16.070

Introduction to Computers & Programming

Machine architecture:
the binary system, storing numbers, data compression

Prof. Kristina Lundqvist
Dept. of Aero/Astro, MIT

The Binary System

- Decimal: Position represents a power of 10
  - $5382_{10} = 5x10^3 + 3x10^2 + 8x10^1 + 2x10^0$
- Binary: Position represents a power of 2
  - $1011_2 = 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0$
    $= 8 + 0 + 2 + 1 = 11_{10}$
- Binary addition

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

- A byte

bit number	7	6	5	4	3	2	1	0
	0	**1**	**0**	**1**	**0**	**0**	**1**	**1**
bit value	128	64	32	16	8	4	2	1

Representing Negative Numbers

- Using one byte, any suggestions for representing negative numbers?
  - E.g., $00010011_2 = 19_{10}$. How to represent -19_{10} in binary?

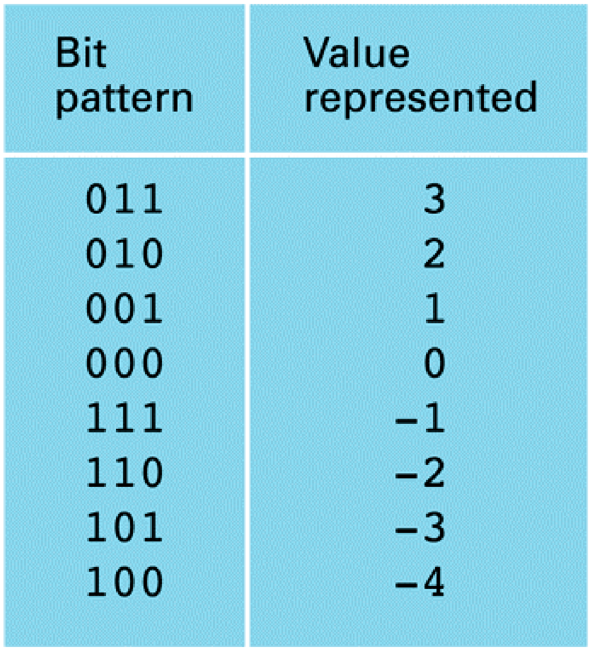
- Reserve 1 bit (#7) for sign, 7 bits for number (sign magnitude)
  - **0**$0000001_2 = 1_{10}$ **1**$0000001_2 = -1_{10}$
  - **0**$0010011_2 = 19_{10}$ **1**$0010011_2 = -19_{10}$

Representing Negative Numbers

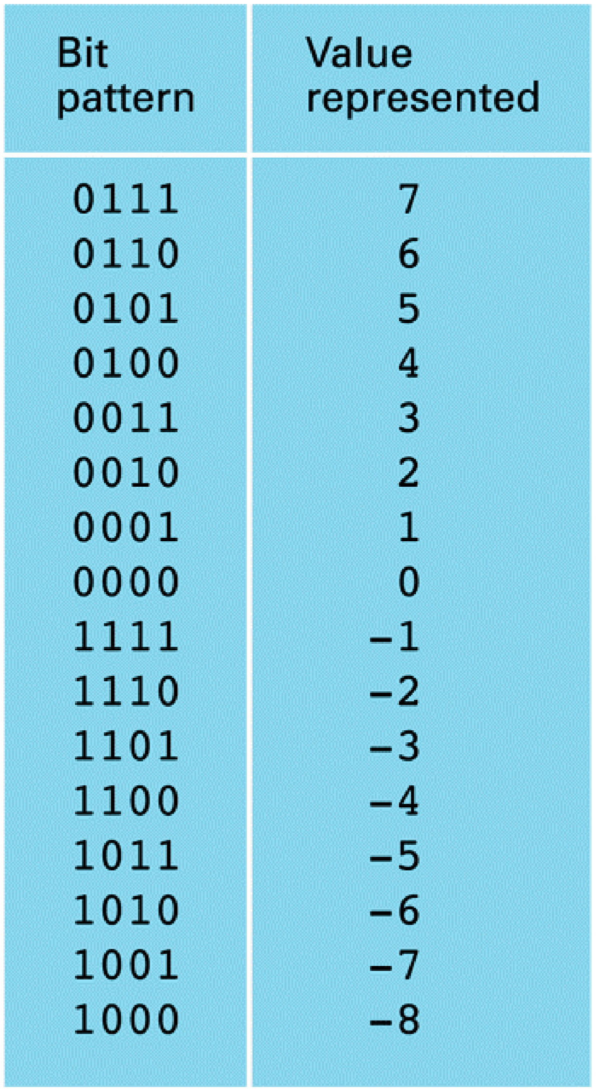
- One's complement – invert each bit
  - $00010011_2 = 19_{10}$ $11101100_2 = -19_{10}$
  - 0_{10}: 00000000_2 and 11111111_2

- **Two's complement** – invert each bit and add 1
  - $00010011_2 = 19_{10}$ $11101101_2 = -19_{10}$
  - Try to negate 0:
    - $0_{10} = 00000000_2$
    - invert: $00000000_2 \rightarrow 11111111_2$
    - add 1: $11111111_2 + 00000001_2 = 00000000_2$

Two's Complement Notation Systems

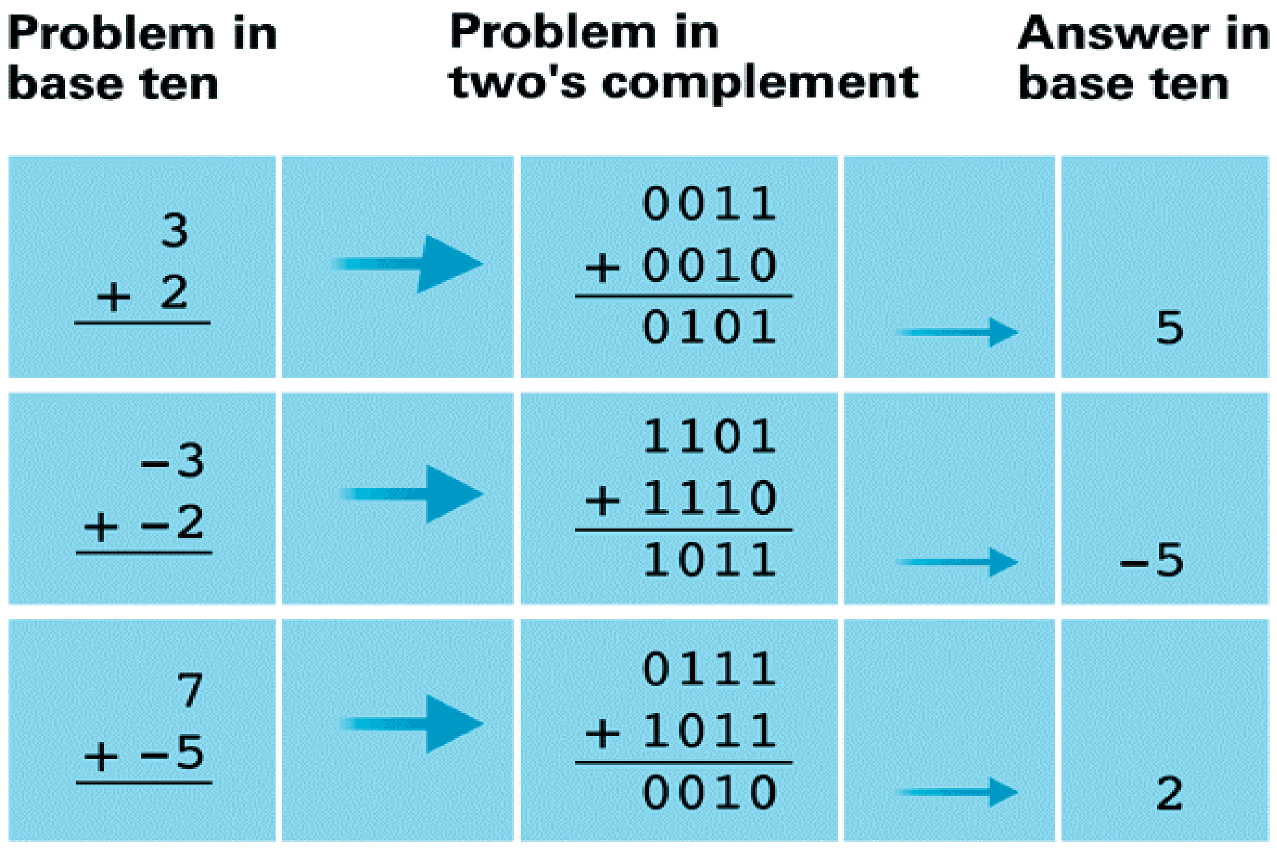


Bit pattern	Value represented
011	3
010	2
001	1
000	0
111	−1
110	−2
101	−3
100	−4



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

Addition



Problem in base ten		Problem in two's complement		Answer in base ten
$3 + 2$	$\rightarrow$	$0011 + 0010 = 0101$	$\rightarrow$	5
$-3 + -2$	$\rightarrow$	$1101 + 1110 = 1011$	$\rightarrow$	−5
$7 + -5$	$\rightarrow$	$0111 + 1011 = 0010$	$\rightarrow$	2

- Example: we are using 8 bit two's complement and have the problem 97-81.

```
 97 |  0 1 1 0 0 0 0 1
-81 | +1 0 1 0 1 1 1 1
----+-----------------
 16 |  0 0 0 1 0 0 0 0
```

Summary: One's/Two's Complement

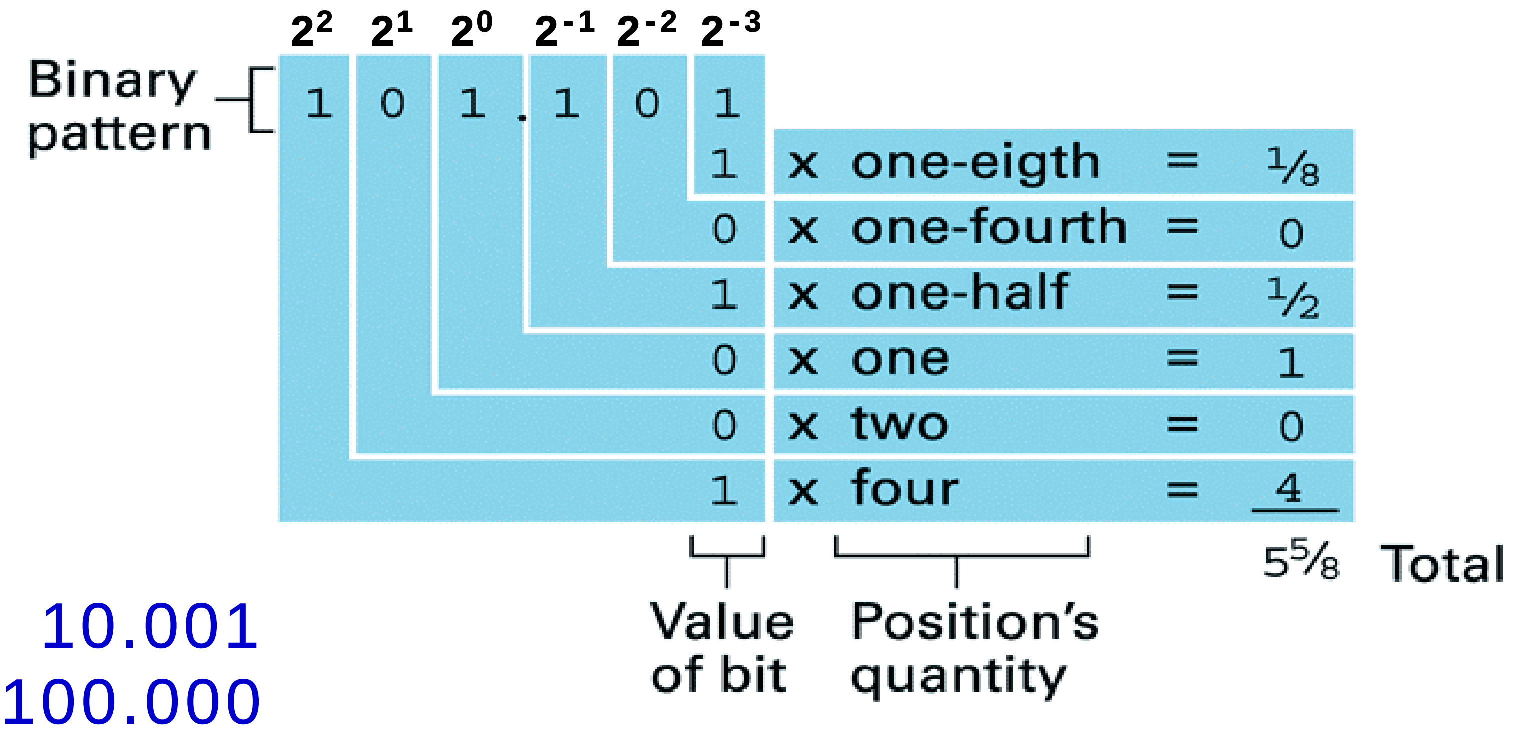
- Note that in *sign magnitude* and in *one's complement* there are two possible representations of 0, and we can represent every integer n with $-(2^k - 1) \leq n \leq 2^k - 1$ with a k-bit field.
- With *two's complement* there is only one representation of 0, and we can represent the integers n with $-2^k \leq n \leq 2^k - 1$.
- The most significant advantage of two's complement is the fact that subtraction can be performed with addition circuitry.

The Problem of Overflow

- Overflow: when a value to be represented falls outside the range of values that can be represented.
- An overflow is indicated if the addition of two positive values results in the pattern for a negative value or vice versa.
- Remember: small values can accumulate to produce large numbers.

Fractions in Binary

- Each position is assigned a quantity of twice the size of the one to its right.
- 101.101 = ?



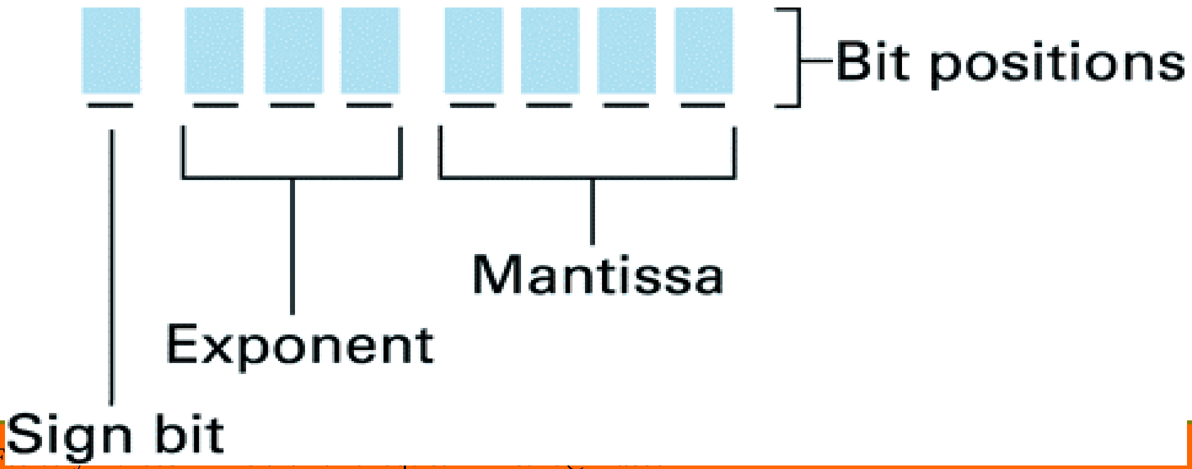
```
   10.001
+ 100.000
  111.001
```

Floating Point representation

- To represent a floating point number the number is divided into two parts: the integer and the fraction
  - 3.1415 has integer 3 and fraction 0.1415
- Converting FP to binary:
  - Convert integer part to binary
  - Convert fraction part to binary
  - Put a decimal point between the two numbers
- 71.3125 = +1000111.0101

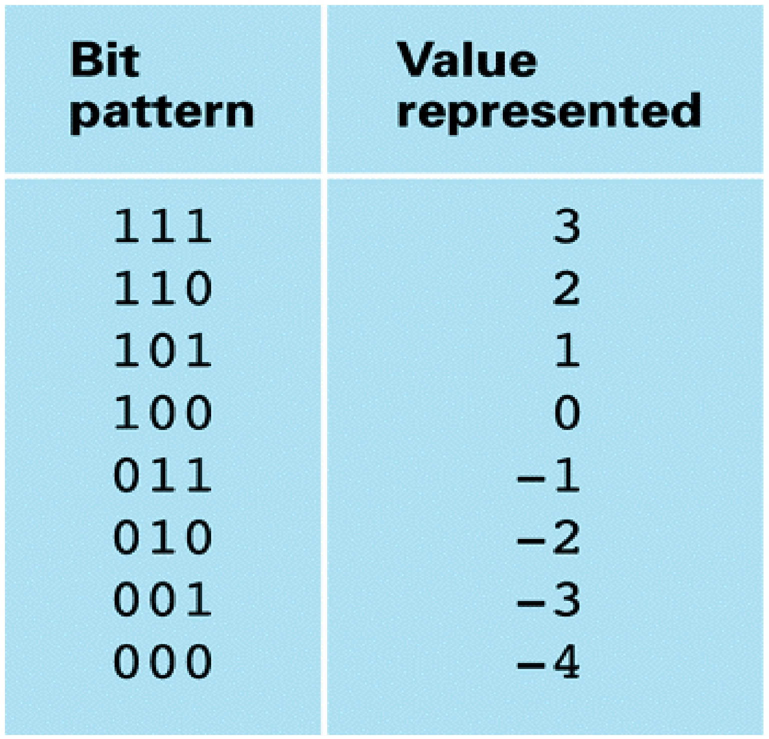
Normalization

- To represent +1000111.0101
  - Store sign, all bits, and the position of decimal point
- Instead we use Normalization
  - Move the decimal point so that there is only one 1 to the left of the decimal point.
  - To indicate the original value of the number, multiply by 2^e where e is the number of bits that the decimal point moved, positive for left movement, negative for right movement
  - Store:
    - The sign
    - The exponent
    - The mantissa



Excess (or bias) Notation

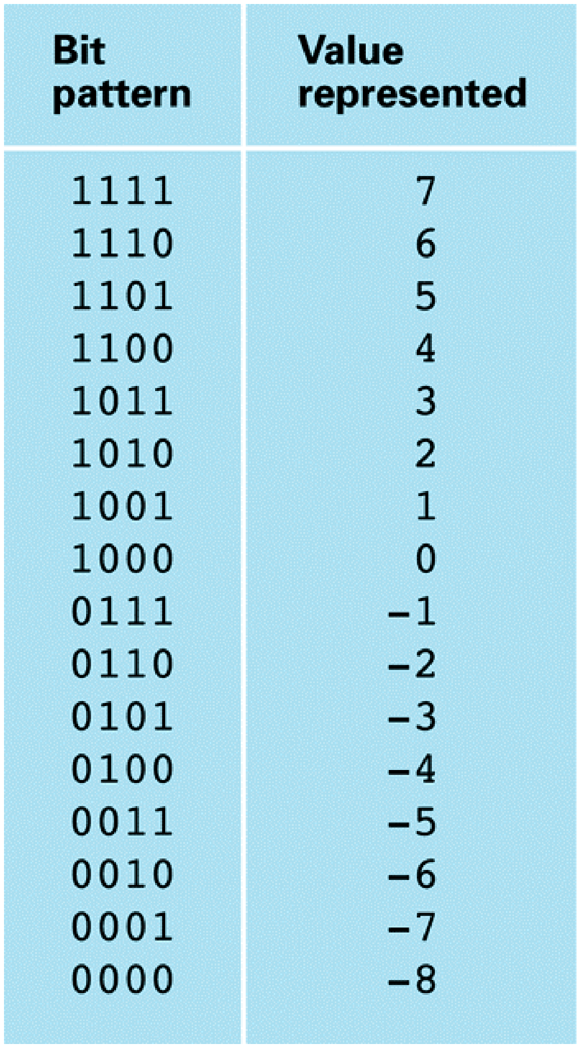
- Alternative to two's complement used to store the exponent for floating point numbers.



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

With 3 bits we have excess 4 ($=2^{3-1}$)

Excess (or bias) Notation



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

With 4 bits we have excess 8 ($=2^{4-1}$)

With N bits we have excess 2^{N-1}

We add the magic number, 2^{N-1}, to the integer, change the result to binary, and add 0's (on the left) to make up N bits.

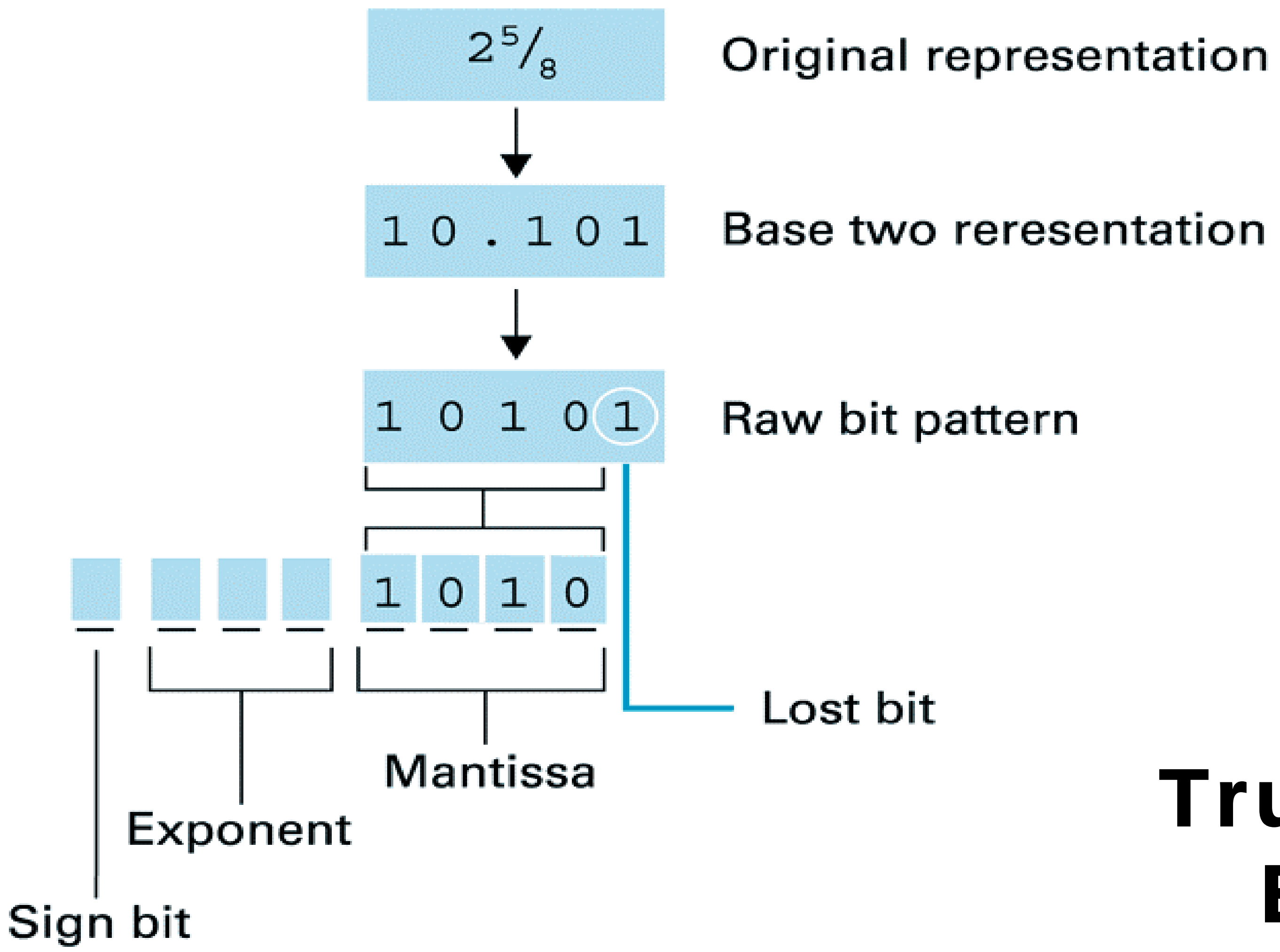
FP Representation

- The standard IEEE representation for FP uses 32 bits for single-precision format:
  - 1 bit sign
  - 8 bits exponent (excess 127 = 2^{8-1}-1)
  - 23 bits mantissa
- Representation. Store the:
  - Sign as 0 (positive), 1 (negative)
  - Exponent (power of 2) as excess127
  - Mantissa as an unsigned integer

Example: FP Representation

- Consider a 16-bit representation with
  - 1 bit sign
  - 5 bits exponent (excess 16 = 2^{5-1})
  - 10 bits mantissa
- 0 10011 1101100000
  - The sign 0 represents positive
  - The exponent 10011 = 19_{10} represents 3
  - The mantissa is .1101100000 (1 to left of . is not stored, it is understood)

Coding the value 2 5/8



Truncation Errors

Data Compression

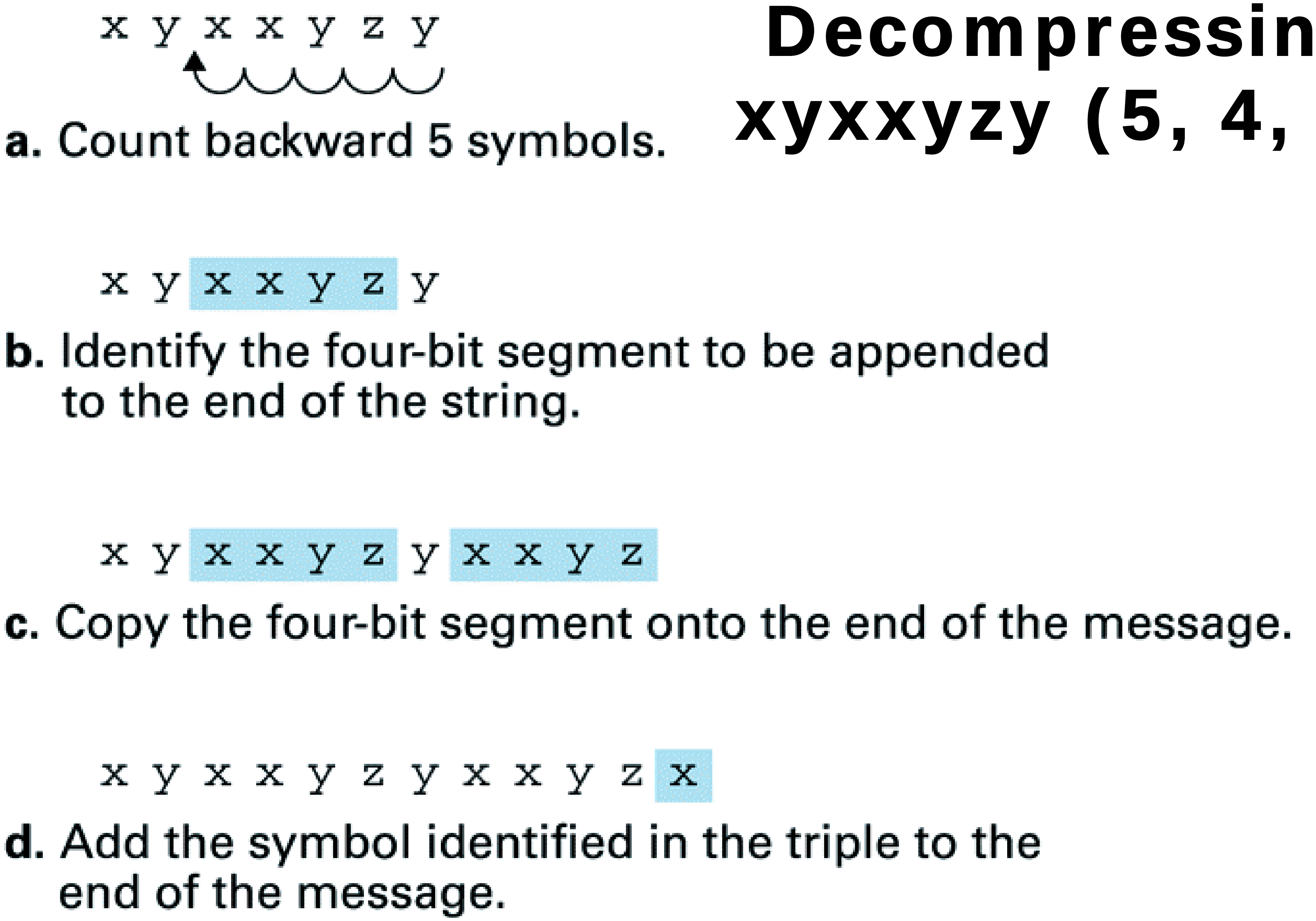
- Storing data in a format that requires less space than usual
  - E.g., transformation of a string of characters in some representation (such as ASCII) into a new string (of bits, for example) which contains the same information but whose length is as small as possible.
- CCITT (***C**omité **C**onsultatif **I**nternational **T**éléphonique et **T**élégraphique*)
  - Transmitting faxes
  - Communication through modems (CCITT V.42*bis)*
- File compression formats
  - ZIP
  - ARC (BBSs)
- Also widely used in
  - Backup utilities, spreadsheet applications, database management systems
  - Certain types of data, such as bit mapped graphics, can be compressed to a small fraction of their normal size.

Generic Data Compression Techniques

- Run-length encoding
- Relative encoding
- Frequency-dependent encoding
  - Huffman codes
- Adaptive dictionary encoding
  - Lempel-Ziv encoding
- Graphic Interchange Format
- Joint Photographic Experts Group

Lempel-Ziv Encoding

Decompressing
xyxxyzy (5, 4, x)



x y x x y z y

a. Count backward 5 symbols.

x y x x y z y

b. Identify the four-bit segment to be appended to the end of the string.

x y x x y z y x x y z

c. Copy the four-bit segment onto the end of the message.

x y x x y z y x x y z x

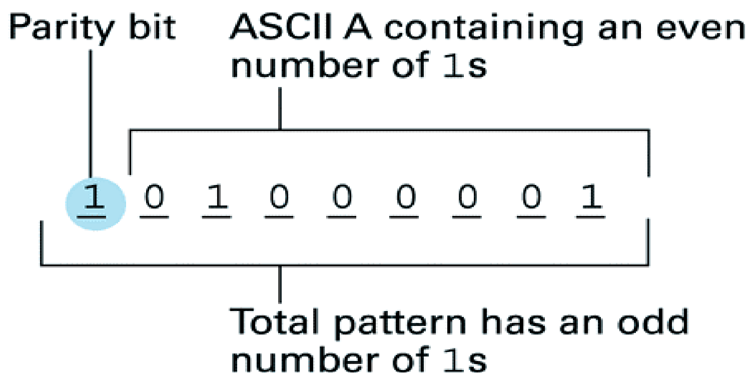
d. Add the symbol identified in the triple to the end of the message.

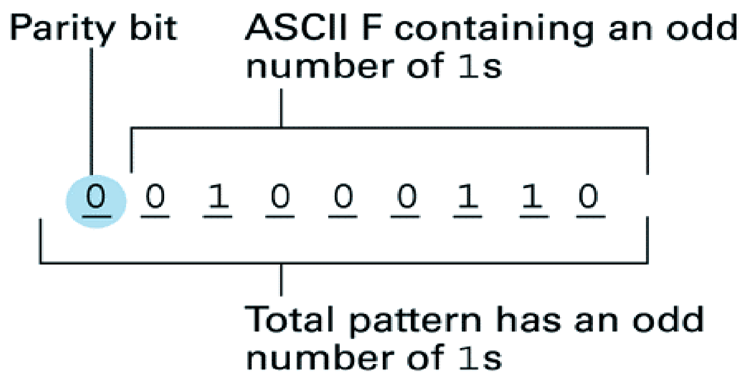
Error Detection and Correction

- Errors in data representation can be caused by factors such as noisy transmission (phone lines, radio links), power surges, …
- Bits can flip during transmission or on orbit inside spacecraft processors
- What if one bit is in the wrong state?
  - One bit flip can have dramatic ramifications
  - Must develop/use routines to
    - Detect single bit errors
    - Correct for single bit errors (for system critical info)
    - Make sure communication link and on-board processors are robust enough that more than one error per word is extremely unlikely

Communication Errors

- Parity bit

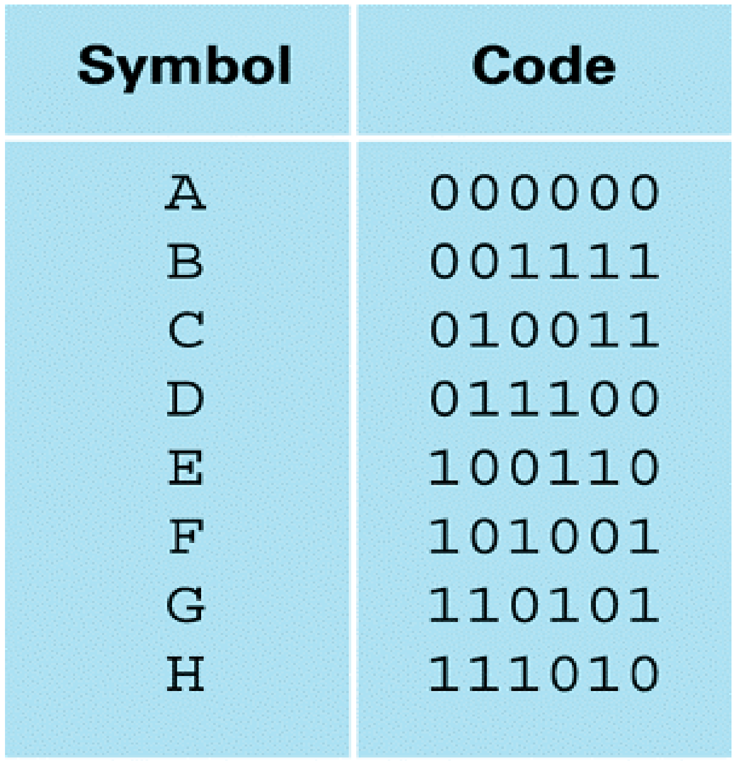

Parity bit
ASCII A containing an even number of 1s
1 0 1 0 0 0 0 0 1
Total pattern has an odd number of 1s


Parity bit
ASCII F containing an odd number of 1s
0 0 1 0 0 0 1 1 0
Total pattern has an odd number of 1s

  - **Odd** parity
  - Even parity
- Checksum

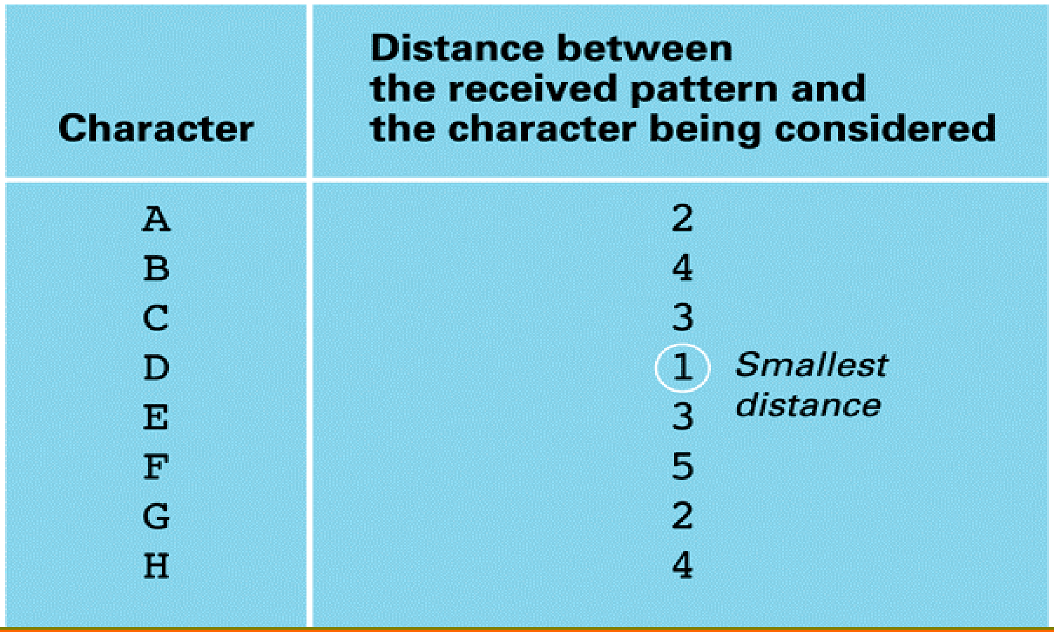
Error-Correcting Codes

- Hamming distance between two bit patterns = number of bits that differ
  - Hamming distance between A and B is 4
  - Hamming distance between B and C is 3



Symbol	Code
A	000000
B	001111
C	010011
D	011100
E	100110
F	101001
G	110101
H	111010

010100



Character	Distance between the received pattern and the character being considered
A	2
B	4
C	3
D	1 *Smallest distance*
E	3
F	5
G	2
H	4

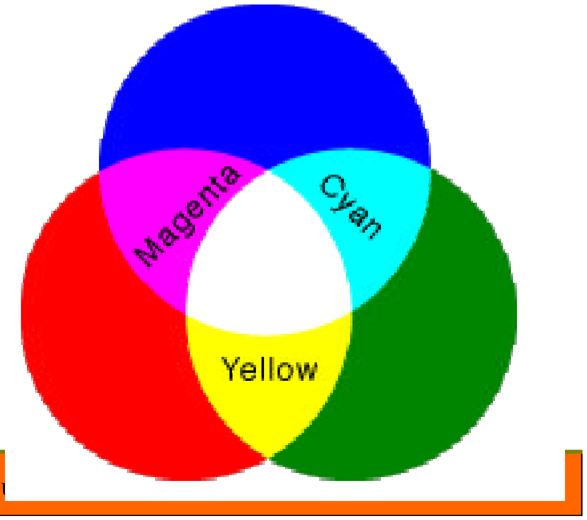
Evening Quiz

Day:	Tuesday
Date:	Apr. 8, 2003
Time:	7:30p – 9:30p
Room:	35-225

- Monday lecture Apr. 7, 2003 is **canceled**

Colors

- Primary colors of **pigment**: Red, **Yellow**, Blue, that combined produce secondary colors
- Primary colors of **light**: Red, **Green**, Blue (RGB).
- Pigment color system does not apply to computer monitors because colors are created on monitors by adding light.
  - RGB is an "additive" color system, i.e., color is added to a black background.



http://www.webopedia.com/DidYouKnow/Computer_Science/2002/Color.asp

RGB

- A **monochrome monitor** can display only two colors, one for the background and one for the foreground.
- **Color monitors** implement the RGB color model by using three different phosphors that appear Red, Green, and Blue when activated. Placing the phosphors directly next to each other, and activating them with different intensities, colour monitors can create an unlimited number of colours.
  (in practice though, the real number of colors that any monitor can display is controlled by the "video adapter")