Topic 1

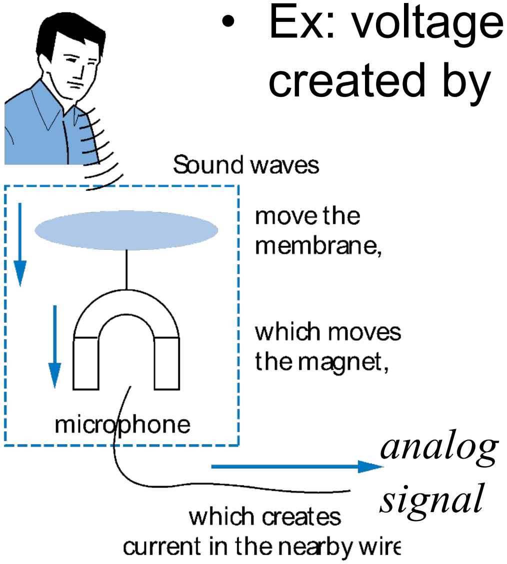
Introduction to Digital Design

Why Study Digital Circuits?

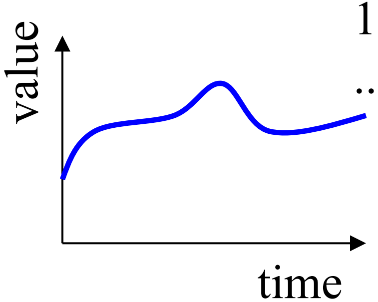
- Many elements of our lives are or will become digital
  - Computer, robotics, IoT, cell phone, TV, car, assembly line...

"Digital" vs. "Analog"

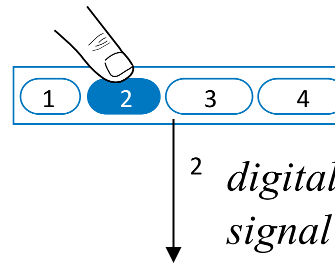
- Analog signal
  - Infinite possible values
    - Ex: voltage on a wire created by microphone



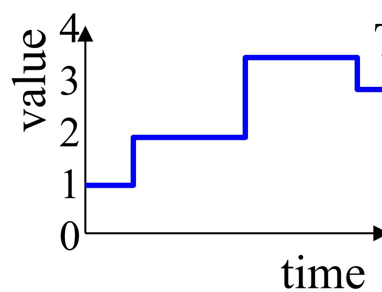
Possible values:
1.00, 1.01, 2.0000009,
... infinite possibilities



- Digital signal
  - Finite possible values
    - Ex: button pressed on a keypad

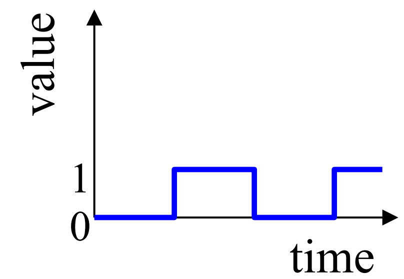


Possible values:
0, 1, 2, 3, or 4.
That's it.



Digital Signals with Only Two Values: Binary

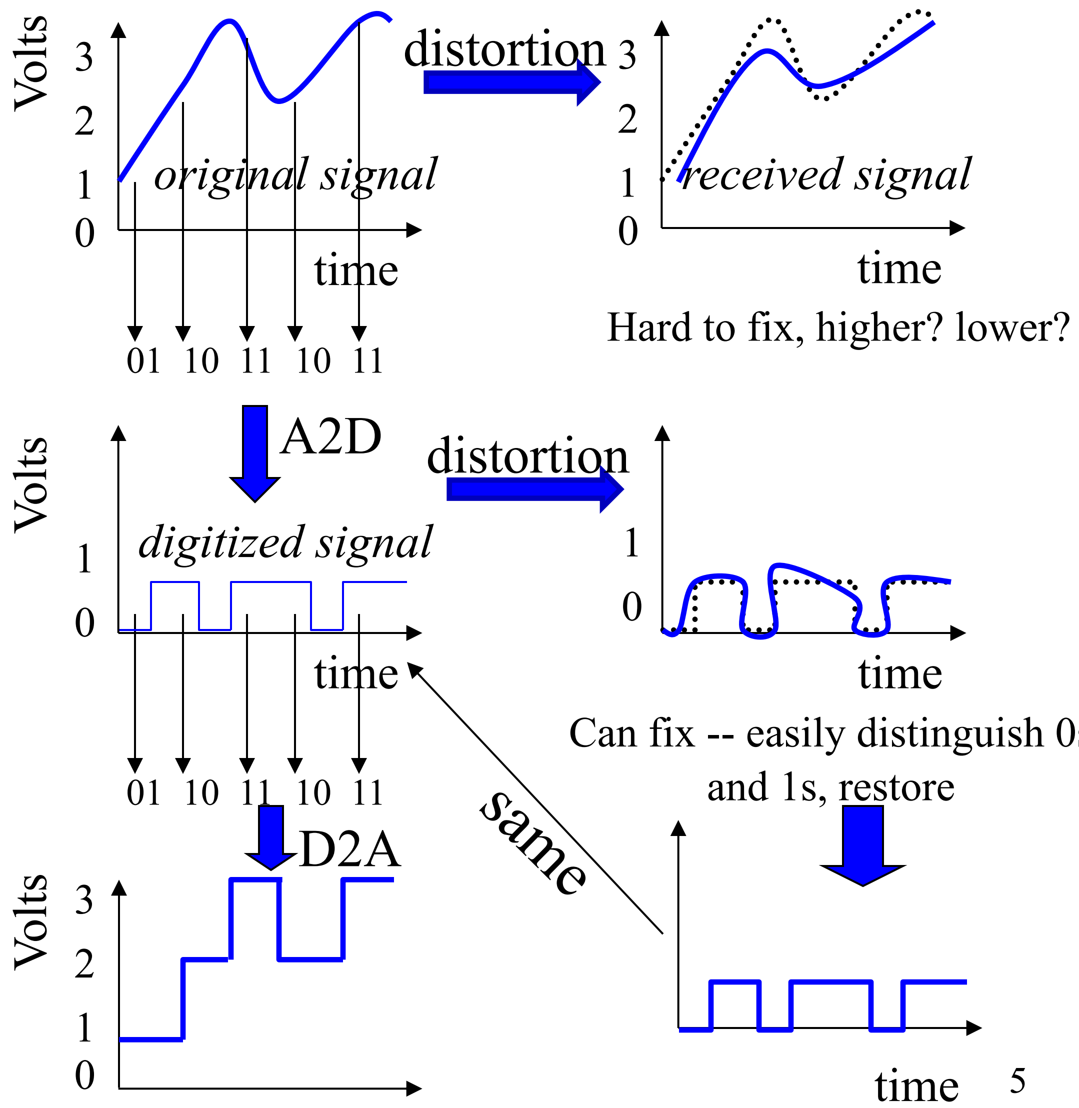
- ***Binary*** digital signal -- only *two* possible values
  - Typically represented as **0** and **1**, respectively
  - Everything is represented as combinations of 0's and 1's, e.g. 1011, 110101001
    - Called binary value or binary number
    - Each binary digit is a ***bit***
  - We'll only consider binary digital signals
    - Although there are other types of digital signals
  - Binary digital signal is popular because
    - Transistors, the basic digital electric component, operate at *two* voltages: low (e.g. 0V or -5V) and high (e.g. 3.3V or 5V)
    - Storing/transmitting one of *two* values is easier than three or more



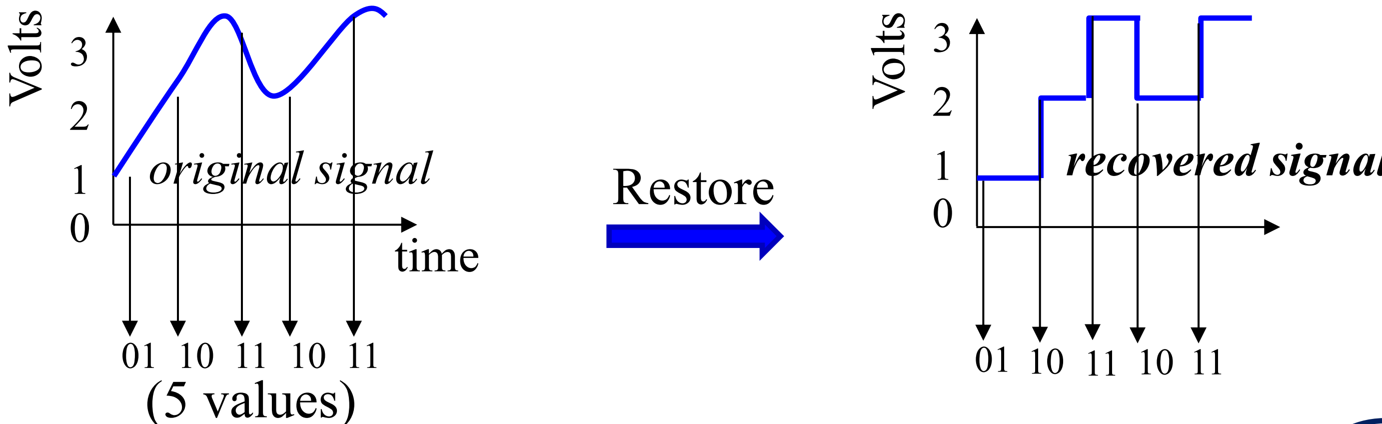
From Analog to Digital (A2D) – Digitization

- Analog signal (e.g., audio) may lose quality
  - Voltage levels not saved/copied/transmitted perfectly
  - Hard to recover
- Digitized version:
  - "Sample" voltage at particular rate
  - Easy to distinguish 0s from 1s, thus easy to recover
  - Increase sample rate to improve quality

Example:
if only 4 sampled values
let binary representation be:
0 V: "00"
1 V: "01"
2 V: "10"
3 V: "11"



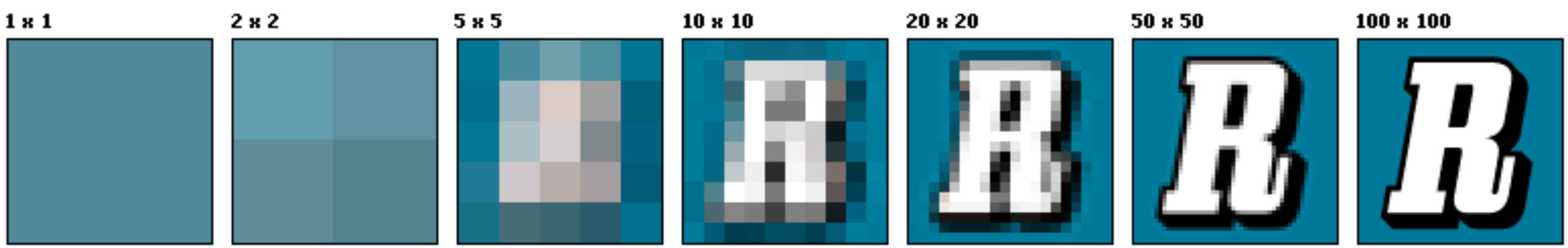
From Analog to Digital – Digitization



Issue is the Resolution

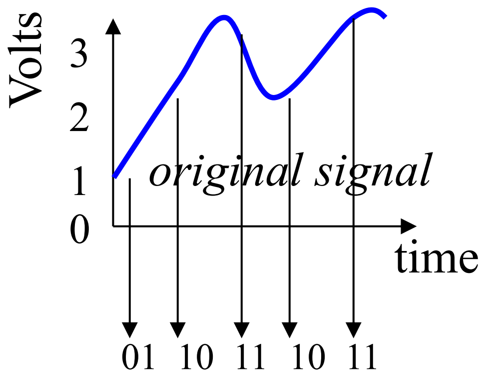
What happened?

(*source: cntv.cn*)



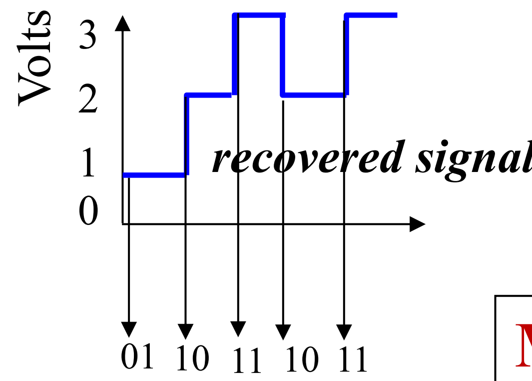
(*source: wikipedia.org*)

From Analog to Digital – Digitization

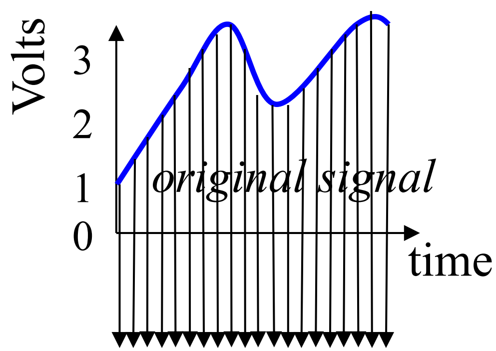


(3 different values represented with only 2 bits)

Restore

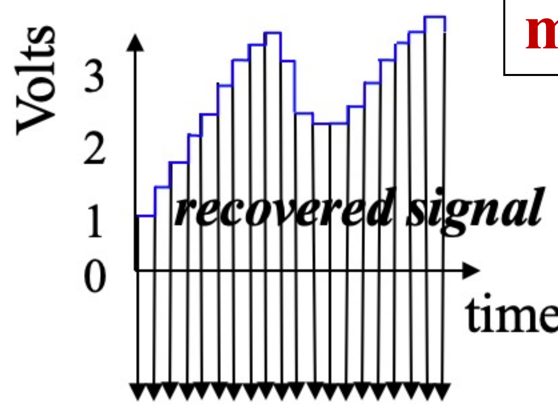


More bits are needed to represent more values



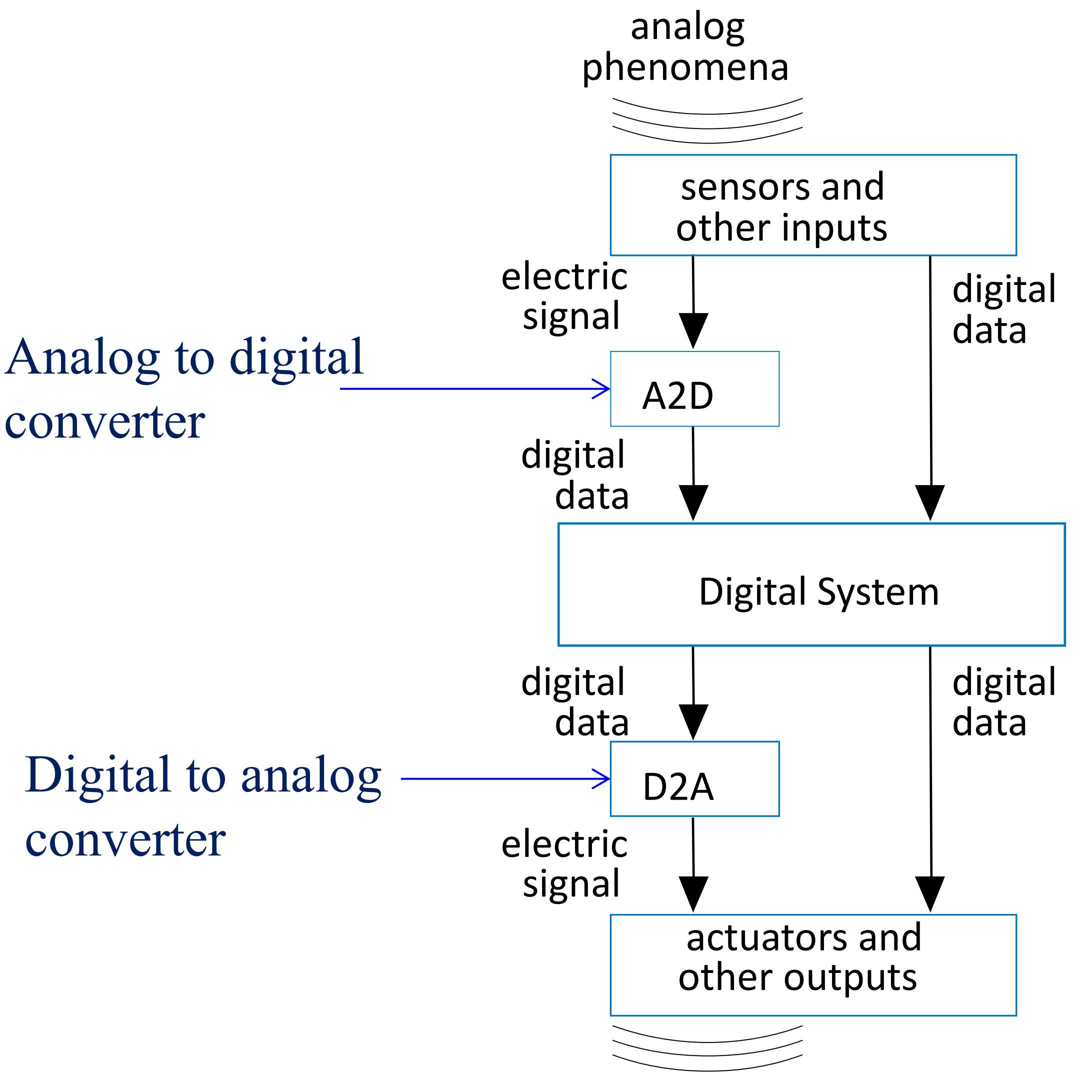
(20 different values)
Higher resolution (sample rate)

Restore



Better recovered analog signal

Typical Digital System



Then, how to represent numbers with bits?

How to Represent Numbers with Bits?

- Number systems: decimal, binary, octal, hexadecimal, …
  - Base ten (***decimal***)

$$\underline{}\ \ \underline{}\ \ \underline{5}\ \ \underline{2}\ \ \underline{3}$$
$$10^4\ \ 10^3\ \ 10^2\ \ 10^1\ \ 10^0$$

  - Base two (***binary***)

$$\underline{}\ \ \underline{}\ \ \underline{1}\ \ \underline{0}\ \ \underline{1}$$
$$2^4\ \ 2^3\ \ 2^2\ \ 2^1\ \ 2^0$$

- Each position of a number is associated with a weight quantity

Encode Numbers with Bits

- **More bits are needed to represent bigger/more numbers**
  - $37_{10} = 100101_2$ (6 bits)
  - $137_{10} = 10001001_2$ (8 bits)
  - $10307_{10} = 10100001000011_2$ (14 bits)

- N bits can represent 2^N non-negative integers
  - 0, 1, 2, …, 2^N-1
  - Negative numbers will be discussed later

Binary System

- The Binary System is a base 2 (modulo 2) number system:
  - 2 digits: 0 or 1
- Counting beyond 1 requires additional place
- In a binary number, each position has a decimal weight in power of 2, **10011.01**

	1	0	0	1	1	.	0	1
	$\times 16$	$\times 8$	$\times 4$	$\times 2$	$\times 1$		½	¼
weight	(2^{4})	(2^{3})	(2^{2})	(2^{1})	(2^{0})		(2^{-1})	(2^{-2})
position	**4**	**3**	**2**	**1**	**0**		**-1**	**-2**

Find Equivalent Decimal for Binary Numbers

- Example: Convert binary number 10011.01_2 to decimal

Number:	**1**	**0**	**0**	**1**	**1 .**	**0**	**1**
Position:	**4**	**3**	**2**	**1**	**0**	**-1**	**-2**
Weight:	$\mathbf{2^4}$	$\mathbf{2^3}$	$\mathbf{2^2}$	$\mathbf{2^1}$	$\mathbf{2^0}$	$\mathbf{2^{-1}}$	$\mathbf{2^{-2}}$

$$1\times 2^4 + 0\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0 + 0\times 2^{-1} + 1\times 2^{-2}$$

$$= 16 + 0 + 0 + 2 + 1 + 0 + 1/4$$

$$= 19.25_{10} = 10011.01_2$$

Convert Decimal to Binary Numbers:
Subtraction Method (Easy for Humans)

- Subtraction method
  - To make the job easier (especially for big numbers), we can just subtract a selected binary weight from the (remaining) quantity
    - Then, we have a new remaining quantity, and we start again (from the present binary position)
    - Stop when remaining quantity is 0

Remaining quantity: <u>12</u>

__	__	__	__	__	__	
32	16	8	4	2	1	

1	__	__	__	__	__	32 is too much
32	16	8	4	2	1	

0	1	__	__	__	__	16 is too much
32	16	8	4	2	1	

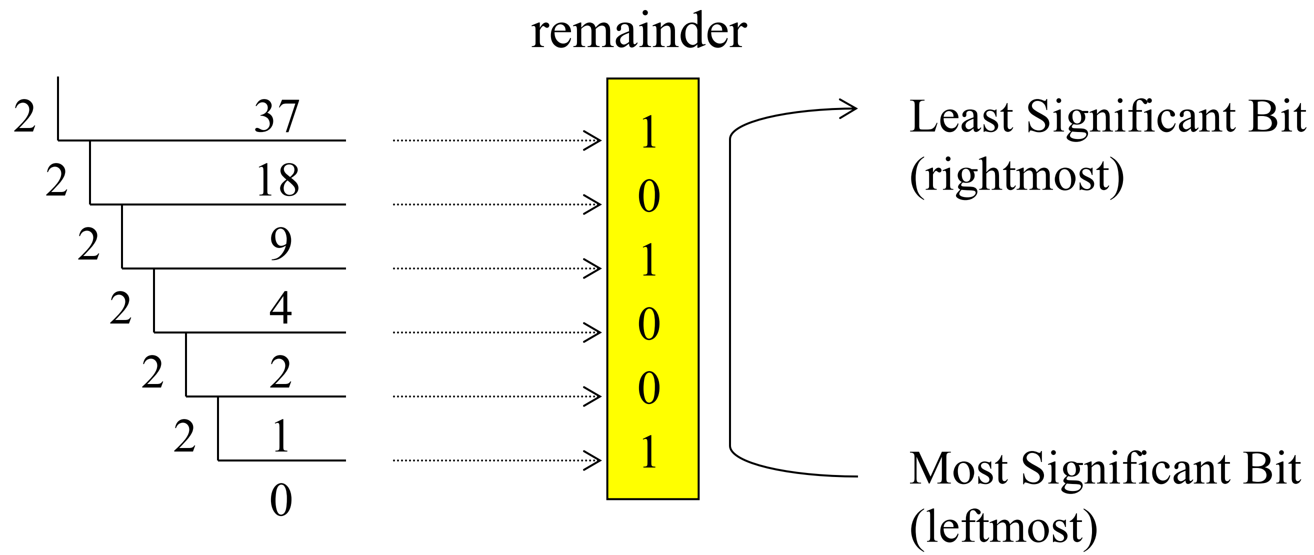
0	0	1	__	__	__	**<u>12</u> – 8 = <u>4</u>**
32	16	8	4	2	1	

0	0	1	1	__	__	**<u>4</u>-4=<u>0</u>** DONE
32	16	8	4	2	1	

0	**0**	**1**	**1**	**0**	**0**	answer
32	16	8	4	2	1	

Convert Decimal to Binary Numbers:
Division Method (Good for Computers)

- Example: Convert decimal number 37 to binary
  - Repeated-division-by-base (here, base 2)



$(37)_{10} = (100101)_2$

Convert Fractional Decimal in Binary

- Example: Convert fractional part 0.715_{10} to binary
  - Repeated-multiplication-by-base (here, base 2)

$0.715 \times 2 = 1\ .430$	1	Most Significant Bit (leftmost)
$0.430 \times 2 = 0\ .860$	0	
$0.860 \times 2 = 1\ .720$	1	
$0.720 \times 2 = 1\ .440$	1	
$0.440 \times 2 = 0\ .880$	0	Least Significant Bit (rightmost)

$(0.715)_{10} \approx (0.10110\ldots\ldots)_2$

Encode Decimal Numbers by Binary Bits

Binary				Decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15
				16
……				……

……

Hexadecimal System

- The Hexadecimal system is a base 16 (modulo 16) number system:
  - 16 digits: 0 1 2 3 4 5 6 7 8 9 A B C D E F
- Letters A ~ F represent decimal 10 through decimal 15
- Each position has a decimal weight in power of 16, e.g. **E3A**

	E	3	A
	$\times$**256**	$\times$**16**	$\times$**1**
weight	**(16^2)**	**(16^1)**	**(16^0)**

$$(E3A)_{16} = E\times 256 + 3\times 16 + A\times 1 = 3584 + 48 + 10 = 3642$$

Convert Decimal to Hexadecimal

- Example: Convert decimal number 58 to hexadecimal
  - Repeated-division-by-base (here, base 16)

16 | 58	→	10 (A)	Least Significant Digit
16 | 3	→	3	Most Significant Digit
0			

$(58)_{10} = (3A)_{16}$

Summary

Binary				Decimal	Hexaecimal
0	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**1**	**1**	**1**
0	**0**	**1**	**0**	**2**	**2**
0	**0**	**1**	**1**	**3**	**3**
0	**1**	**0**	**0**	**4**	**4**
0	**1**	**0**	**1**	**5**	**5**
0	**1**	**1**	**0**	**6**	**6**
0	**1**	**1**	**1**	**7**	**7**
1	**0**	**0**	**0**	**8**	**8**
1	**0**	**0**	**1**	**9**	**9**
1	**0**	**1**	**0**	**10**	**A**
1	**0**	**1**	**1**	**11**	**b**
1	**1**	**0**	**0**	**12**	**C**
1	**1**	**0**	**1**	**13**	**d**
1	**1**	**1**	**0**	**14**	**E**
1	**1**	**1**	**1**	**15**	**F**

Convert Hexadecimal to Binary

- Each digit is converted to 4 bits in binary
- Arrange the groups of 4 bits in the same order
- Example: convert $(3F7)_{16}$ to binary:

$$(3\ F\ 7)_{16}$$

$$(\ 0011 \quad 1111 \quad 0111\ )_2$$

Binary	Decimal	Hexaecimal
0 0 0 0	0	0
0 0 0 1	1	1
0 0 1 0	2	2
0 0 1 1	3	3
0 1 0 0	4	4
0 1 0 1	5	5
0 1 1 0	6	6
0 1 1 1	7	7
1 0 0 0	8	8
1 0 0 1	9	9
1 0 1 0	10	A
1 0 1 1	11	b
1 1 0 0	12	C
1 1 0 1	13	d
1 1 1 0	14	E
1 1 1 1	15	F

- Drop the initial 0's to simplify

$$(3F7)_{16} = (11\ 1111\ 0111)_2$$

Convert Binary to Hexadecimal

- Look for groups of 4 bits starting from the LSB
- Example: Convert 11 1011 0101.11 to hexadecimal:

$11\ 1011\ 0101.11 = (0011\ 1011\ 0101.1100)_2$

3 b 5 c

$(11\ 1011\ 0101.11)_2 = (3b5.c)_{16}$

Binary				Decimal	Hexaecimal
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	2	2
0	0	1	1	3	3
0	1	0	0	4	4
0	1	0	1	5	5
0	1	1	0	6	6
0	1	1	1	7	7
1	0	0	0	8	8
1	0	0	1	9	9
1	0	1	0	10	A
1	0	1	1	11	b
1	1	0	0	12	C
1	1	0	1	13	d
1	1	1	0	14	E
1	1	1	1	15	F

Octal System

- The Octal number system is a base 8 (modulo 8) number system:
  - 8 digits: 0 1 2 3 4 5 6 7
- Each position has a decimal weight in power of 8
- Each octal digital corresponds to a 3-bit binary number

$$(3\ 1\ 7)_8$$

$$(\mathbf{011}\quad \mathbf{001}\quad \mathbf{111})_2$$

Binary and Hexadecimal Addition

- Binary Addition

```
   11110111   carry
    10110101
  + 11010011
-------------------------
   110001000   Sum
```

- Hexadecimal Addition

```
   111    carry
   8F5A
 + 11BC
----------------------
   A116   Sum
```

How to represent texts with bits?

How to Represent Text with Bits?

- A popular code: ASCII (American Standard Code for Information Interchange)
  - 7- (or 8-) bit encoding of each letter, number, or symbol

Symbol	Encoding
R	1010010
S	1010011
T	1010100
L	1001100
N	1001110
E	1000101
0	0110000
.	0101110
<tab>	0001001

Symbol	Encoding
r	1110010
s	1110011
t	1110100
l	1101100
n	1101110
e	1100101
9	0111001
!	0100001
<space>	0100000

- Unicode: Increasingly popular 16-bit encoding
  - Encodes characters from various world languages

Question:
What does this ASCII bit sequence represent?

1010010 1000101 1010011 1010100

R E S T

ASCII Coding Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
64	01000000	100	40	@	96	01100000	140	60	`
65	01000001	101	41	A	97	01100001	141	61	a
66	01000010	102	42	B	98	01100010	142	62	b
67	01000011	103	43	C	99	01100011	143	63	c
68	01000100	104	44	D	100	01100100	144	64	d
69	01000101	105	45	E	101	01100101	145	65	e
70	01000110	106	46	F	102	01100110	146	66	f
71	01000111	107	47	G	103	01100111	147	67	g
72	01001000	110	48	H	104	01101000	150	68	h
73	01001001	111	49	I	105	01101001	151	69	i
74	01001010	112	4A	J	106	01101010	152	6A	j
75	01001011	113	4B	K	107	01101011	153	6B	k
76	01001100	114	4C	L	108	01101100	154	6C	l
77	01001101	115	4D	M	109	01101101	155	6D	m

⋮

How to represent negative numbers with bits?

How to Represent Signed Numbers with Bits?

- Cannot use minus sign, binary systems work with only two values: 0 and 1
- The left-most bit of a binary number represents the sign of a number – sign bit
  - Sign bit 0 indicates positive numbers
  - Sign bit 1 indicates negative number

Representation of Negative Numbers

- Negative numbers are represented as one of following formats
  - Sign and magnitude
  - 1’s complement code
  - 2’s complement code
- Sign and magnitude
  - MSB is the sign bit: 0 → positive, 1 → negative
- 1’s complement representation of –N
  - Negation of every bit of N
  - Example, 1’s complement representation of -3
    - N = 3 = 0011
    - -N = -3 = 1100
- 2’s complement representation of –N is
  - Negation of every bit of N, then plus 1
  - Example, 2’s complement representation of -3
    - N = 3 = 0011
    - -N = -3 = 1100 + 1 = 1101

Signed 2's Complement Number

- **Signed numbers are represented as 2's complement numbers in computers**
- **Recognize a signed 2's complement number**
  - Sign bit = 0, positive number, recognize as a regular binary number
    - **0**101 = +5;
  - Sign bit = 1, negative number, the magnitude of the number is obtained by 2's complement operation
    - **1**011
      - Sign: negative number
      - Magnitude: 2's complete operation of (1011) = 0100+1 = 0101 = 5
      - So 1011 = –5
- **Sign Extension (repeat sign bit w/o changing the value)**
  - 1011 = **1111**1011
  - 0101 = **0000**0101

Binary Addition (revisit)

- Example:

Carry bits: 11110111

    10110101

\+ 11010011

\-------------------------

$\boxed{1}$10001000

Should the Carry be brought here?

What are the signs and values of the numbers?

Binary Subtraction

- Using two's complement representation

  A – B = A + (-B)

  = A + (two's complement of B)

  = A + invert_bits of B + 1

- Example:

What are the signs and values of the numbers?

```
          10110101   (A)
        - 11010011   (B)
------------------------
```

```
   00111101
    10110101   (A)
 +  00101100   (Inversion of B)
 +         1
-----------------
   011100010
```

Should the Carry be brought here?

Binary Arithmetic

- Example:

```
    10010111
     10010101
  +  11010011
-------------------------
   101101000
```

What are the signs and values of the numbers?

Should the Carry be brought here?

Ranges of Signed 2's Complement Number

In general the 2's complement values range from -2^{n-1} to $2^{n-1}-1$

- For n = 4, the 2's complement values range from –8 to 7
- For n = 8, the 2's complement values range from –128 to 127
- For n = 16, the 2's complement values range from -2^{15} to $2^{15}-1$

- **Overflow**
  - If an n-bit 2's complement number is greater than $2^{n-1}-1$ or less than -2^{n-1}, we say there is an overflow

Detecting Overflow: Method 1

- Overflow detection logic
  - Two numbers' sign bits are the same but are different from the result's sign bit
  - If the two numbers' sign bits are different, overflow is impossible
    - Adding a positive and negative can't exceed largest magnitude positive or negative
- 4-bit example

sign bits

0 1 1 1	1 1 1 1	1 0 0 0
+ 0 0 0 1	+ 1 0 0 0	+ 0 1 1 1
1 0 0 0	0 1 1 1	1 1 1 1
overflow	overflow	no overflow
(a)	(b)	(c)

Detecting Overflow - Method 2

- Simpler method: Detect difference between carry-in to sign bit and carry-out from sign bit

$c4 \neq c3$

	(a)	(b)	(c)
carries	0 1 1 1	1 0 0 0	0 0 0 0
	0 1 1 1	1 1 1 1	1 0 0 0
+	0 0 0 1	1 0 0 0	0 1 1 1
	1 0 0 0	0 1 1 1	1 1 1 1
	overflow	overflow	no overflow
	(a)	(b)	(c)

- **If the carry into the sign bit column differs from the carry out of that column, overflow has occurred.**
- **When overflow occurs, one more bit (carry out) is needed.**