uses bits to represent a number in **base two**
  - All numeric values in a computer are stored in sequences of 0s and 1s

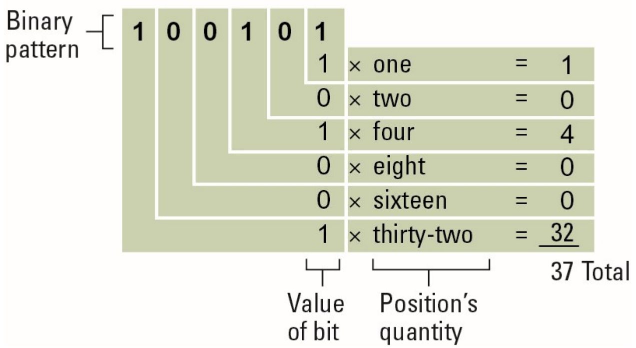


Base 10 system

Base 2 system

The Binary System

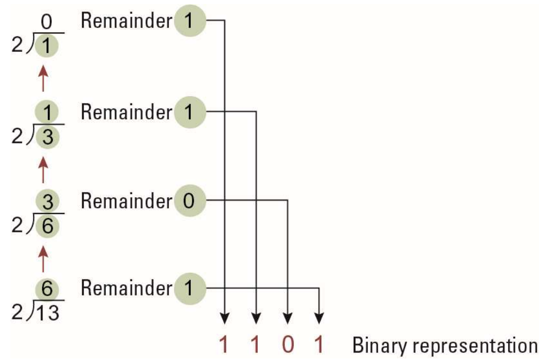
- Decode the binary representation



37

The Binary System (cont.)

- Algorithm for translation from Base 10 system to Base 2 system



38

The Binary System (cont.)

- Algorithm for translation from Base 10 system to Base 2 system

Step 1. Divide the value by two and record the remainder.

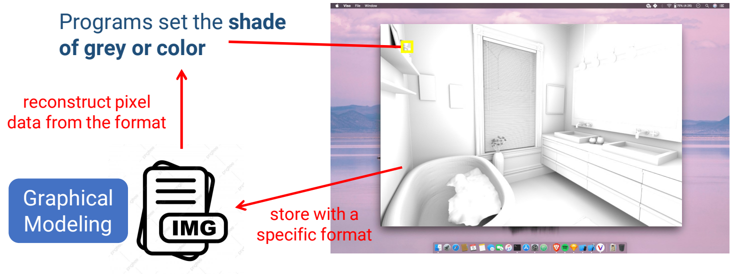
Step 2. As long as the quotient obtained is not zero, continue to divide the newest quotient by two and record the remainder.

Step 3. Now that a quotient of zero has been obtained, the binary representation of the original value consists of the remainders listed from right to left in the order they were recorded.

39

Representing Images

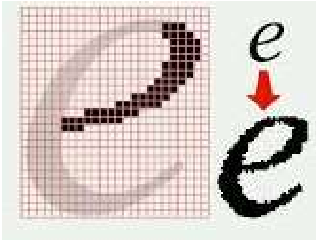
- Monitor display pictures as a **rectangular array of pixels** (small, usually square, dots of color)
  - Merge optically when viewed at a suitable distance to produce the impression of continuous tones



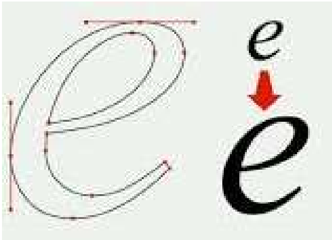
40

Representing Images (cont.)

- Two approaches for graphical modeling



bitmapped images

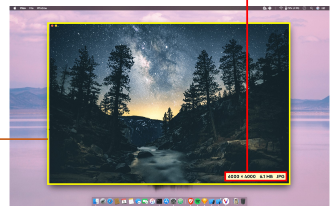


vector graphics

41

Representing Images (cont.)

- **Bitmap techniques**
  - Logical pixels: stored value in an image file
  - Physical pixels: physical dots in an image file
  - RGB for color: red, green, and blue components (alternatives: CMYK, HSV, YUV … etc.)



42

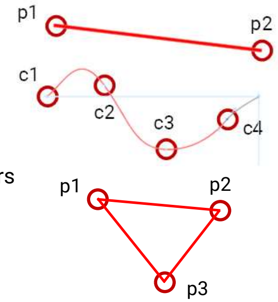
Representing Images (cont.)

- Bitmap image examples

43

Representing Images (cont.)

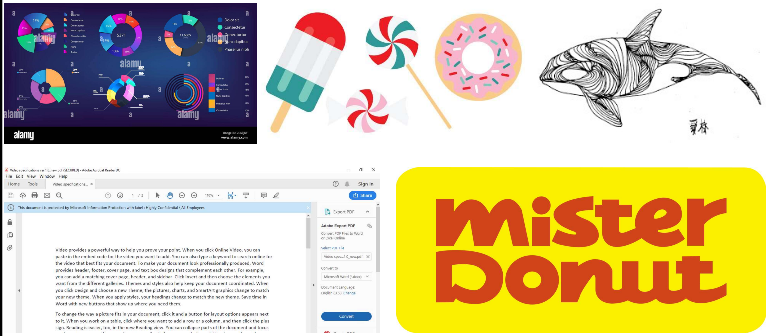
- **Vector techniques**
  - An image is modelled by the mathematical description of a collection of individual objects making up the image
  - **Lines**
    - End points
  - **Curves**
    - Control points
  - **Shapes**
    - Shape-dependent parameters



44

Representing Images (cont.)

- Vector graphics examples

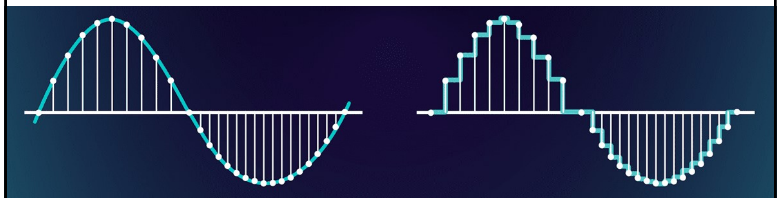


Representing Images (cont.)

- Bitmapped v.s. Vector graphics
  - Bitmapped images provide better control of pixel values, thus being more suitable for natural images
  - Vector graphics are **resolution independent**, thus being more suitable for texts and icons

Representing Sound

- **Sampling** techniques that record actual audio
  - Sampling rate
    - Long-distance telephone: 8000 samples/sec.
    - CD sound: 44,100 samples/sec.
  - Bit resolution
  - Bit rate = (sampling rate) × (bit resolution)



Representing Sound (cont.)

- MIDI (synthesis)
  - Stores directions for making sound
  - Encodes which instrument, note, and duration

Aeris's theme (FF7) Tifa's theme (FF7)

Outline

- Bits and their storage
- Main memory
- Mass storage
- Representing information as bit patterns
- The binary system
- Data and compression
- Communication errors

The Binary System Revisit

- **Addition**

$$\begin{array}{r} 0 \\ +0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ +0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0 \\ +1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +1 \\ \hline 10 \end{array}$$

The Binary System Revisit (cont.)

- **Subtraction**
  - Subtraction can be treated as adding a negative number
  - Need to define negative numbers first
- Two ways for representing a negative number
  - **Two's complement** (more popular)
  - **Excess notation**

Two's Complement

- Assume 3 bit
  - Positive numbers only: 0 ~ 7
  - Positive and negative numbers: -4 ~ +3

0 1 1	3	3
0 1 0	2	2
0 0 1	1	1
0 0 0	0	0
1 1 1	-1	7
1 1 0	-2	6
1 0 1	-3	5
1 0 0	-4	4

Example: 1 - 1

= 1 + (-1)

$$\begin{array}{r} 0\ 0\ 1 \\ +\ 1\ 1\ 1 \\ \hline \cancel{1}\ 0\ 0\ 0 \end{array}$$

Two’s Complement (cont.)

Problem in base 10	Problem in two's complement	Answer in base 10
3 + 2	0011 + 0010 = 0101	5
−3 + −2	1101 + 1110 = 1011	−5
7 + −5	0111 + 1011 = 0010	2

Bit pattern	Value represented
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

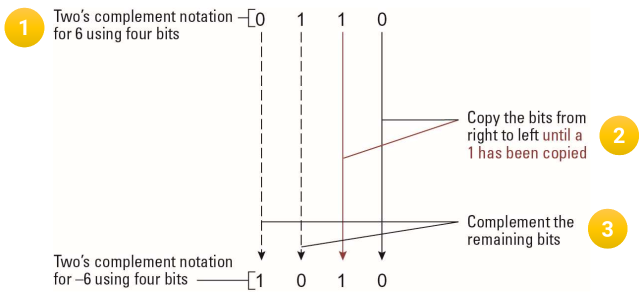
Two’s Complement Encoding

- Represent a negative number with two’s complement
- **Approach 1:**
  - Example: -2

	3-bit	4-bit
1. Consider +2	0 1 0	0 0 1 0
2. Copy the numbers from right to left until the first “1”	1 0	1 0
3. Flip the rest numbers	1 1 0	1 1 1 0

Two’s Complement Encoding (cont.)

- Represent a negative number with two’s complement
- **Approach 1:**
  - Example: -6



Two’s Complement Encoding (cont.)

- Represent a negative number with two’s complement
- **Approach 2:**
  - Example: -2 (3-bit)

0 1 1	3	3
0 1 0	2	2
0 0 1	1	1
0 0 0	0	0
1 1 1	-1	7
$\boxed{1\ 1\ 0}$	$\boxed{-2}$	$\boxed{6}$
1 0 1	-3	5
1 0 0	-4	4

1. Add 2^n for n-bit representation

$-2 + 2^3 = 6$

2. Write down its binary representation

1 1 0

Excess Notation

- Assume 3 bit
  - Positive numbers only: 0 ~ 7
  - Positive and negative numbers: -4 ~ +3

0 0 0	0	-4
0 0 1	1	-3
0 1 0	2	-2
0 1 1	3	-1
$\boxed{1\ 0\ 0}$	$\boxed{4}$	$\boxed{0}$
1 0 1	5	1
1 1 0	6	2
1 1 1	7	3

Advantages
A larger (smaller) number remains larger (smaller)

Excess Notation Encoding

- Represent a negative number with two's complement

1. Add 2^{n-1} for n-bit representation
2. Write down its binary representation

Examples

$2 \xrightarrow{+2^{(3-1)}} 6 \rightarrow 110$

$-3 \rightarrow 1 \rightarrow 001$

0 0 0	0	-4
0 0 1	1	-3
0 1 0	2	-2
0 1 1	3	-1
$\boxed{1\ 0\ 0}$	$\boxed{4}$	$\boxed{0}$
1 0 1	5	1
1 1 0	6	2
1 1 1	7	3

Addition using Excess Notation

0 0 0	0	-4
0 0 1	1	-3
0 1 0	2	-2
0 1 1	3	-1
$\boxed{1\ 0\ 0}$	$\boxed{4}$	$\boxed{0}$
1 0 1	5	1
1 1 0	6	2
1 1 1	7	3

Example: -2 + 3 (3-bit)

$= (0\ 1\ 0)_{\text{in EX}} + (1\ 1\ 1)_{\text{in EX}}$

$= \cancel{1}(0\ 0\ 1)_{\text{in EX}} \;!=\; (1\ 0\ 1)_{\text{in EX}}$

$= (-2 + 4)_{\text{in Bin}} + (3 + 4)_{\text{in Bin}}$

$= (-2 + 3 + \cancel{8} + 4)_{\text{in Bin}}$

Overflow

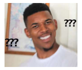
- There is a limit to the size of the values that can be represented in any system
- **Overflow**
  - Occurs when a computation produces a value that falls outside the range of values that can be represented in the machine
  - If the resulting sign bit is incorrect, an overflow has occurred

Overflow (cont.)

- Examples (using two's complement)

2 + 3 (3-bit)

  0 1 0
+ 0 1 1
  1 0 1



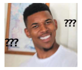
= -3 (???)

Bit pattern	Value represented
011	3
010	2
001	1
000	0
111	−1
110	−2
101	−3
100	−4

(-2) + (-3) (3-bit)

  1 1 0
+ 1 0 1
~~1~~ 0 1 1

= 3 (???)

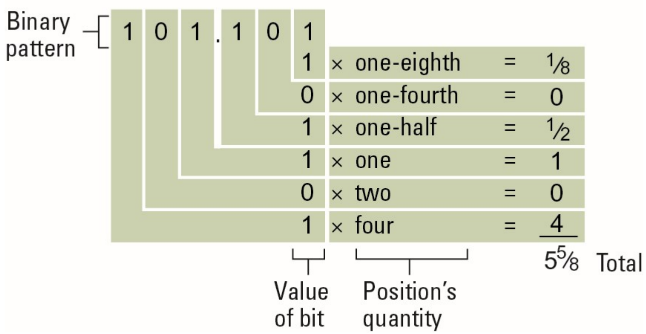


Solutions
- Use more bits (int ➔ long/long long)
- Use string to implement Big-Integer (by yourself)

61

Fraction

- **Fixed-point representation**



62

Fraction (cont.)

- Problems with fixed-point representation

The speed of light: 300000000

One mole: 600000000000000000000000

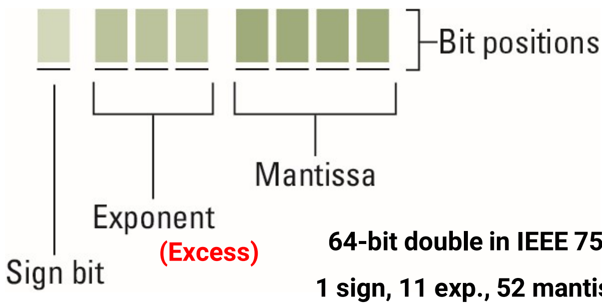
The probability of winning the lottery: 1/10000000

- We like **floating-point notation**:
  - 3×10^{8}, 6×10^{23}, 1×10^{-7}

63

Storing Fractions

- **Floating-point notation**
  - Consists of a sign bit, a mantissa field, and an exponent field



64-bit double in IEEE 754:

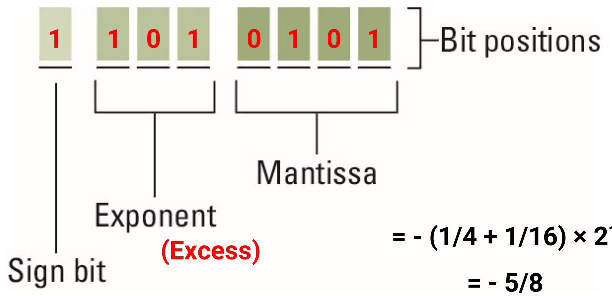
1 sign, 11 exp., 52 mantissa

64

Storing Fractions (cont.)

Bit pattern	Value represented
111	3
110	2
101	1
100	0
011	–1
010	–2
001	–3
000	–4

- **Floating-point notation**
  - Example:



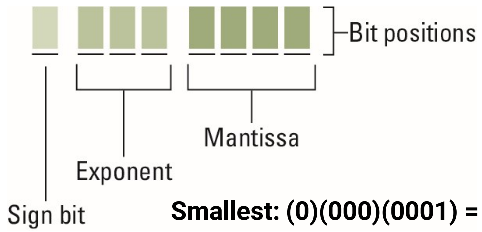
$= -(1/4 + 1/16) \times 2^1$

$= -5/8$

65

Storing Fractions (cont.)

- Example questions:
  - What are the smallest (larger than zero) / largest **positive** fractions using the following floating-point notation? **(assume the normalized form is not used)**



Smallest: (0)(000)(0001) = + (1/16) × 2^{-4}

Largest: (0)(111)(1111) = + (15/16) × 2^{3}

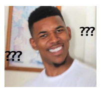
66

Truncation (Round-off) Errors

- Occur when part of the value being stored is lost because the **mantissa is not large enough**
- Non-terminating expansions of fractions
  - This happens more often with binary notation
  - The value of one-tenth cannot be stored exactly in binary notation
  - Often these values are converted to integers
- Example

$2\frac{5}{8} = \underline{10.101} = .1010\cancel{1} \times 2^2 \rightarrow 0(110)(1010) = 2\frac{1}{2}$

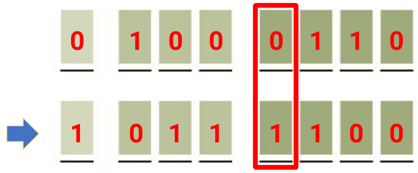
fixed point



67

Normalized Form for Fractions

- Issue: both 0011100 and 01000110 would decode to 3/8
- **Normalized form**
  - Eliminate the possibility of multiple representations for the same value
  - Fill the mantissa **starting with the left-most 1**



- **Special case: zero (all eight bits be zero)**

68

Numerical Analysis

- The study of dealing with problems when computing large values that require significant accuracy
- The order in which values are added can lead to two different results
- Adding very small values to very large values can result in errors

Introduction to Computer 2022

Numerical Analysis (cont.)

- Example

4 + (1/4) + (1/4)   (using 3-bit)

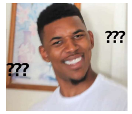
= 0(111)(1000) + 0(011)(1000) + 0(011)(1000)

= 0(111)(1000) + 0(111)(0000~~1~~) + 0(011)(1000)

= 0(111)(1000) + 0(011)(1000)

= 0(111)(1000) + 0(111)(0000~~1~~)

= 0(111)(1000) = 4



Bit pattern	Value represented
111	3
110	2
101	1
100	0
011	−1
010	−2
001	−3
000	−4

Introduction to Computer 2022

Numerical Analysis (cont.)

- Example

4 + (1/4) + (1/4)   (using 3-bit)

= 0(111)(1000) + 0(011)(1000) + 0(011)(1000)

= 0(111)(1000) + 0(100)(1000)

= 0(111)(1000) + 0(111)(0001)

= 0(111)(1001) = 4 + (1/2)

Bit pattern	Value represented
111	3
110	2
101	1
100	0
011	−1
010	−2
001	−3
000	−4

Introduction to Computer 2022

Outline

- Bits and their storage
- Main memory
- Mass storage
- Representing information as bit patterns
- The binary system
- Data and compression
- Communication errors

Data Compression Classification

Lossless compression

Run-length encoding

Frequency-dependent encoding (Huffman codes)

Dictionary encoding (including LZW encoding)

Lossy compression

Images
- GIF
- JPEG

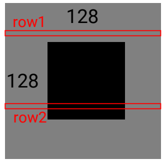
Sound
- MP3
- Frequency/Temporal masking

Video
- MPEG

Relative / Difference encoding

Run-length Encoding

- One of the simplest compression techniques
- A stored value is followed by a count to indicate the number of **consecutive occurrences** of that value
- Example
  - Consider the following gray-scale image with two colors: gray (pixel value = 128) and black (pixel value = 0)

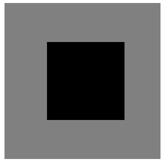


RLE for row1: 128 128

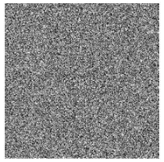
RLE for row2: 128 32 0 64 128 32

Insights from Run-length Encoding

- The effectiveness of data compression depends on the content of the data
  - The size can become bigger after applying compression
  - Definitely true, otherwise, any data can be compressed into one byte



128 bytes for a row



256 bytes for a row

Frequency-dependent Encoding

- General idea: using more bits for more frequent data

Word	Parts of speech	OEC rank	COCA rank[8]	Dolch level	Polysemy
the	Article	1	1	Pre-primer	12
be	Verb	2	2	Primer	21
to	Preposition	3	7, 9	Pre-primer	17
of	Preposition	4	4	Grade 1	12
and	Conjunction	5	3	Pre-primer	16
a	Article	6	5	Pre-primer	20
in	Preposition	7	6, 128, 3038	Pre-primer	23
that	Conjunction et al.	8	12, 27, 903	Primer	17
have	Verb	9	8	Primer	25
I	Pronoun	10	11	Pre-primer	7
it	Pronoun	11	10	Pre-primer	18
for	Preposition	12	13, 2339	Pre-primer	19

The most-frequently used words in a general article (data from wiki)

Huffman Encoding

- The best-known frequency-dependent (variable-length) coding
- Example

X Y W X X X X W X X X W X X Z X
00011100000000110000001100001000

Naïve approach (fixed-length coding)

needs 32 bits for 16 word

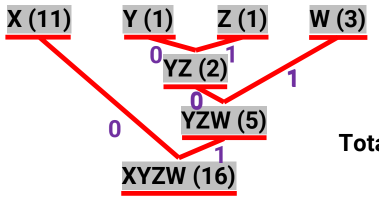
Data	Code
X	00
Y	01
Z	10
W	11

Huffman Encoding (cont.)

- The best-known frequency-dependent (variable-length) coding
- Example

X Y W X X X X W X X X W X X Z X
01001100001100011001010

Huffman encoding



Data	Frequency	Code
X	11	0
Y	1	100
Z	1	101
W	3	11

Total bits: 11×1 + 1×3 + 1×3 + 3×2
= 23 bits (+ dictionary)

Huffman Decoding

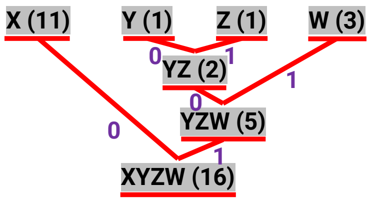
Naïve approach (fixed-length coding)

X Y W X X X X W X X X W X X Z X
00011100000000110000001100001000

Data	Code
X	00
Y	01
Z	10
W	11

Huffman encoding

X Y W X X X X W X X X W X X Z X
01001100001100011001010



Data	Frequency	Code
X	11	0
Y	1	100
Z	1	101
W	3	11

LZW Encoding

- A popular dictionary-based encoding algorithm
  - Dynamically grow the dictionary
  - **Do not need to share the codebook (dictionary)**

xyx xyx xyx xyx

1
12
121
1213 → knowing xyx forms a word
12134
121343
1213434
12134343
121343434

Dictionary

Symbol	Code
x	1
y	2
space	3
xyx	**4**

From ASCII code

Dynamically grow

LZW Decoding

- A popular dictionary-based encoding algorithm
  - Dynamically grow the dictionary
  - **Do not need to share the codebook (dictionary)**

121343434

x
xy
xyx
xyx_ → knowing xyx forms a word
xyx_xyx
xyx_xyx_
xyx_xyx_xyx
xyx_xyx_xyx_
xyx_xyx_xyx_xyx

Dictionary

Symbol	Code
x	1
y	2
space	3
xyx	**4**

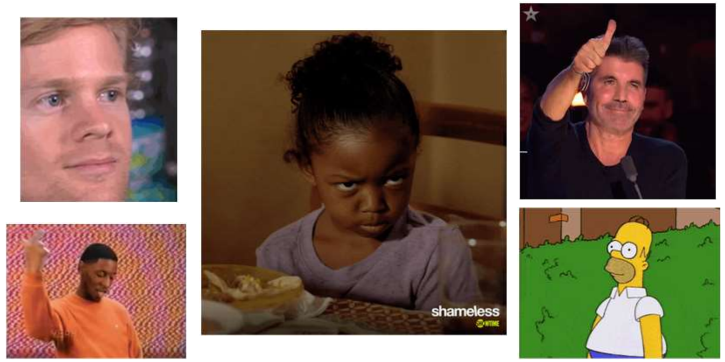
From ASCII code

Dynamically grow

81

GIFs

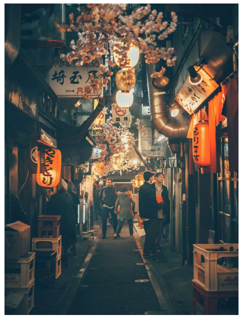
- Only store 256 different colors (dictionary)

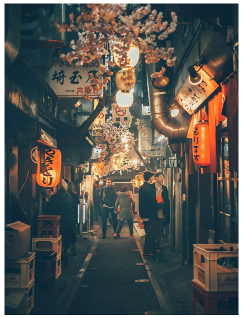


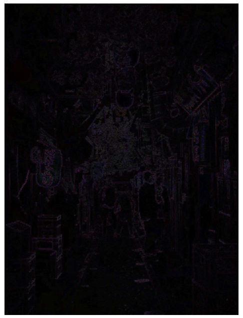
82

JPEG

- Which image is compressed?


PNG 7.46 MB

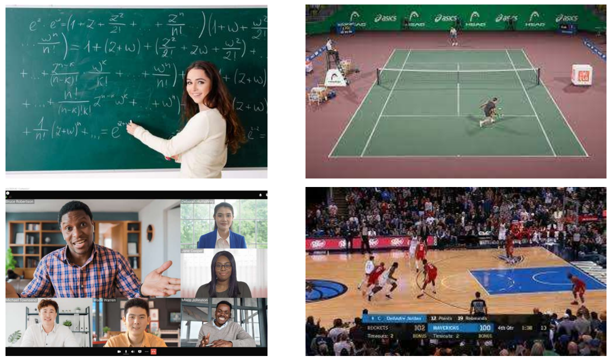

JPG 464 KB


4x difference

83

Relative / Difference Encoding

- Only record the dynamically changing parts



84

Outline

- Bits and their storage
- Main memory
- Mass storage
- Representing information as bit patterns
- The binary system
- Data and compression
- Communication errors

Communication Errors

- **Compression**
  - Remove redundancy
- **Error detection** and **correction**
  - Add redundancy to prevent (communication) errors
- **Error detection (using check code)**
  - Cannot correct errors, but can check if errors occur
  - Examples: ID numbers, ISBN, parity code
- **Error correction**
  - Can correct errors to some degrees

Error Detection: Taiwan's ID

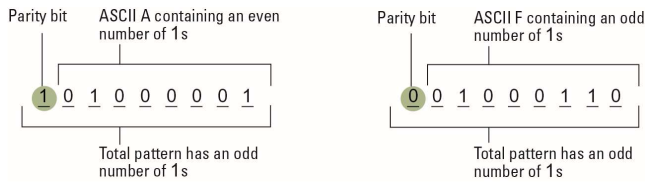
$$Ca_1a_2a_3a_4a_5a_6a_7a_8a_9$$

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
10	11	12	13	14	15	16	17	34	18	19	20	21	22	35	23	24	25	26	27	28	29	32	30	31	33

- **Rule**
  - Convert the English letter into a number xy
  - Compute $d_1 = x + 9y$
  - Compute $$d_2 = \sum_{i=1}^{8}(9-i)a_i = 8 \cdot a_1 + 7 \cdot a_2 + \cdots 1 \cdot a_8$$
  - Check code $a_9 = 10 - ((d_1 + d_2) \bmod 10)$

Error Detection: Parity Bits

- Add an additional bit to make the number of 1s odd



- Applications
  - Communication
  - RAID (redundant array of independent disks)

Error Correction: Repetition Code

- (3,1)-repetition code (can correct 1-bit error)

Triplet received	Interpret as
000	0 (error free)
001	0
010	0
100	0
111	1 (error free)
110	1
101	1
011	1

Error Correction: Hamming Distances

- Maximized **Hamming distances** among symbols (at least 3)

Symbol	Code	Pattern received	Distance between received pattern and code
A	000000	0 **1** 0 **1** 0 0	2
B	001111	0 **1** **0** 1 **0** **0**	4
C	010011	0 1 0 **1** **0** **0**	3
D	011100	0 1 **0** 1 0 0	**1**
E	100110	**0** **1** 0 1 **0** 0	3
F	101001	**0** **1** **0** 1 0 0	5
G	110101	**0** 1 0 1 0 **0**	2
H	111010	**0** 1 **0** **1** **0** 0	4

Smallest distance

Any Questions?