

INFO-C 307

Data

Representation and Organization

Unit-1, Physical Representation of Data



Dr. Hossein Hakimzadeh
Computer Science and Informatics
IU South Bend

Part – A

Data Encoding

What is a Computer?

- ❑ An Electronic digital device that can store, and process data.
- ❑ A **Fast** and **Accurate** electronic symbol (or data) manipulating system that is designed to accept and store **input data**, **process them** and **produce output** result.
- ❑ A programmable, multi-use machine that accepts data (raw facts) and processes, or manipulates it into information we can use.

How does a Computer work?

□ We must answer the following two questions:

1) How is data represented inside a computer?
(Encoding)

2) How is data manipulated inside a computer?
(Algorithms)

How is Data represented?

- **Encoding** is the process of transforming information from one format into another. The opposite operation is called decoding. This is often used in many digital devices.
(<http://en.wikipedia.org/wiki/Encoding>)



Chinese Calligraphy



UPC (Universal Product Code)

How is Data represented?

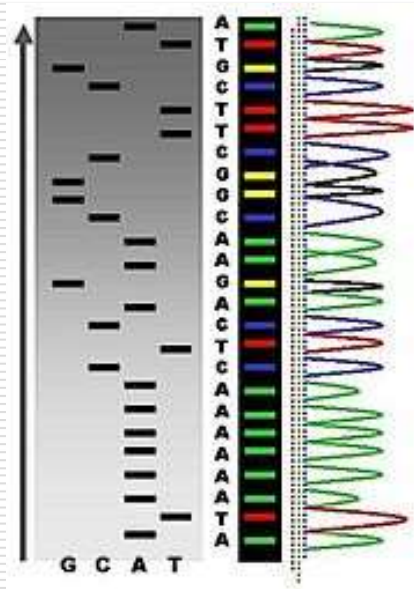
□ Encoding

2800		Braille Patterns												28FF		
	280	281	282	283	284	285	286	287	288	289	28A	28B	28C	28D	28E	28F
0																
1																
2																
3																
4																

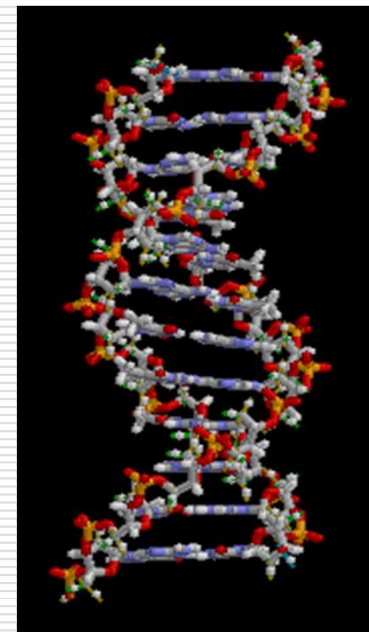
<http://www.unicode.org/charts/PDF/U2800.pdf>

How is Data represented?

□ Encoding



http://en.wikipedia.org/wiki/DNA_sequencing



<http://en.wikipedia.org/wiki/DNA>

□ Encoding

	Miscellaneous Symbols																26FF
	260	261	262	263	264	265	266	267	268	269	26A	26B	26C	26D	26E	26F	
0																	
1																	
2																	
3																	
4																	
5																	
6																	
7																	
8																	
9																	
A																	
B																	
C																	
D																	
E																	
F																	

How is Data represented?

□ Encoding



http://en.wikipedia.org/wiki/Musical_notation

How is Data represented?

□ Encoding

International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.

A	• —	U	• • —
B	— • • •	V	• • • —
C	— • — •	W	• — —
D	— • •	X	— • • —
E	•	Y	— • — —
F	• • — •	Z	— — • •
G	— — •		
H	• • • •		
I	• •		
J	• — — —		
K	— • —	1	• — — — —
L	• — • •	2	• • — — —
M	— —	3	• • • — —
N	— •	4	• • • • —
O	— — —	5	• • • • •
P	• — — •	6	— • • • •
Q	— — • —	7	— — • • •
R	• — •	8	— — — • •
S	• • •	9	— — — — •
T	—	0	— — — — —

How is Data represented?

- ❑ **Encoding (QR code) (Quick Response Code)**



Informatics @ Indiana University

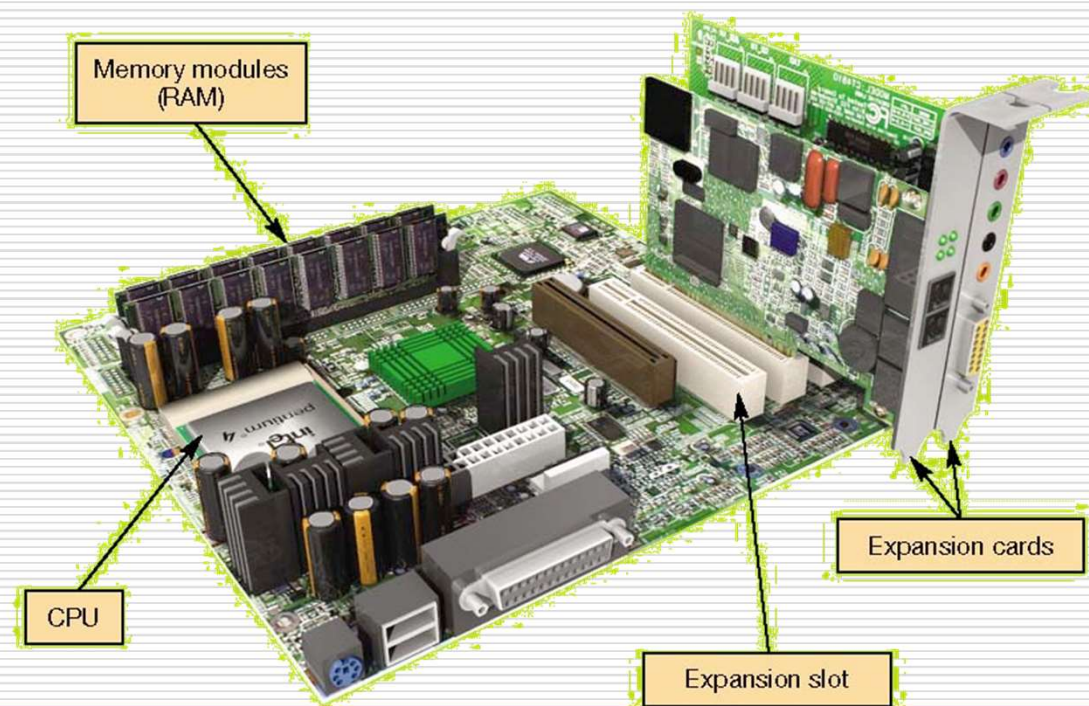


<https://informatics.iusb.edu/>

See more at: <https://www.qr-code-generator.com>

How is Data represented inside the Computer?

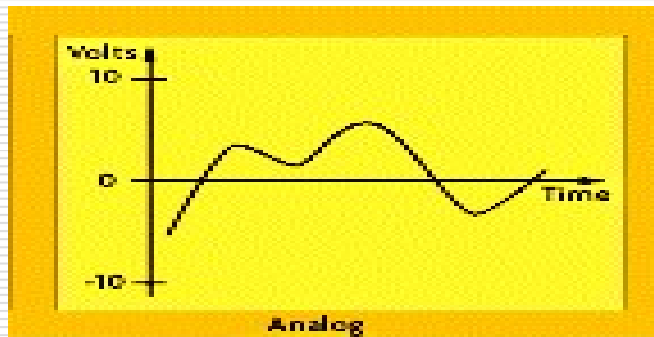
- ❑ **Remember, A computer is** an electronic **digital** device that can store, and process data.



Digital vs. Analog?

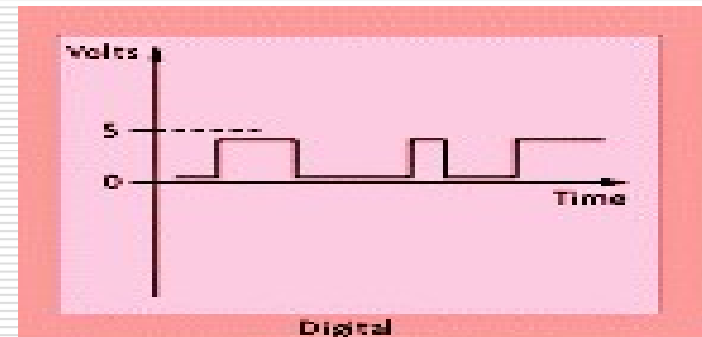
- **Analog** systems have a continuous range of values.

- Vinyl records
- Analog clocks
- Set of real numbers



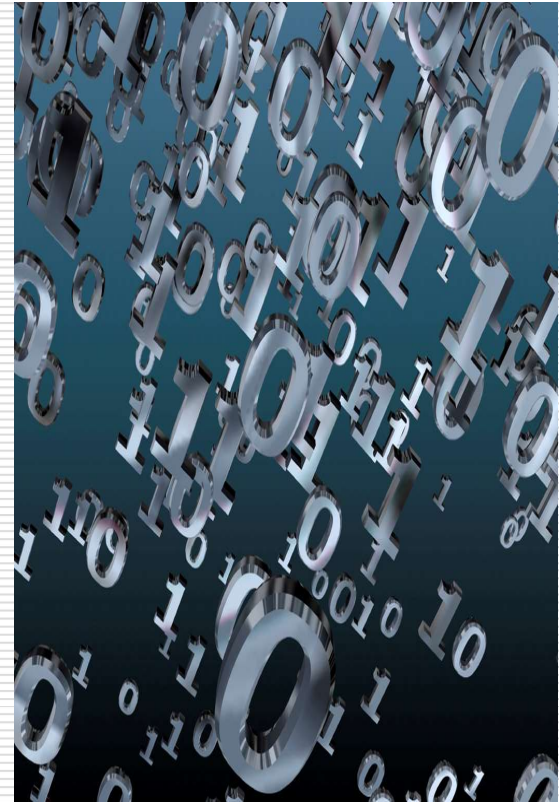
- **Digital** systems have a set of discrete values.

- CD's and DVD's
- Digital clocks
- Set of integer numbers



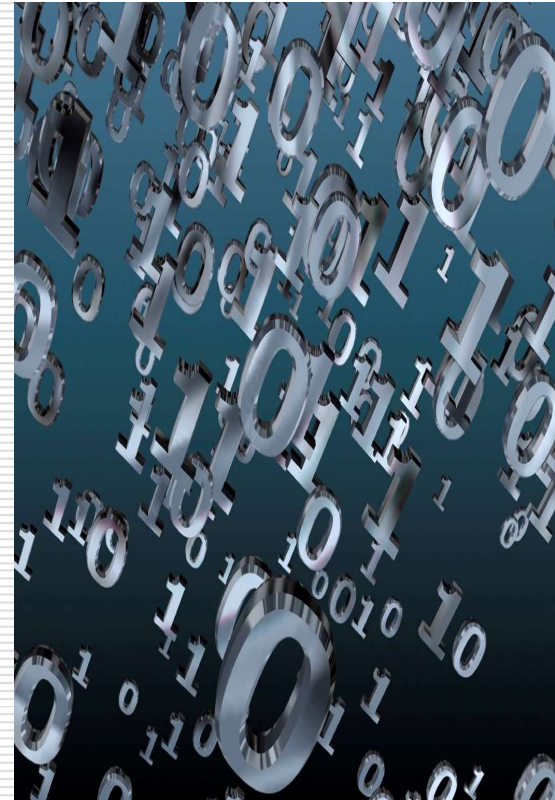
How is information represented inside the Computer?

- ❑ Binary digits or BITS (0's and 1's)
- ❑ Why Binary Digits?



How is information represented inside the Computer?

- ❑ Digital Computers are designed to process data in numerical form. They can store and manipulate information such as numbers, characters, images, and sound using **numbers**.
- ❑ The information inside the computer is expressed in the **binary system**.
- ❑ Binary digits (bits), are made up of **0's and 1's**.
- ❑ Binary digits are easily expressed in the computer circuitry by the **presence or absence of voltage**. For example 1 may mean 5 volts and 0 may mean 0 volts.



How is Data represented?

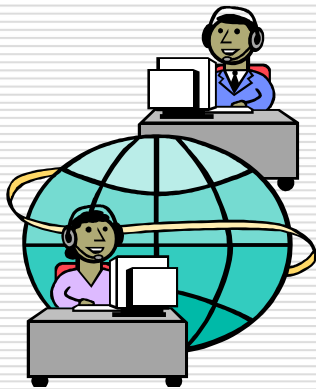
- ❑ **Characters** are also represented using binary numbers.
- ❑ We use 8 bits to uniquely represent each character on the keyboard.
- ❑ The partial ASCII table (right) shows how characters are mapped into binary numbers.
- ❑ ASCII stands for (American Standard Code for Information Interchange)

Decimal	Binary	Character
65	01000001	A
66	01000010	B
67	01000011	C
68	01000100	D

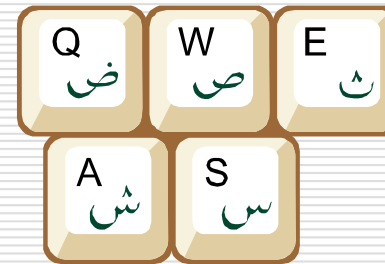


What about other languages?

- UNICODE is another standard used to represent characters other than English.



こんにちは



привет

여보세요

ciao

你好

مرحبا

γειά σου

hälsningar



How is Data represented inside the Computer?

□ Bit (Binary digIT)

- (A bit is a unit of storage in a computer)
- (A bit is a single binary digit. 0 or 1)

□ A Byte is 8 Bits

- KiloByte (KB) = 2^{10} or 1024 bytes (Approximately 1,000 bytes)
- MegaByte (MB) = 2^{20} bytes (Approximately 1,000,000 bytes)
- GigaByte (GB) = 2^{30} bytes (Approximately 1,000,000,000 bytes)
- TeraByte (TB) = 2^{40} bytes (Approximately 1,000,000,000,000 bytes)
- PetaByte (PB) = 2^{50} bytes

Problem 1:

- ❑ You just bought a 60-gigabyte drive.
- ❑ After formatting the drive, you found out that it is only 58.6 Gigabytes? What should you do?

Solution 1:

- You just bought a 60-gigabyte drive. After formatting the drive, you found out that it is only 58.6 Gigabytes? What should you do?
- **Nothing!**
 - Most drive manufactures use a Giga Byte to mean one Billion Bytes!
 - $60,000,000,000 / (1024 * 1000 * 1000) = 58.593$

Encoding

- Given that computers only understand binary numbers, in order to store and manipulate information inside a computer, we must find a way to encode information in binary.
- This information may be **NUMBERS**, **TEXT**, or other types of data such as **AUDIO**, **IMAGE** or **VIDEO**.

Encoding Text

- Imagine our language was restricted to the following symbols (letters):
- Source alphabet: {n,k,b,e,r,d,i}
- Target alphabet: {0, 1}
- Encoding: (we are using 3 bits to represent each letter)
 - n = 000
 - k = 001
 - b = 010
 - e = 011
 - r = 100
 - d = 101
 - i = 110
- **Decode:** 101100110000001010011011100 = _____

Part 1 – B

Data Encoding, Continued...

Problem 2:

- How many bits do we need in order to represent all the 26 upper case English letters?
- How many bits do we need to represent the **upper** and **lower** case letters, plus **all the numbers**, and the symbols (@, \$, & and #)

Problem 2:

- How many bits do we need in order to represent all the 26 upper case English letters?

calculate the smallest x , such that:

$$2^x \geq 26$$

- How many bits do we need to represent the **upper** and **lower** case letters, plus **all the numbers**, and the symbols (@, \$, & and #)

calculate the smallest x , such that:

$$2^x \geq (26 + 26 + 10 + 4)$$

Problem 2:

- How many bits do we need in order to represent all the 26 upper case English letters?

calculate the smallest x , such that:

$$2^x \geq 26$$

$2^5 = 32$ which will accommodate more than 26 letters

- How many bits do we need to represent the **upper** and **lower** case letters, plus **all the numbers**, and the symbols (@, \$, & and #)

calculate the smallest x , such that:

$$2^x \geq (26 + 26 + 10 + 4)$$

$2^7 = 128$ which will accommodate more than 66 symbols

ASCII Code

- ❑ American Standard Code for Information Interchange
- ❑ Why? Standardization between computers
- ❑ 7 or 8 bits are used to represent all the letters, numbers, and symbols, that appear on the English language keyboard.
 - A = 01000001 = 65
 - B = 01000010 = 66
 - C = 01000011 = 67
- ❑ <http://www.asciitable.com/>

UNICODE Code

- ❑ 16 to 32 bits vs. 8 bit code
- ❑ Why? ... Internationalization of computers and applications)
- ❑ **Hello** = U+0048 U+0065 U+006C U+006C U+006F
- ❑ There are many UNICODE encoding standards. These include:
 - UTF-8 (Treats English as normal ASCII would, then accommodate other characters as 2 or more byte characters)
 - UTF-16 or UCS-2 (2 byte code to store each Unicode character)
 - UTF-32 or UCS-4 (4 byte code to store each code point or Unicode character)
- ❑ It is important to know what encoding standard is being used before attempting to decode a string!

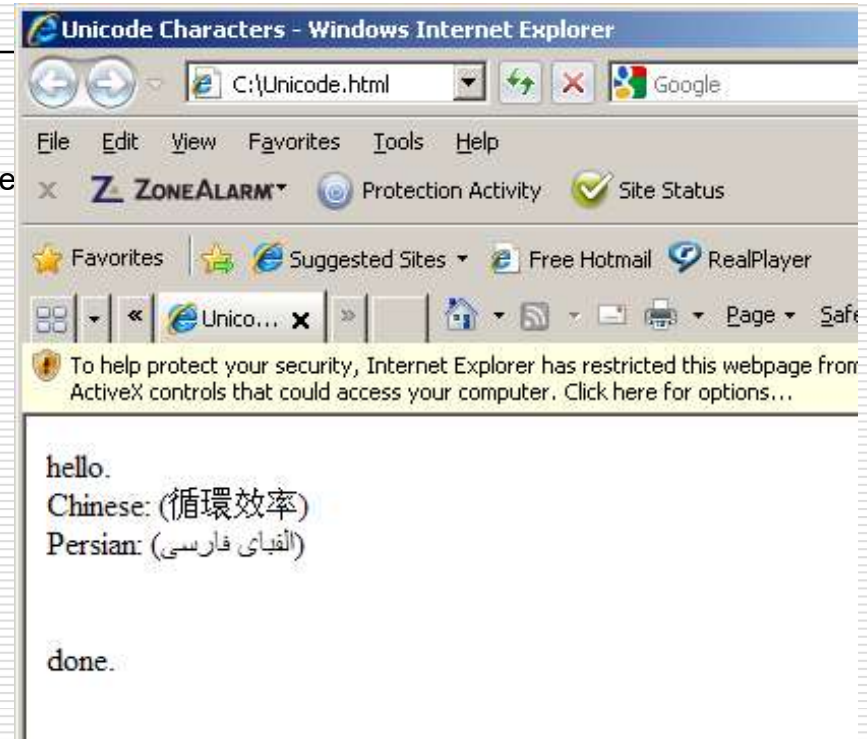
Unicode Example:

- Insert the following Unicode into text a html file and try to view it using a browser:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Unicode Characters</title>
</head>
<body>
hello.
<BR>
Chinese:
(&#x5FAA;&#x74B0;&#x6548;&#x7387;)
<BR>
Persian:
(&#1575;&#1604;&#1601;&#1576;&#1575;&#1740;
&#1601;&#1575;&#1585;&#1587;&#1740;) </h2>
<BR>
done.
</body>
</html>
```

Unicode Example:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charse
<title>Unicode Characters</title>
</head>
<body>
hello.
<BR>
Chinese:
(&#x5FAA;&#x74B0;&#x6548;&#x7387;)
<BR>
Persian:
(&#1575;&#1604;&#1601;&#1576;&#1575;&#1740;
&#1601;&#1575;&#1585;&#1587;&#1740;) </h2>
<BR>
done.
</body>
</html>
```



امروز سنتر ندارم
گوشی یادم نره
قرصهای صبح یادم نره
امروز بهرام میاد اینجا



امروز چهارشنبه است
(۱۳ ژانویه)

ساعت ۱۰/۰۵ صبح

روزهای سنتر:
(سه شنبه - پنج شنبه - جمعه)
(ساعت ۷:۳۰-۸:۰۰ صبح میان دنبال شما)



Encoding Numbers:

- How do we represent numbers?

- Character "1" in the ASCII table is encoded as 00110001 (decimal 49.)
- Character "2" is 00110010 = 50
- Character "3" is 00110011 = 51

See: <https://www.ascii-code.com/>

- Can we use ASCII representation of numbers for the purpose of calculations?
- Can we add Character "1" and Character "2" to get "3"?

Encoding Numbers:

- ❑ If ASCII representation of numbers can not be used, then we need a different encoding to be able to represent numbers and perform calculations.
- ❑ What is a suitable encoding?
- ❑ ***Hold on to this idea. We'll come back to it.....***

Part 1– C

Number Systems

Number Systems

- ☐ **Decimal (Base 10)**
- ☐ **Binary (Base 2)**
- ☐ **Octal (Base 8)**
- ☐ **Hexadecimal (Base 16)**

Decimal (Base 10)



- ❑ Used by humans (probably because we have 10 fingers!)
- ❑ Numbers in base 10 are (0, 1, 2, 3, 4, , 9) (always from 0 to Base -1)
- ❑ Example: (254)

■ What is 254?

2	5	4	
$2 * 10^2$	$5 * 10^1$	$4 * 10^0$	
200	50	4	254

Binary (Base 2)

- ❑ Used by digital computers (remember the ON / OFF states)
- ❑ Numbers in base 2 are (0, 1) (always from 0 to Base -1)
- ❑ Example: (Binary 110)

■ What is Binary 110?

1	1	0	
$1 * 2^2$	$1 * 2^1$	$0 * 2^0$	
4	2	0	6

Octal (Base 8)

- ❑ Used by people when they want to represent large binary numbers. Its easier to deal with.
- ❑ Numbers in base 8 are (0, 7) (always from 0 to Base -1)

Octal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Octal (Base 8)

□ Example: (Octal 251)

2	5	1	
$2 * 8^2$	$5 * 8^1$	$1 * 8^0$	
128	40	1	169

□ Octal 251 can be converted to binary very easily. Each number will be represented by 3 binary digits (bits).

2	5	1	
010	101	001	010 101 001
			169

Hexadecimal (Base 16)

- ❑ Similar to Octal, Hex numbers are used by people when they want to represent larger binary numbers. Its easier to deal with.
- ❑ Numbers in base 16 are (0,1,2,....,9, A, B, C, D, E, F) (always from 0 to Base -1)
- ❑ To keep each number as one character we use the letters "A" through "F" as numbers 10 to 15. (A = 10, B = 11, C=12, D=13, E=14, F=15)

Hexadecimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Hexadecimal (Base 16)

□ Example:

- What is HEX 25A?

2	5	A	
$2 * 16^2$	$5 * 16^1$	$A * 16^0$	
512	80	10	602

- HEX 25A can be converted to binary very easily. Each number will be represented by 4 binary digits (bits).

2	5	A	
0010	0101	1010	0010 0101 1010
			602

Part 1 – D

Encoding Integers

Different Ways of Representing Binary Numbers:

- ☐ **Unsigned Integers**
- ☐ **Signed Magnitude**
- ☐ **1's Complement**
- ☐ **2's Complement**

Unsigned Integers

(non-
negative
numbers)

With k bits, we
can represent 2^k
positive Integers
Ranging from
 0 to 2^k-1

Unsigned Integers	Representation
0	00000
1	00001
2	00010
3	00011
4	00100
5	00101
6	00110
7	00111
8	01000
9	01001
10	01010
11	01011
12	01100
13	01101
14	01110
15	01111
16	10000
17	10001
18	10010
19	10011
20	10100
21	10101
22	10110
23	10111
24	11000
25	11001
26	11010
27	11011
28	11100
29	11101
30	11110
31	11111

Signed Magnitude

With k bits, we can represent 2^k integers ranging from negative $2^{k-1}-1$ to positive $2^{k-1}-1$

The left most bit is a sign bit.

(0 = positive,
1 = negative)

Signed Magnitude	Representation
0	00000
1	00001
2	00010
3	00011
4	00100
5	00101
6	00110
7	00111
8	01000
9	01001
10	01010
11	01011
12	01100
13	01101
14	01110
15	01111
-0	10000
-1	10001
-2	10010
-3	10011
-4	10100
-5	10101
-6	10110
-7	10111
-8	11000
-9	11001
-10	11010
-11	11011
-12	11100
-13	11101
-14	11110
-15	11111

1's Complement

With k bits, we can represent 2^k integers ranging from negative $2^{k-1}-1$ to positive $2^{k-1}-1$ (-15 to +15)

Negative numbers are represented by taking the positive numbers and flipping all their bits.

1's Complement	Representation
0	00000
1	00001
2	00010
3	00011
4	00100
5	00101
6	00110
7	00111
8	01000
9	01001
10	01010
11	01011
12	01100
13	01101
14	01110
15	01111
-15	10000
-14	10001
-13	10010
-12	10011
-11	10100
-10	10101
-9	10110
-8	10111
-7	11000
-6	11001
-5	11010
-4	11011
-3	11100
-2	11101
-1	11110
-0	11111

2's Complement

With k bits, we can represent 2^k integers ranging from negative 2^{k-1} to positive $2^{k-1}-1$ (-16 to +15)

Negative numbers are represented by taking the positive numbers and

flipping all their bits,
then
adding 1 to it.

2's Complement	Representation
0	00000
1	00001
2	00010
3	00011
4	00100
5	00101
6	00110
7	00111
8	01000
9	01001
10	01010
11	01011
12	01100
13	01101
14	01110
15	01111
-16	10000
-15	10001
-14	10010
-13	10011
-12	10100
-11	10101
-10	10110
-9	10111
-8	11000
-7	11001
-6	11010
-5	11011
-4	11100
-3	11101
-2	11110
-1	11111

What are the advantages of one number system vs. another?

- ❑ It is a lot easier to implement computer hardware that is able to calculate numbers in 2's complement.
- ❑ Virtually all computers use the 2's complement number system to do binary arithmetic.

Part 1– E

Binary Arithmetic

Binary Arithmetic

(Using 2's complement)

- Two binary numbers can be added, starting at the rightmost bit and adding the corresponding bits.

Carry					
First number (addend)	0	0	0	1	1
Second number (augend)	+0	+0	+1	+0	+1
Sum					

- If a carry is generated, it is carried one position to the left, just as in decimal arithmetic.

Binary Arithmetic

(Using 2's complement)

- Two binary numbers can be added, starting at the rightmost bit and adding the corresponding bits.

Carry					
First number (addend)	0	0	0	1	1
Second number (augend)	+0	+0	+1	+0	+1
Sum					0

- If a carry is generated, it is carried one position to the left, just as in decimal arithmetic.

Binary Arithmetic

(Using 2's complement)

- Two binary numbers can be added, starting at the rightmost bit and adding the corresponding bits.

Carry				1	
First number (addend)	0	0	0	1	1
Second number (augend)	+0	+0	+1	+0	+1
Sum					0

- If a carry is generated, it is carried one position to the left, just as in decimal arithmetic.

Binary Arithmetic

(Using 2's complement)

- Two binary numbers can be added, starting at the rightmost bit and adding the corresponding bits.

Carry				1	
First number (addend)	0	0	0	1	1
Second number (augend)	+0	+0	+1	+0	+1
Sum				0	0

- If a carry is generated, it is carried one position to the left, just as in decimal arithmetic.

Binary Arithmetic

(Using 2's complement)

- Two binary numbers can be added, starting at the rightmost bit and adding the corresponding bits.

Carry			1	1	
First number (addend)	0	0	0	1	1
Second number (augend)	+0	+0	+1	+0	+1
Sum				0	0

- If a carry is generated, it is carried one position to the left, just as in decimal arithmetic.

Binary Arithmetic

(Using 2's complement)

- Two binary numbers can be added, starting at the rightmost bit and adding the corresponding bits.

Carry			1	1	
First number (addend)	0	0	0	1	1
Second number (augend)	+0	+0	+1	+0	+1
Sum			0	0	0

- If a carry is generated, it is carried one position to the left, just as in decimal arithmetic.

Binary Arithmetic

(Using 2's complement)

- Two binary numbers can be added, starting at the rightmost bit and adding the corresponding bits.

Carry		1	1	1	
First number (addend)	0	0	0	1	1
Second number (augend)	+0	+0	+1	+0	+1
Sum			0	0	0

- If a carry is generated, it is carried one position to the left, just as in decimal arithmetic.

Binary Arithmetic

(Using 2's complement)

- Two binary numbers can be added, starting at the rightmost bit and adding the corresponding bits.

Carry		1	1	1	
First number (addend)	0	0	0	1	1
Second number (augend)	+0	+0	+1	+0	+1
Sum	0	1	0	0	0

- If a carry is generated, it is carried one position to the left, just as in decimal arithmetic.

Binary Arithmetic

(Using 2's complement)

□ Examples:

5	00101
+4	+00100
===	-----
9	01001

7	00111
+4	+00100
===	-----
11	01011

7	00111
+7	+00111
===	-----
14	01110

Overflow in 2's Complement

- If the sum of two positive numbers carry into the last bit (left most bit), then an overflow has occurred, and the sum becomes a negative number (incorrect).

15+15 = 30 (Note that this sum produces a negative number)	01111 +01111 ----- 11110
---	-----------------------------------

Adding Negative Numbers

- In **2's complement arithmetic**, a carry generated by the addition of the leftmost bits is simply thrown away.

7+(-7) = 0 (2's complement)	000111 +111001 ----- 000000
7+(-6) = 1 (2's complement)	000111 +111010 ----- 000001
(-6)+(-6) = (-12) (2's complement)	111010 +111010 ----- 110100

Part 1-F

Hardware for Performing Arithmetic Operations.

Hardware Circuitry for Representing and Manipulating Information

□ Gates and Circuits:

- The NOT circuit
- The AND Gate
- The OR Gate
- The XOR Gate

Hardware Circuitry for Representing and Manipulating Information

□ Gates and Circuits:

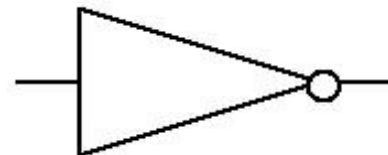
■ The NOT Gate

NOT



Input	Output
0	1
1	0

NOT



Input	Output
0	1
1	0

Hardware Circuitry for Representing and Manipulating Information

□ Gates and Circuits:

■ The AND Gate

AND



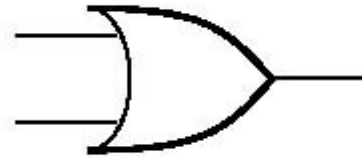
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Hardware Circuitry for Representing and Manipulating Information

□ Gates and Circuits:

■ The OR Gate

OR



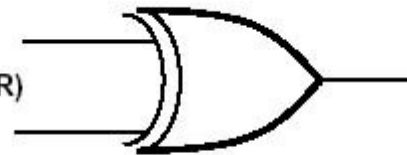
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

Hardware Circuitry for Representing and Manipulating Information

□ Gates and Circuits:

■ The XOR Gate

Exclusive OR (XOR)

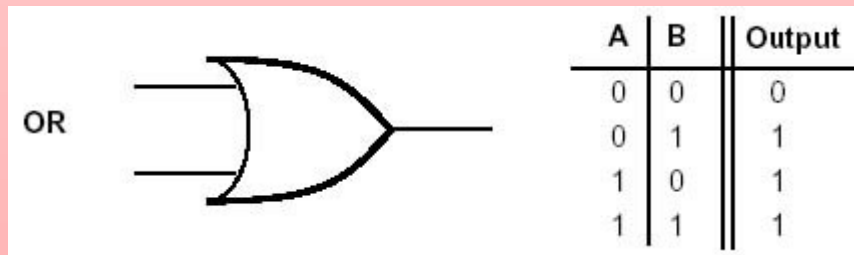


A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

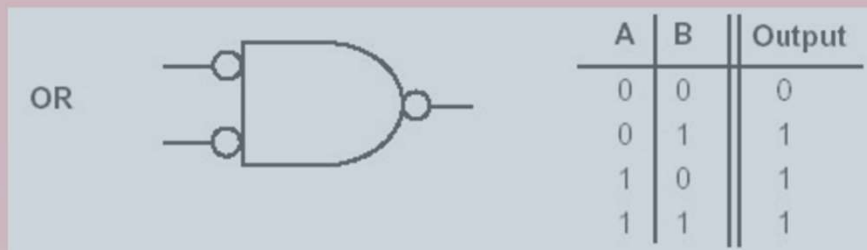
Other Circuits:

□ Gates and Circuits:

■ The OR Gate



- The OR gate can also be implemented as an AND gate with a few NOT gates:
- $A \text{ OR } B = \text{NOT} ((\text{NOT } A) \text{ AND } (\text{NOT } B))$

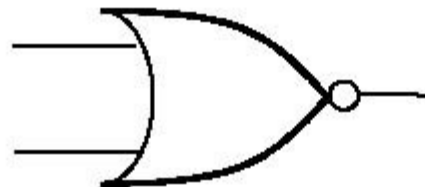


Other Circuits:

□ Gates and Circuits:

■ The NOR Gate

NOR



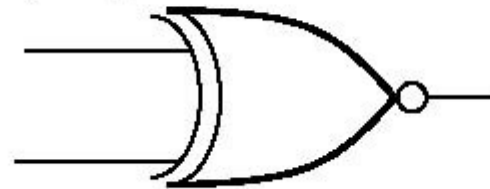
A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

Other Circuits:

□ Gates and Circuits:

■ The XNOR Gate

Exclusive NOR (XNOR)



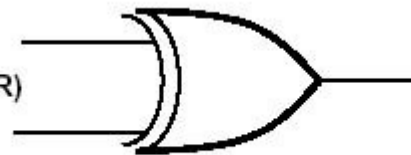
A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

Other Circuits:

□ Gates and Circuits:

■ The XOR Gate

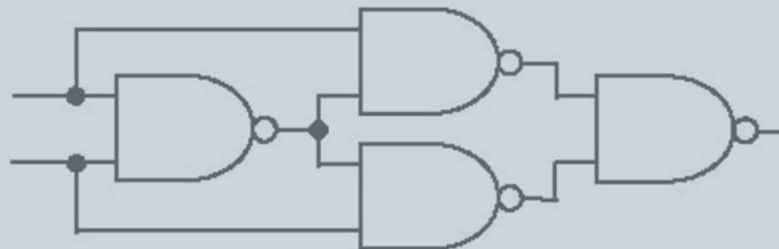
Exclusive OR (XOR)



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

■ XOR gate made with 4 NAND gates

Exclusive OR (XOR)



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Hardware Circuitry for Representing and Manipulating Information

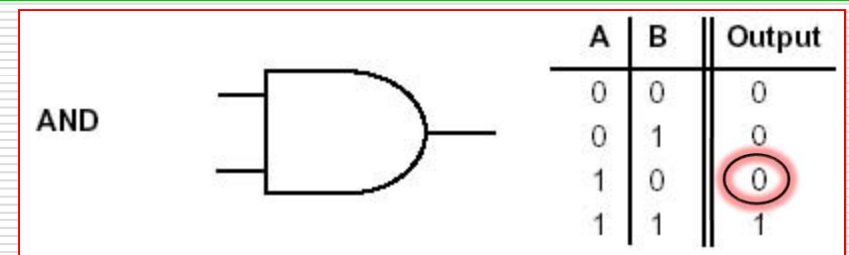
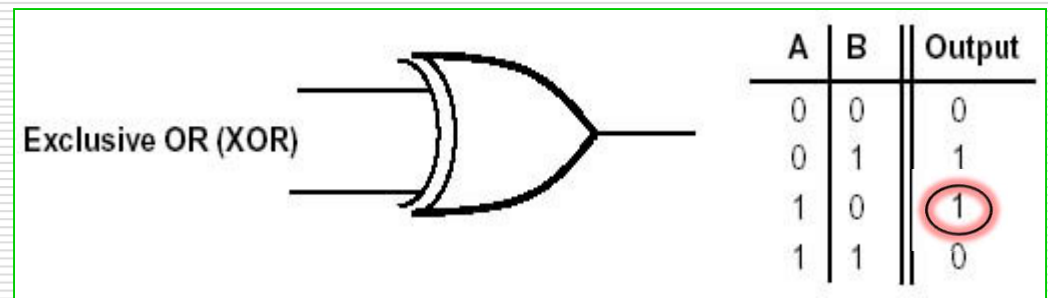
- Building an ADDer Circuit
- We proceed from the rightmost (least significant) bit position to the leftmost (most significant) bit position. In each position, we add three binary digits A, B, and Cin and as a result we get two binary digits S (Sum) and Cout.
- A and B are the bits from the two numbers we want to add. Cin is the "carry-in" from the previous bit position, and Cout is the "carry-out" to the next bit position.

```
  1 1 1 1 0 0 0 0 <--- Cin
  0 1 0 1 1 1 0 1 <--- A
+ 0 0 1 1 1 0 1 0 <--- B
-----
  1 0 0 1 0 1 1 1 <--- S
  0 1 1 1 1 0 0 0 <--- Cout
```

Hardware Circuitry for Representing and Manipulating Information

- Building an ADDer Circuit using an **XOR** and a **AND** Circuit

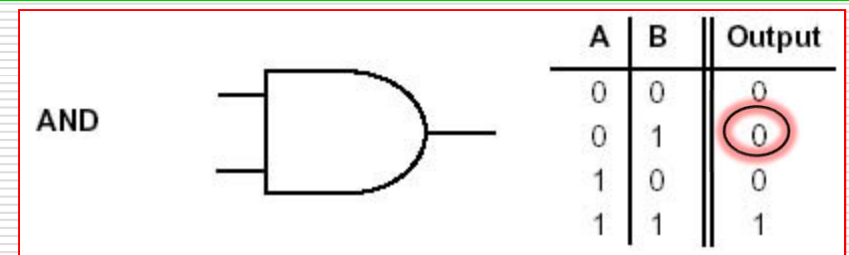
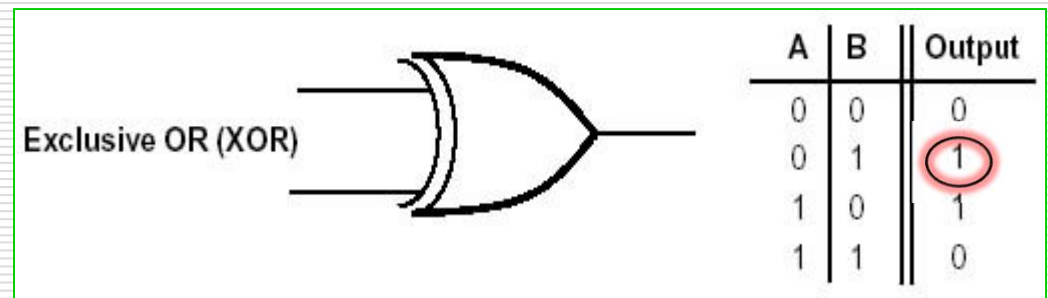
$$\begin{array}{r} 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0 <--- \text{Cin} \\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1 <--- A \\ +\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0 <--- B \\ \hline 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1 <--- S \\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0 <--- \text{Cout} \end{array}$$



Hardware Circuitry for Representing and Manipulating Information

□ Building an ADDer Circuit

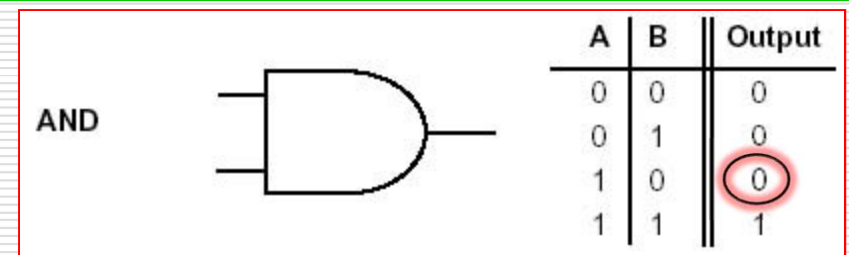
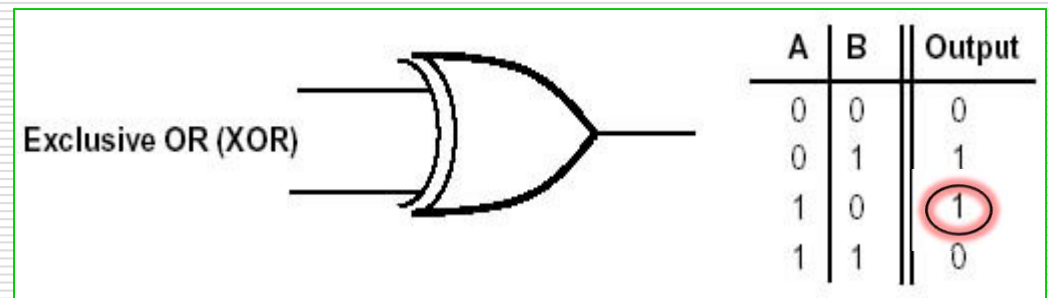
$$\begin{array}{r} 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0 <--- \text{Cin} \\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1 <--- A \\ +\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0 <--- B \\ \hline 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1 <--- S \\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0 <--- \text{Cout} \end{array}$$



Hardware Circuitry for Representing and Manipulating Information

□ Building an ADDer Circuit

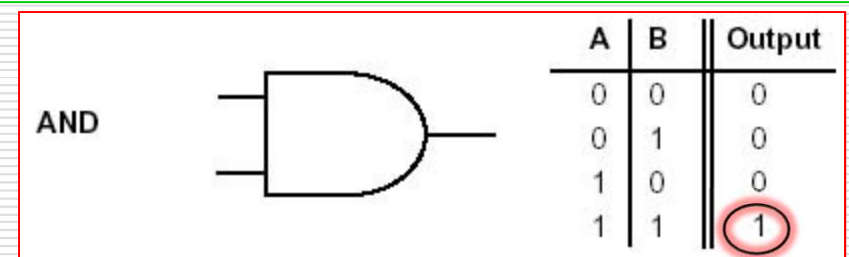
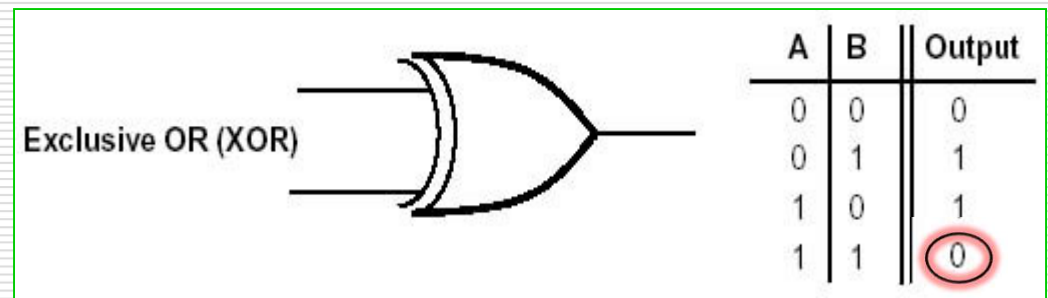
$$\begin{array}{r} 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0 <--- \text{Cin} \\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1 <--- A \\ +\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0 <--- B \\ \hline 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1 <--- S \\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0 <--- \text{Cout} \end{array}$$



Hardware Circuitry for Representing and Manipulating Information

□ Building an ADDer Circuit

$$\begin{array}{r} 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0 <--- \text{Cin} \\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1 <--- A \\ +\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0 <--- B \\ \hline 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1 <--- S \\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0 <--- \text{Cout} \end{array}$$

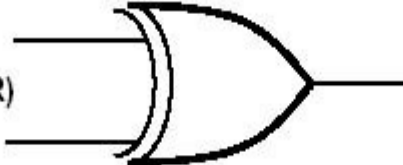


Hardware Circuitry for Representing and Manipulating Information

□ Building an ADDer Circuit

$$\begin{array}{r}
 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0 <--- \text{Cin} \\
 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1 <--- A \\
 +\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0 <--- B \\
 \hline
 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1 <--- S \\
 \\
 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0 <--- \text{Cout}
 \end{array}$$

Exclusive OR (XOR)



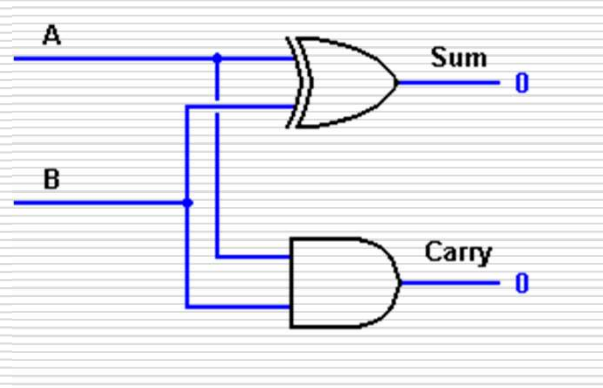
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

AND



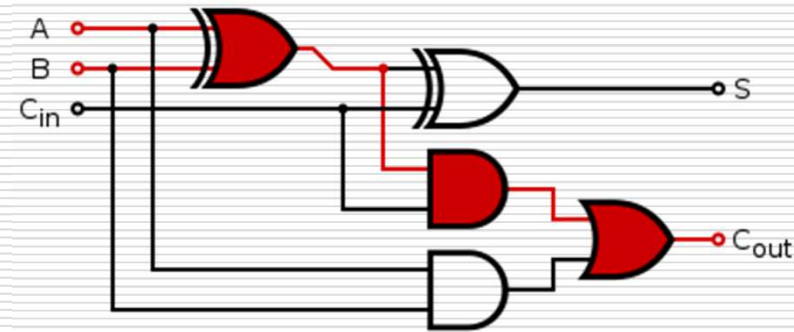
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Half Adder



Hardware Circuitry for Representing and Manipulating Information

□ A full adder



To learn more about logic gates and Boolean algebra...

See:

- ❑ <http://anh.cs.luc.edu/150/notes/hardware.pdf>
- ❑ <https://www.norwegiancreations.com/2017/10/an-intro-to-boolean-algebra-and-logic-gates-part-1/>

Part 1 – G

Encoding Floating Point Numbers

Representing Real Numbers

- How do we represent real numbers such as $2/3$ or π which may have infinite repeating or non-repeating digit sequences?
 - We approximate using Floating Point representation. Conceptually this representation is very similar to scientific notation. For example $3456 = 3.456 * 10^3$
 - Floating point numbers are generally allocated as either 32 or 64 bits. These bits are divided into 3 parts:

Sign bit	Exponent	Fraction
----------	----------	----------

- Floating point number = $(+/-) (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

Representing Real Numbers

□ Sign Bit:

- The sign bit is the very first bit of the floating-point number and determines whether the number is positive or negative.
 - 0=positive
 - 1=negative

□ Exponent:

- The exponent consists of the next 8 (32bFP) or 11 (64bFP) bits.
- The bias is a fixed number:
 - 127 for 32bFP
 - 1023 for 64bFP

Sign (1 bit)	Exponent (8 bits)	Fraction (23 bits)
0 or 1	0000 0000	000 00000 00000 00000 00000

□ The Fraction:

- The last 23 (32bFP) or 52 (64bFP) bits is the fraction.
- This is an unsigned binary string that represents binary places to the right of the decimal point and is therefore a value between 0 and 1.

Example:

Interpreting Floating Point Numbers

Sign (1 bit)	Exponent (8 bits)	Fraction (23 bits)
0 or 1	0000 0000	000 00000 00000 00000 00000

- ❑ 0 0000 0000 1100000000000000000000000000
- ❑ This fraction represents $1 \cdot 2^{-1} + 1 \cdot 2^{-2}$
- ❑ Therefore, our fraction value is $1/2 + 1/4$ or .75
- ❑ All floating-point fractions are expressed in powers of 2
- ❑ $(+/-) (1 + \text{Fraction}) \times 2^{\text{(Exponent-Bias)}}$

Example:

Interpreting Floating Point Numbers

Sign (1 bit)	Exponent (8 bits)	Fraction (23 bits)
0 or 1	0000 0000	000 00000 00000 00000 00000

□ 0 10000001 101 00000 00000 00000 00000

□ Sign bit: 0 = positive

□ Exponent: 129

□ Bias: 127

□ Fraction: $1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$

□ = $1/2 + 1/8$

□ = $.5 + .125 = .625$

Example:

Interpreting Floating Point Numbers

Sign (1 bit)	Exponent (8 bits)	Fraction (23 bits)
0 or 1	0000 0000	000 00000 00000 00000 00000

□ 0 10000001 101 00000 00000 00000 00000

□ (+/-) (1+Fraction) x 2^(Exponent-Bias)

□

□ + (1.625) x 2⁽¹²⁹⁻¹²⁷⁾

□ = 1.625 x 2²

□ = 1.625 x 4

□ = 6.5

Part 1 – H

Encoding more complex data types such as audio, video, images, or combinations of those.

Encoding Images

	red	red	blue	blue	green	green	green	black	black	black	black		
			blue	blue	green	green	green						
					green	green	green						

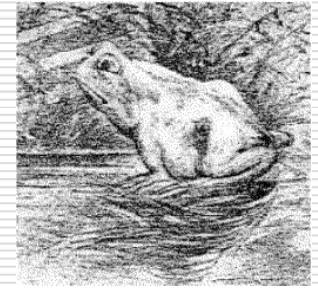
Encoding Images

	red	red	blue	blue	green	green	green	black	black	black	black		
			blue	blue	green	green	green						
					green	green	green						

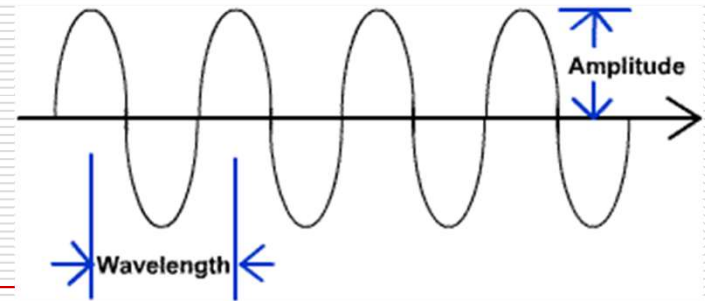
The image is represented as a 2D array and each color is represented as a number.

	53	53	79	79	32	32	32	0	0	0	0		
			79	79	32	32	32						
					32	32	32						

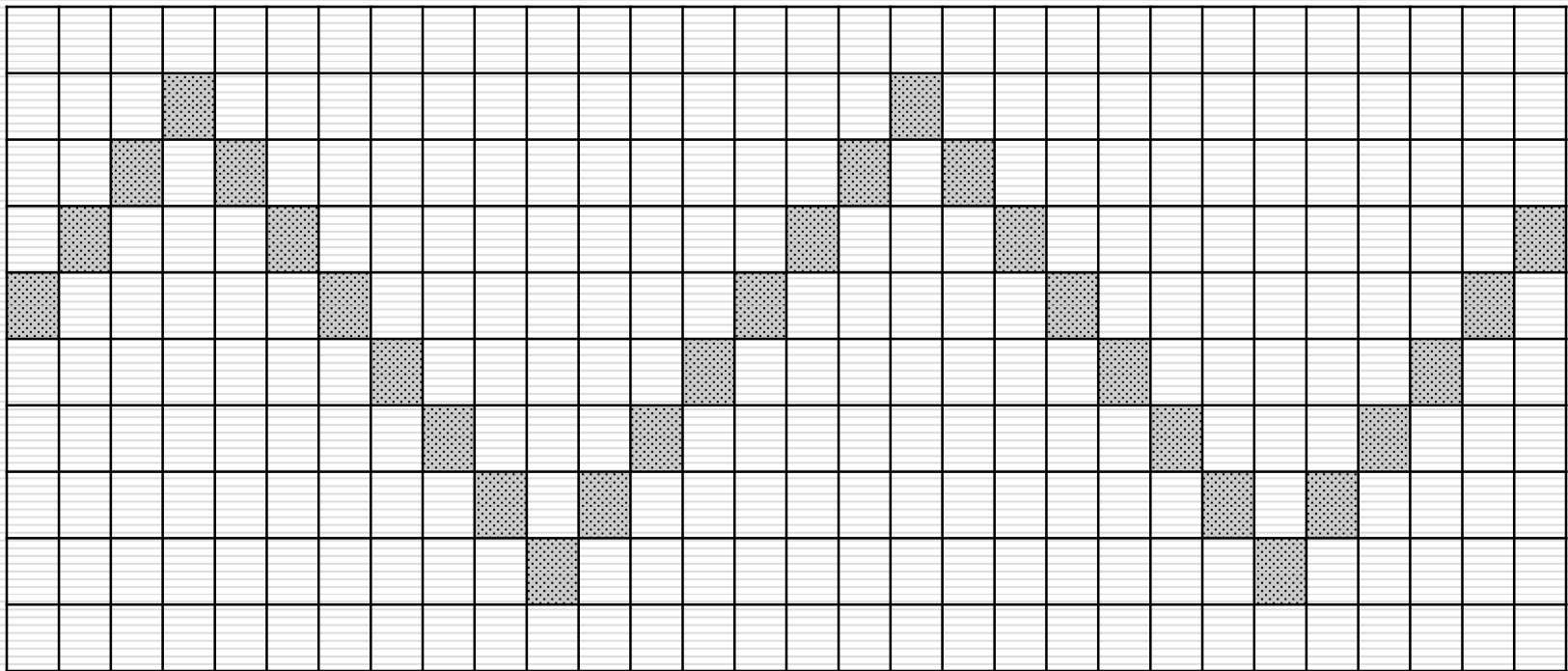
Storing images:



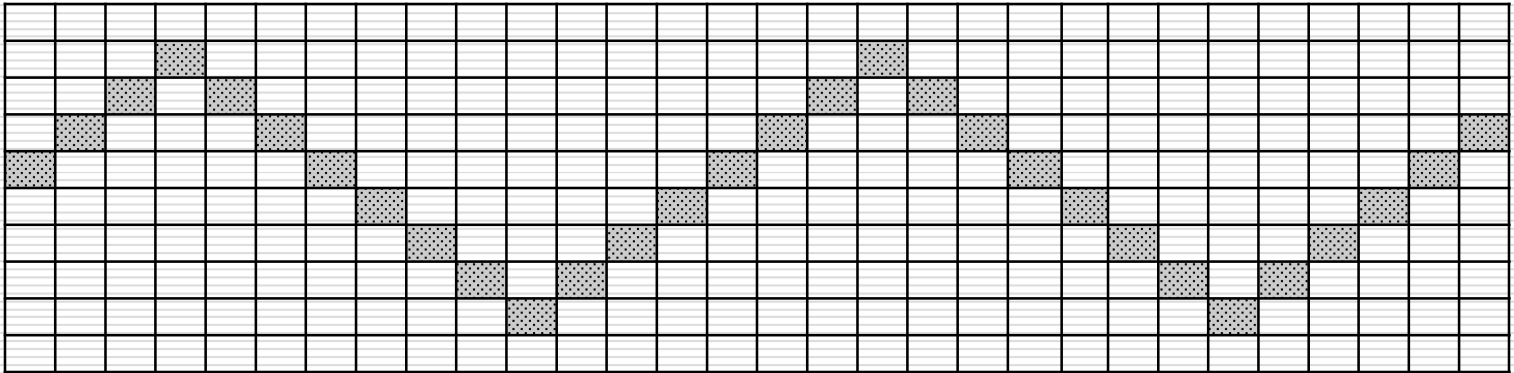
Encoding Sound



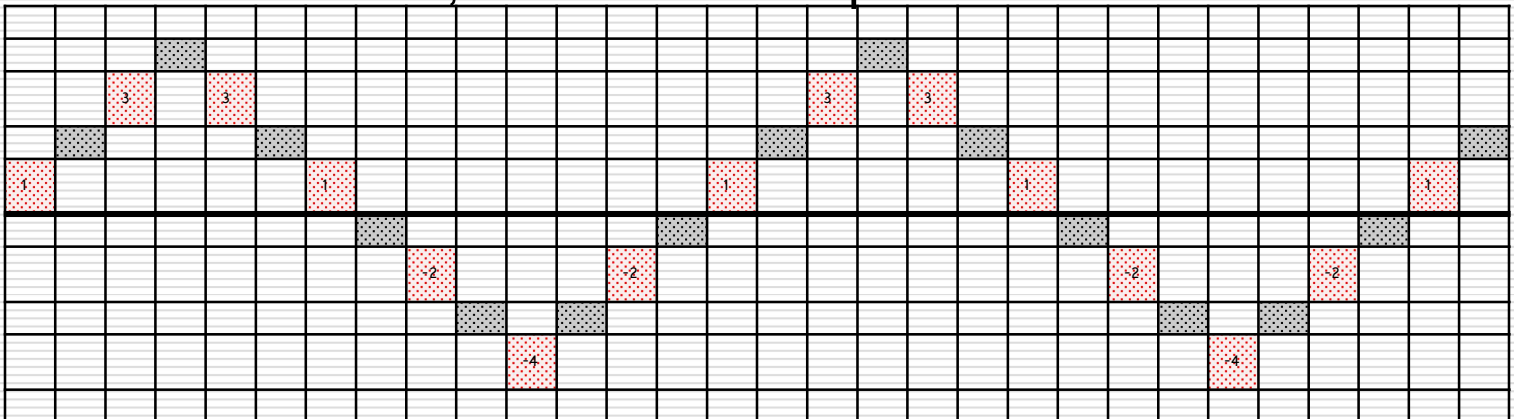
<http://www.school-for-champions.com/science/sound.htm>



Encoding Sound



In order encode sound, we have to sample the wave



Encoding of the above sound: 1,3,3,1,-2,-4,-2,1, 3,3,1, -2,-4,-2,1

Encoding Video

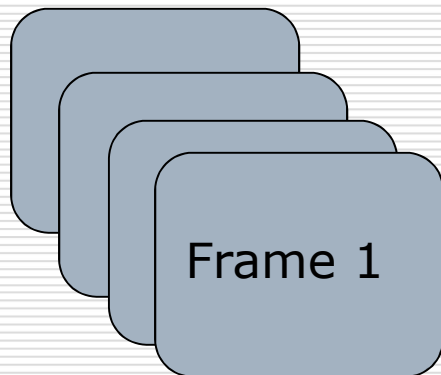
- Video can be encoded by combining images (typically 30 frames per second) plus one or more channels of sound.

Problem 1: (Video recording)

- ☐ We want to record a 4 minute 400x400 video. Assume the .bmp file has a 54 byte header, and each pixel is 16-bits (64K colors) and the frame rate is 10/sec. Audio is stereo, 16 bit samples taken at 40khz.
- ☐ What is the total size of the file?

Solution:

Video



+

Audio



Step 1: Calculate Frame size

Frame size = Header + Image resolution * Size of each pixel

Header: 54B

Pixels: $400 * 400 = 160,000$ pixels

Each Pixel (color): 2B

Each frame = $160,000 * 2B = 320,000B$

Total size for each image: $54B + 320,000B = 320,054B$

Approx: 320 KB

Step 2: Calculate Video size

Video size = Frame size * Frame rate * Duration

Video size = 320 KB * 10/sec * 60 sec/min * 4min

= 320KB * 2400

= 768,000KB

Step 3: Calculate Audio Size

Audio size = #Channels * Sample size * Sample rate * Duration

$$\begin{aligned}\text{Audio size} &= 2 * 16b * 40,000/sec * (60sec/min * 4min) \\ &= 307,200,000b * (1B/8b) \\ &= 38,400,000B * (1KB/1024B) \\ &= 37,500KB\end{aligned}$$

Step 4: Video + Audio

$$\begin{aligned}\text{File size} &= 768,000\text{KB} + 37,500\text{KB} \\ &= 805,500\text{KB} \\ &= 805 \text{ MB (approx.)}\end{aligned}$$

Watch this video about Analog vs. Digital

<https://www.youtube.com/watch?v=btgAUdbj85E>