DIGITAL SYSTEM DESIGN

UNIT I: Introduction to Number Systems and Boolean Algebra

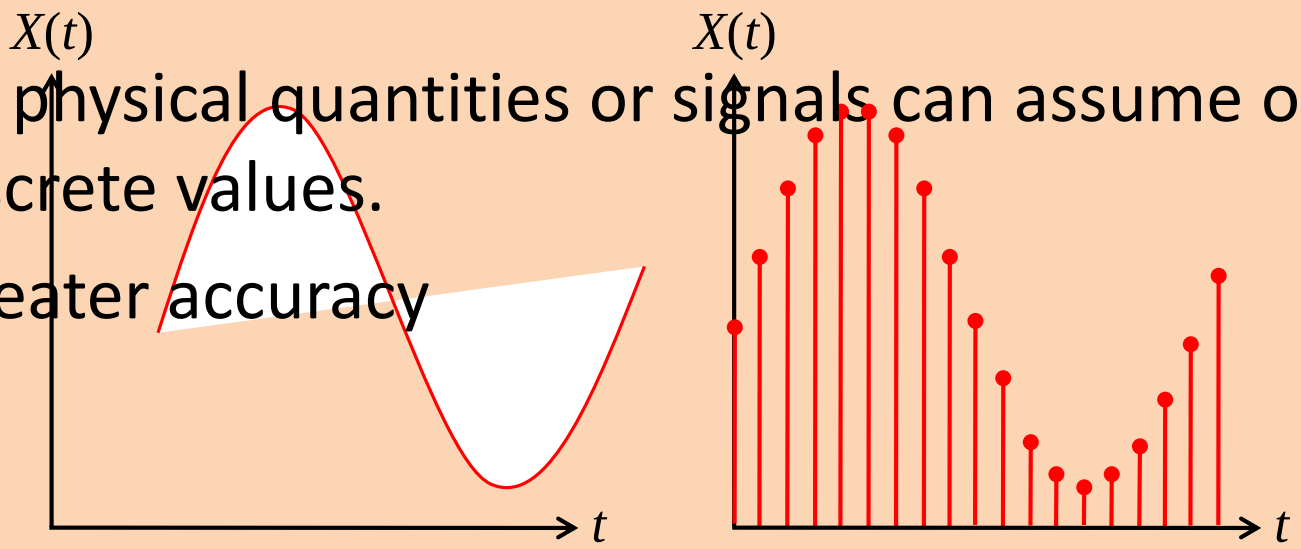
Digital and Analog Basic Concepts, Some history of Digital Systems-Introduction to number systems , Binary numbers , Number Base Conversion-Complement Codes, Binary Arithmetic , Binary codes: BCD, Weighted codes -2421,8421,gray code-Binary Logic functions, Boolean Algebra, Theorems and Properties of Boolean Algebra

Digital Systems and Binary Numbers

- Digital age and information age
- Digital computers
  - General purposes
  - Many scientific, industrial and commercial applications
- Digital systems
  - Telephone switching exchanges
  - Digital camera
  - Electronic calculators, PDA's
  - Digital TV
- Discrete information-processing systems
  - Manipulate discrete elements of information
  - For example, {1, 2, 3, …} and {A, B, C, …}…

Analog and Digital Signal

- Analog system
  - The physical quantities or signals may vary continuously over a specified range.
- Digital system
  - T
  - he physical quantities or signals can assume only discrete values.
  - Greater accuracy



Analog signal

Digital signal

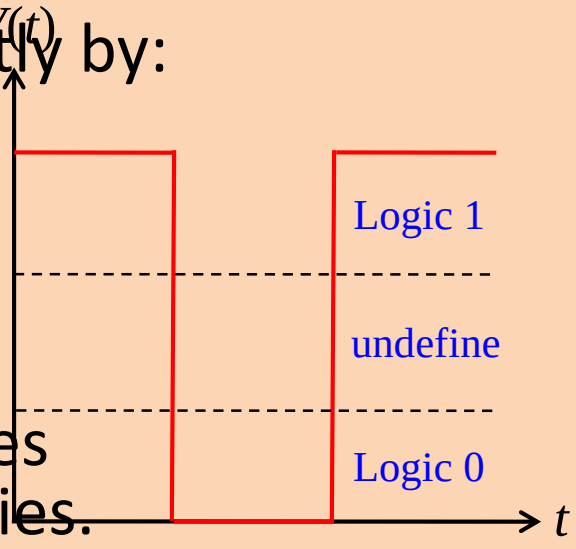
Analog		Digital
Signal	Analog signal is a continuous signal which represents physical measurements.	Digital signals are discrete time signals generated by digital modulation.
Waves	Denoted by sine waves	Denoted by square waves
Representation	Uses continuous range of values to represent information	Uses discrete or discontinuous values to represent information
Example	Human voice in air, analog electronic devices.	Computers, CDs, DVDs, and other digital electronic devices.
Technology	Analog technology records waveforms as they are.	Samples analog waveforms into a limited set of numbers and records them.
Data transmissions	Subjected to deterioration by noise during transmission and write/read cycle.	Can be noise-immune without deterioration during transmission and write/read cycle.
Response to Noise	More likely to get affected reducing accuracy	Less affected since noise response are analog in nature
Flexibility	Analog hardware is not flexible.	Digital hardware is flexible in implementation.
Uses	Can be used in analog devices only. Best suited for audio and video transmission.	Best suited for Computing and digital electronics.

history

- Abacus
- Napiers device
- Pascaline device
- Gottrfield device
- Finally Charles Babbage digital computer.

Binary Digital Signal

- An information variable represented by physical quantity.
- For digital systems, the variable takes on discrete values.
  - Two level, or binary values are the most prevalent values.
- Binary values are represented abstractly by:
  - Digits 0 and 1
  - Words (symbols) False (F) and True (T)
  - Words (symbols) Low (L) and High (H)
  - And words On and Off
- Binary values are represented by values or ranges of values of physical quantities.



Binary digital signal

Common Number Systems

- **Decimal**
- **10**
- **0, 1, … 9**

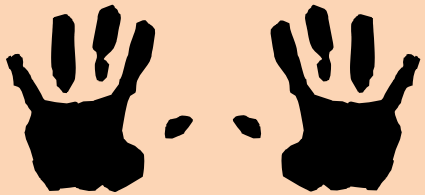
- **Binary**
- **2**
- **0, 1**

- **Octal**
- **8**
- **0, 1, … 7**

- **Hexa-decimal**
- **16**
- **0, 1, … 9,**
- **A, B, … F**

Decimal Number System

- Base (also called radix) = 10
  - 10 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
- Digit Position
  - Integer & fraction
- Digit Weight
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation



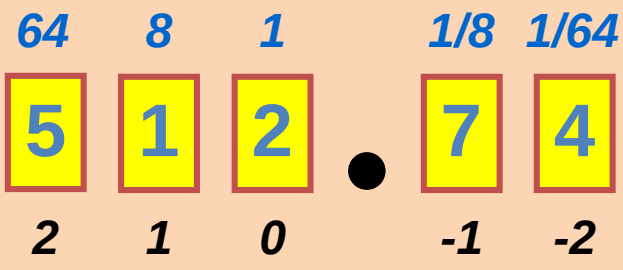
2	1	0		-1	-2
5	1	2	.	7	4
100	10	1		0.1	0.01
500	10	2		0.7	0.04

$$d_2*B^2+d_1*B^1+d_0*B^0+d_{-1}*B^{-1}+d_{-2}*B^{-2}$$

$$(512.74)_{10}$$

Octal Number System

- Base = 8
  - 8 digits { 0, 1, 2, 3, 4, 5, 6, 7 }
- Weights
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation



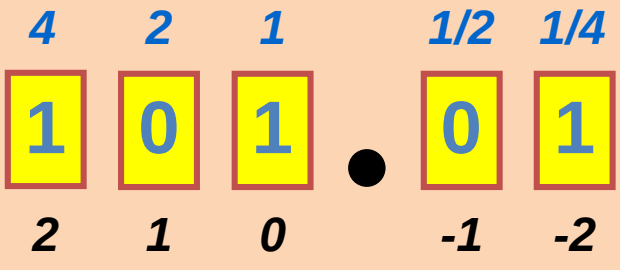
$5_2 * 8^2 + 1 * 8^1 + 2 * 8^0 + 7 * 8^{-1} + 4 * 8^{-}$

$= (330.9375)_{10}$

$(512.74)_8$

Binary Number System

- Base = 2
  - 2 digits { 0, 1 }, called *b*inary dig*its* or "*bits*"
- Weights
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Bit* x *Weight*"
- Formal Notation
- Groups of bits



$1_2 *2^2+0 *2^1+1 *2^0+0 *2^{-1}+1 *2^{-}$

$=(5.25)_{10}$

$(101.01)_2$

4 bits = *Nibble*

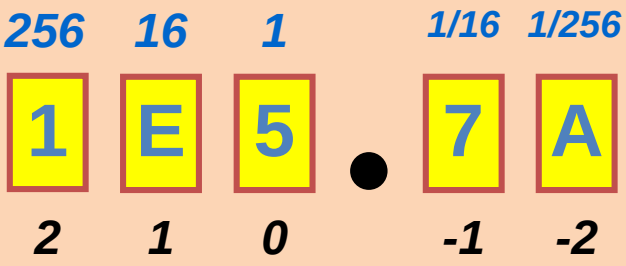
8 bits = *Byte*

1 0 1 1

1 1 0 0 0 1 0 1

Hexadecimal Number System

- Base = 16
  - 16 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F }
- Weights
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation



$1*16^{2}+14*16^{1}+5*16^{0}+7*16^{-1}+10*16^{-2}$

$=(485.4765625)_{10}$

$(1E5.7A)_{16}$

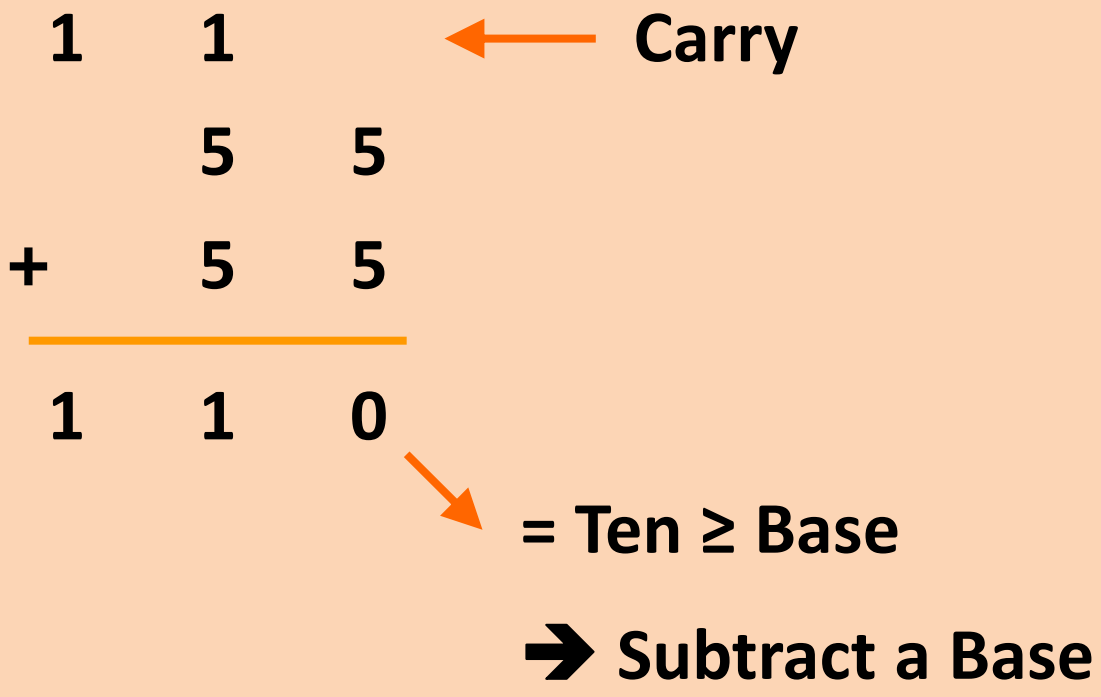
The Power of 2

n	2^n
0	$2^0=1$
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$
4	$2^4=16$
5	$2^5=32$
6	$2^6=64$
7	$2^7=128$

n	2^n	
8	$2^8=256$	
9	$2^9=512$	
10	$2^{10}=1024$	Kilo
11	$2^{11}=2048$	
12	$2^{12}=4096$	
20	$2^{20}=1M$	Mega
30	$2^{30}=1G$	Giga
40	$2^{40}=1T$	Tera

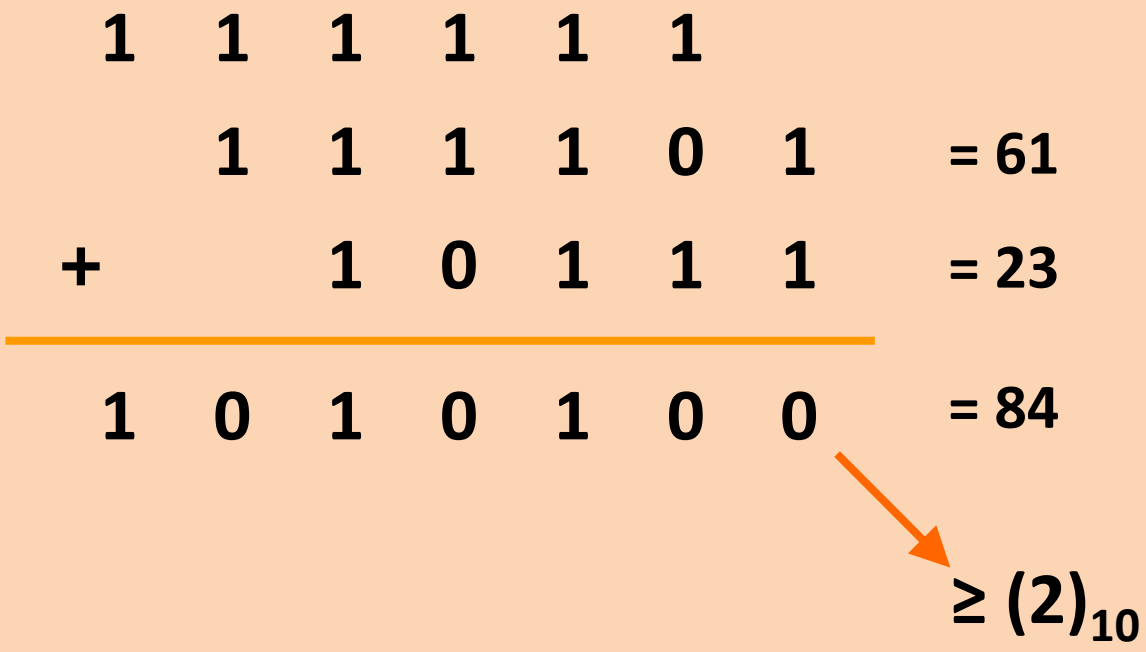
Addition

- Decimal Addition



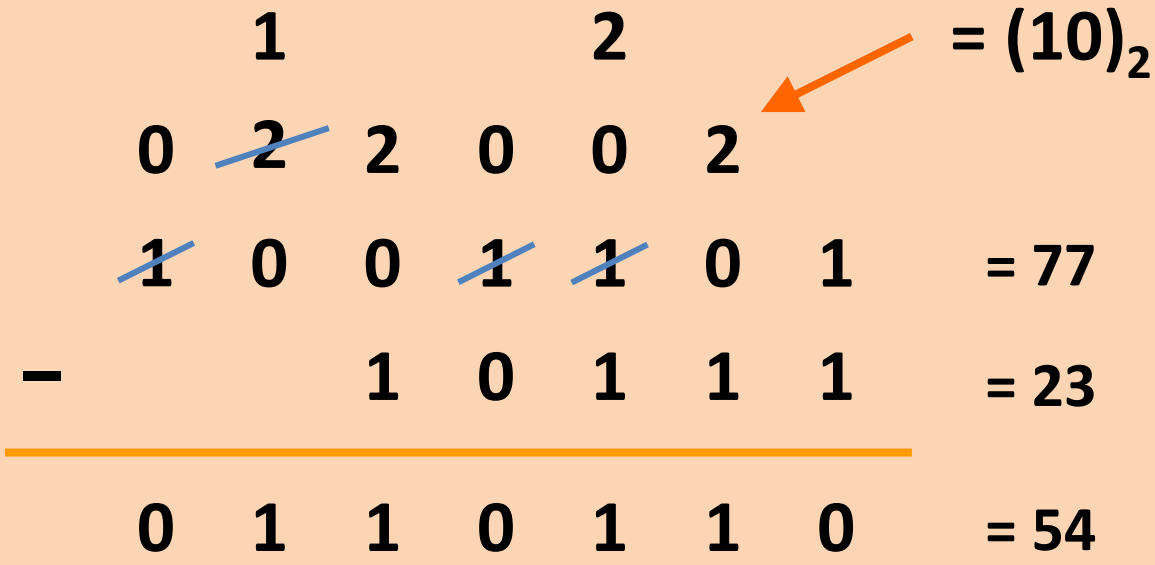
Binary Addition

- Column Addition



Binary Subtraction

- Borrow a “Base” when needed

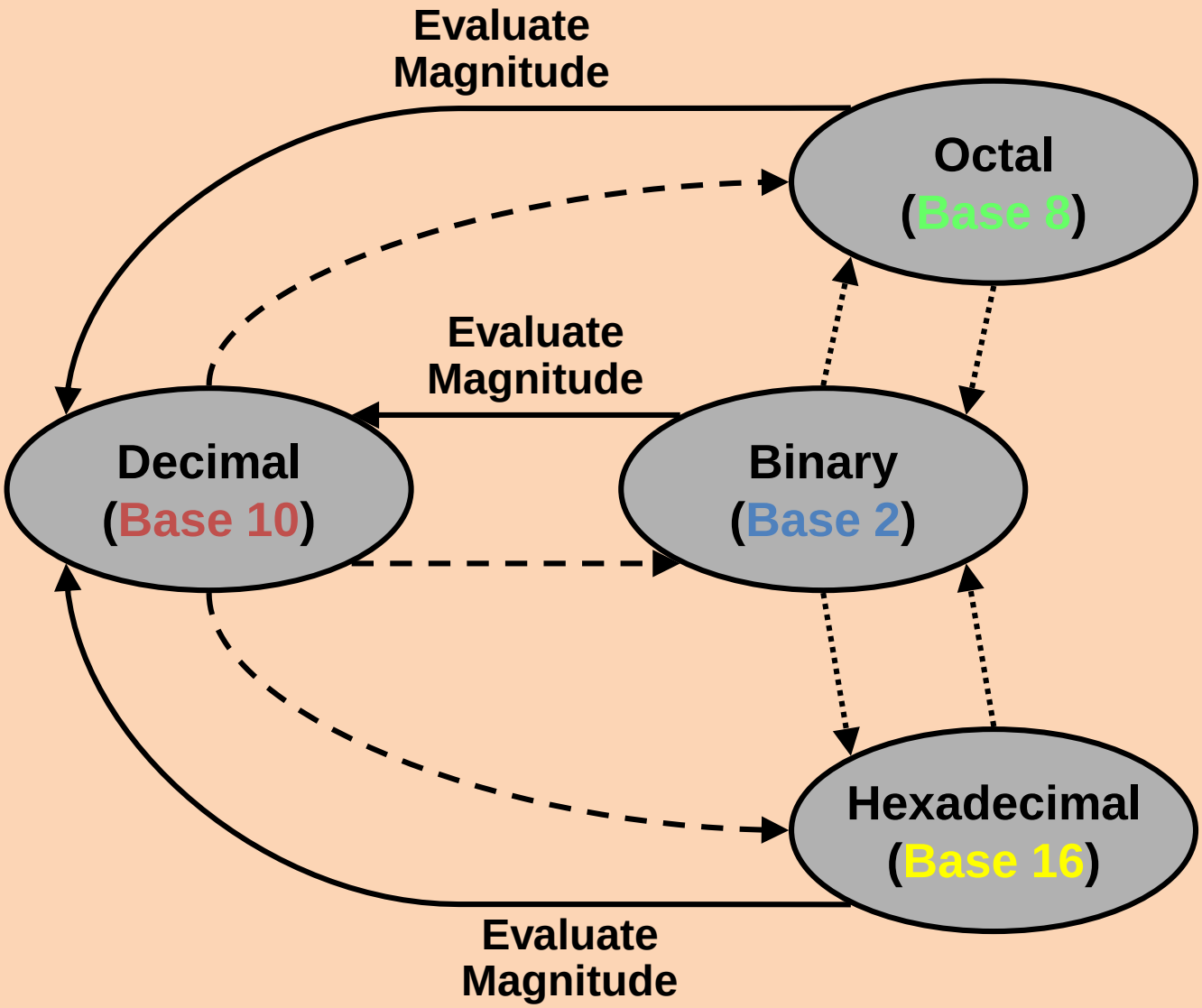


Binary Multiplication

- Bit by bit

			1	0	1	1	1
x				1	0	1	0
			0	0	0	0	0
		1	0	1	1	1	
	0	0	0	0	0		
1	0	1	1	1			
1	1	1	0	0	1	1	0

Number Base Conversions



Decimal (*Integer*) to Binary Conversion

- Divide the number by the 'Base' (=2)
- Take the remainder (either 0 or 1) as a coefficient
- Take the quotient and repeat the division

Example: $(13)_{10}$

	Quotient	Remainder	Coefficient
$13 / 2 =$	6	1	$a_0 = 1$
$6 / 2 =$	3	0	$a_1 = 0$
$3 / 2 =$	1	1	$a_2 = 1$
$1 / 2 =$	0	1	$a_3 = 1$

Answer: $(13)_{10} = (a_3\ a_2\ a_1\ a_0)_2 = (1101)_2$

MSB LSB

Decimal (*Fraction*) to Binary Conversion

- Multiply the number by the 'Base' (=2)
- Take the integer (either 0 or 1) as a coefficient
- Take the resultant fraction and repeat the division

Example: $(0.625)_{10}$

		Integer		Fraction	Coefficient
0.625	* 2 =	1	.	25	$a_{-1} = 1$
0.25	* 2 =	0	.	5	$a_{-2} = 0$
0.5	* 2 =	1	.	0	$a_{-3} = 1$

Answer: $(0.625)_{10} = (0.a_{-1}\, a_{-2}\, a_{-3})_2 = (0.101)_2$

MSB LSB

Decimal to Octal Conversion

Example: $(175)_{10}$

		Quotient	Remainder	Coefficient
175	/ 8 =	21	7	$a_0 = 7$
21	/ 8 =	2	5	$a_1 = 5$
2	/ 8 =	0	2	$a_2 = 2$

Answer: $(175)_{10} = (a_2\ a_1\ a_0)_8 = (257)_8$

Example: $(0.3125)_{10}$

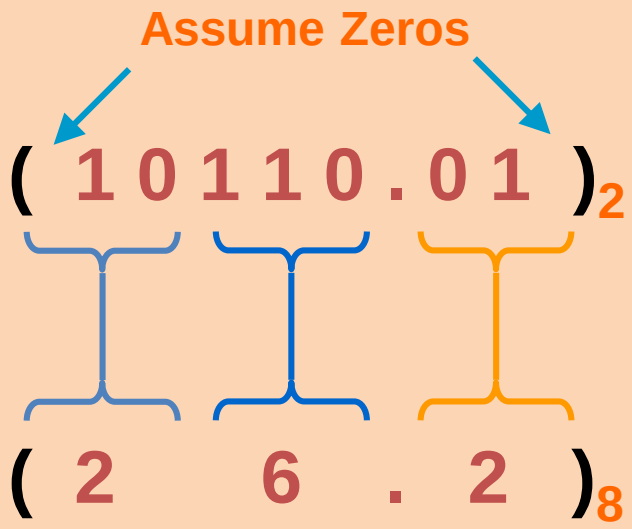
		Integer		Fraction	Coefficient
0.3125	* 8 =	2	.	5	$a_{-1} = 2$
0.5	* 8 =	4	.	0	$a_{-2} = 4$

Answer: $(0.3125)_{10} = (0.a_{-1}\ a_{-2}\ a_{-3})_8 = (0.24)_8$

Binary – Octal Conversion

- $8 = 2^3$
- Each group of 3 bits represents an octal digit

Example:



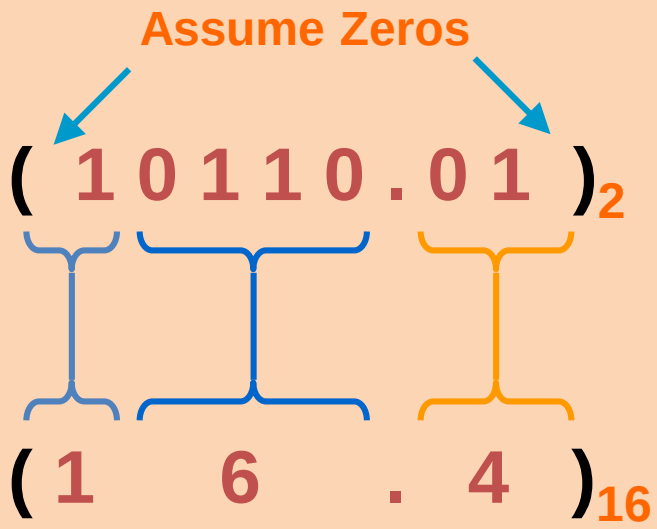
Octal	Binary
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

Works both ways (Binary to Octal & Octal to Binary)

Binary – Hexadecimal Conversion

- $16 = 2^4$
- Each group of 4 bits represents a hexadecimal digit

Example:



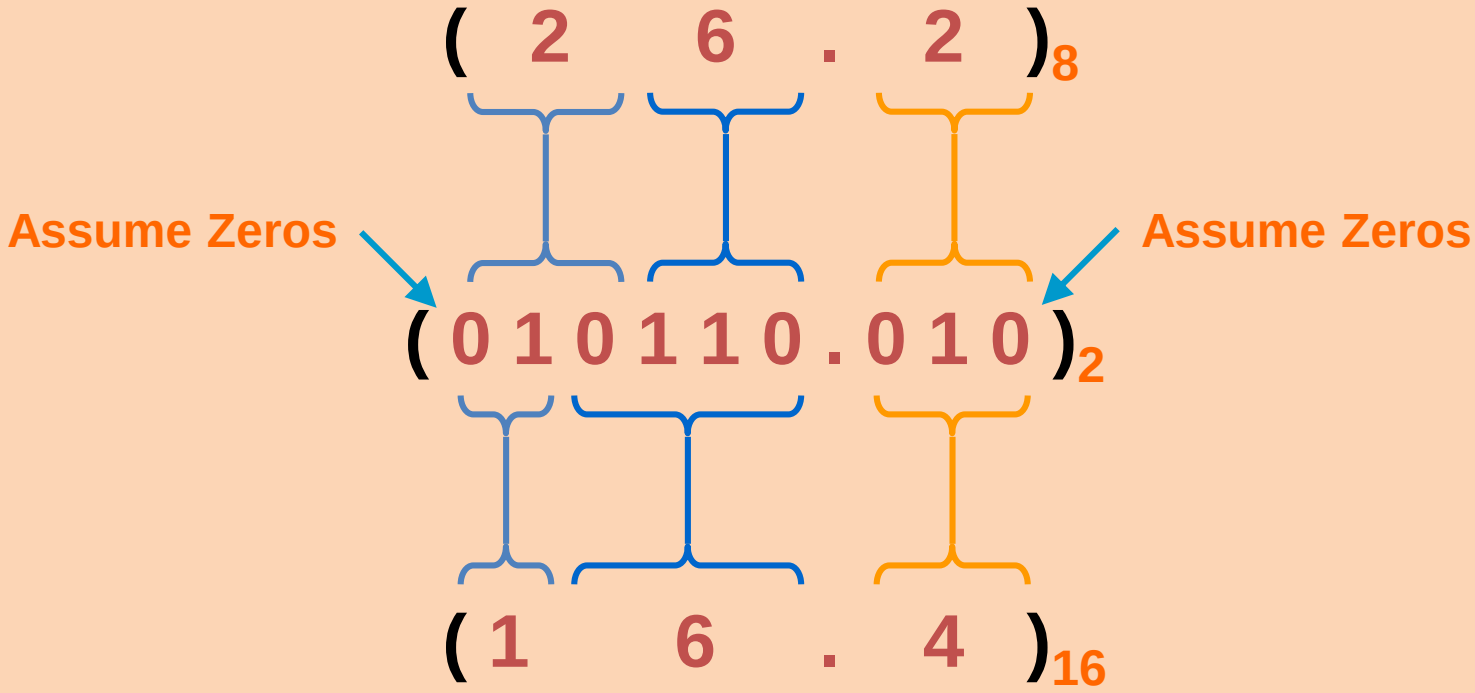
Hex	Binary
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
A	1 0 1 0
B	1 0 1 1
C	1 1 0 0
D	1 1 0 1
E	1 1 1 0
F	1 1 1 1

Works both ways (Binary to Hex & Hex to Binary)

Octal – Hexadecimal Conversion

- Convert to Binary as an intermediate step

Example:



Works both ways (Octal to Hex & Hex to Octal)

Decimal, Binary, Octal and Hexadecimal

Decimal	Binary	Octal	Hex
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Complements

- There are two types of complements for each base-*r* system: the radix complement and diminished radix complement.
- **Diminished Radix Complement - (r-1)'s Complement**
  - Given a number *N* in base *r* having *n* digits, the (*r–1*)'s complement *of N* is defined as:

    $(r^n - 1) - N$
- **Example for 6-digit <u>decimal</u> numbers**:
  - 9's complement is $(r^n - 1) - N = (10^6 - 1) - N = 999999 - N$
  - 9's complement of 546700 is 999999–546700 = 453299
- **Example for 7-digit <u>binary</u> numbers:**
  - 1's complement is $(r^n - 1) - N = (2^7 - 1) - N = 1111111 - N$
  - 1's complement of 1011000 is 1111111–1011000 = 0100111
- **Observation:**
  - Subtraction from $(r^n - 1)$ will never require a borrow
  - Diminished radix complement can be computed digit-by-digit
  - For binary: 1 – 0 = 1 and 1 – 1 = 0

Complements

- 1's Complement (*Diminished Radix* Complement)
  - All '0's become '1's
  - All '1's become '0's

  Example $(10110000)_2$

  ⇨ $(01001111)_2$

  If you add a number and its 1's complement …

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

Complements

- Radix Complement

The r's complement of an n-digit number N in base r is defined as $r^n - N$ for $N \neq 0$ and as 0 for $N = 0$. Comparing with the $(r - 1)$'s complement, we note that the r's complement is obtained by adding 1 to the $(r - 1)$'s complement, since $r^n - N = [(r^n - 1) - N] + 1$.

Example: Base-10

The 10's complement of 012398 is 987602
The 10's complement of 246700 is 753300

Example: Base-2

The 2's complement of 1101100 is 0010100
The 2's complement of 0110111 is 1001001

Complements

- 2's Complement (*Radix* Complement)
  - Take 1's complement then add 1

OR

  - Toggle all bits to the left of the first '1' from the right

Example:

Number: 1 0 1 1 0 0 0 0 | 1 0 1 1 0 0 0 0

1's Comp.: 0 1 0 0 1 1 1 1

\+ 1

---

0 1 0 1 0 0 0 0 | 0 1 0 1 0 0 0 0

Complements

- Subtraction with Complements
  - The subtraction of two *n*-digit unsigned numbers

1. Add the minuend M to the r's complement of the subtrahend N. Mathematically, $M + (r^n - N) = M - N + r^n$.
2. If $M \geqq N$, the sum will produce and end carry r^n, which can be discarded; what is left is the result $M - N$.
3. If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the r's complement of $(N - M)$. To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

Complements

- Example 1.5
  - , subtract 72532 – 3250.

	$M =$	72532
10's complement of	$N =$	$+96750$
	Sum =	169282
	Discard end carry $10^5 =$	-100000
	Answer =	69282

Example 1.6

- Using 10's complement, subtract 3250 – 72532.

	M =	03250
10's complement of	N =	$+27468$
	Sum =	30718

There is no end carry.

Therefore, the answer is – (10's complement of 30718) = – 69282.

Complements

Example 1.7

– Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction (a) $X - Y$; and

(a)

X =	1010100
2's complement of Y =	+0111101
Sum =	10010001
Discard end carry 2^7 =	−10000000
Answer. X − Y =	0010001

(b)

Y =	1000011
2's complement of X =	+ 0101100
Sum =	1101111

There is no end carry. Therefore, the answer is Y – X = – (2's complement of 1101111) = – 0010001.

Complements

- Subtraction of unsigned numbers can also be done by means of the $(r-1)$'s complement. Remember that the $(r-1)$'s complement is one less then the r's complement.
- Example 1.8
  - Repeat Example 1.7, but this time using 1's complement.

(a) $X - Y = 1010100 - 1000011$

$X =$	1010100
1's complement of $Y =$	± 0111100
Sum =	10010000
End-around carry =	+ 1
Answer. $X - Y =$	0010001

(b) $Y - X = 1000011 - 1010100$

$Y =$	1000011
1's complement of $X =$	+ 0101011
Sum =	1101110

There is no end carry, Therefore, the answer is Y – X = – (1's complement of 1101110) = – 0010001.

Signed Binary Numbers

- To represent negative integers, we need a notation for negative values.
- It is customary to represent the sign with a bit placed in the leftmost position of the number since binary digits.
- The convention is to make the sign bit 0 for positive and 1 for negative.
- Example:

Signed-magnitude representation:	10001001
Signed-1's-complement representation:	11110110
Signed-2's-complement representation:	11110111

- Table 1.3 lists all possible four-bit signed binary numbers in the three representations.

Signed Binary Numbers

Table 1.3
Signed Binary Numbers

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

Signed Binary Numbers

- Arithmetic addition
  - The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmetic. **If the signs are the same**, we add the two magnitudes and give the sum the common sign. **If the signs are different**, we subtract the smaller magnitude from the larger and give the difference the sign if the larger magnitude.
  - The addition of two signed binary numbers with negative numbers represented in signed-2's-complement form is obtained from the addition of the two numbers, including their sign bits.
  - A carry out of the sign-bit position is discarded.
- Ex

+ 6	00000110	− 6	11111010
+13	00001101	+13	00001101
+ 19	00010011	+ 7	00000111
+ 6	00000110	− 6	11111010
−13	11110011	−13	11110011
− 7	11111001	− 19	11101101

Signed Binary Numbers

- Arithmetic Subtraction
  - 1. Take the 2's complement of the subtrahend (including the sign bit) and add it to the minuend (including sign bit).
    2. A carry out of sign-bit position is discarded.

$$(\pm A) - (+B) = (\pm A) + (-B)$$
$$(\pm A) - (-B) = (\pm A) + (+B)$$

- Example:

(– 6) – (– 13) → (11111010 – 11110011)

→ (11111010 + 00001101)

→ 00000111 (+ 7)

Binary Codes

- BCD Code
  - A number with k decimal digits will require 4k bits in BCD.
  - Decimal 396 is represented in BCD with 12bits as 0011 1001 0110, with each group of 4 bits representing one decimal digit.
  - A decimal number in BCD is the same as its equivalent binary number only when the number is between 0 and 9.
  - The binary combinations 1010 through 1111 are not used and have no meaning in BCD.

Table 1.4
Binary-Coded Decimal (BCD)

Decimal Symbol	BCD Digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

The Digital Codes

Binary Coded Decimal (BCD)

- Would it be easy for you if you can replace a decimal number with an individual binary code?
  - Such as $00011001 = 19_{10}$
- The 8421 code is a type of BCD to do that.
- BCD code provides an excellent interface to binary systems:
  - Keypad inputs
  - Digital readouts

Binary Coded Decimal

Decimal Digit	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Note: 1010, 1011, 1100, 1101, 1110, and 1111 are **INVALID CODE!**

Let's crack these…

ex1: dec-to-BCD

(a) 35

(b) 98

(c) 170

(d) 2469

ex2: BCD-to-dec

(a) 10000110

(b) 001101010001

(c) 1001010001110000

BCD Addition

- BCD is a numerical code and can be used in arithmetic operations. Here is how to add two BCD numbers:
  - Add the two BCD numbers, using the rules for basic binary addition.
  - If a 4-bit sum is equal to or less than 9, it is a valid BCD number.
  - If a 4-bit sum > 9, or if a carry out of the 4-bit group is generated it is an invalid result. Add 6 (0110) to a 4-bit sum in order to skip the six the invalid states and return the code to 8421. If a carry results when 6 is added, simply add the carry to the next 4-bit group.

BCD Addition

- Try these:

  ex: Add the following numbers

```
(a) 0011+0100
(b) 00100011 + 00010101
(c) 10000110 + 00010011
(d) 010001010000 + 010000010111
(e) 1001 + 0100
(f) 1001 + 1001
(g) 00010110 + 00010101
(h) 01100111 + 01010011
```

The Gray Code

- The Gray code is unweighted and is not an arithmetic code.
  - There are no specific weights assigned to the bit positions.
- Important: the Gray code exhibits only a single bit change from one code word to the next in sequence.
  - This property is important in many applications, such as shaft position encoders.

The Gray Code

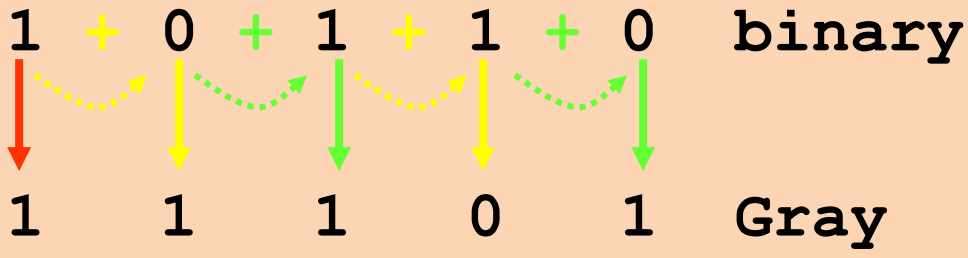
Decimal	Binary	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100

Decimal	Binary	Gray Code
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

The Gray Code

- Binary-to-Gray code conversion
  - The MSB in the Gray code is the same as corresponding MSB in the binary number.
  - Going from left to right, add each adjacent pair of binary code bits to get the next Gray code bit. Discard carries.

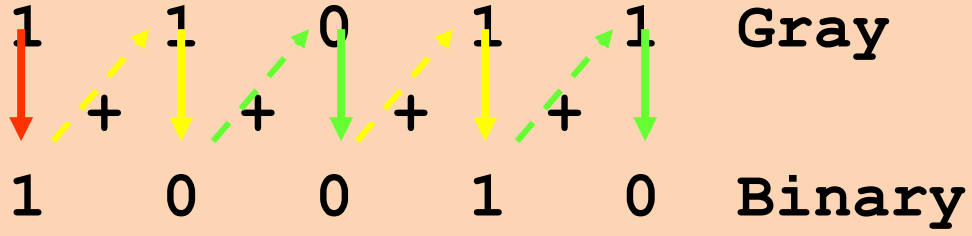
ex: convert 10110_2 to Gray code



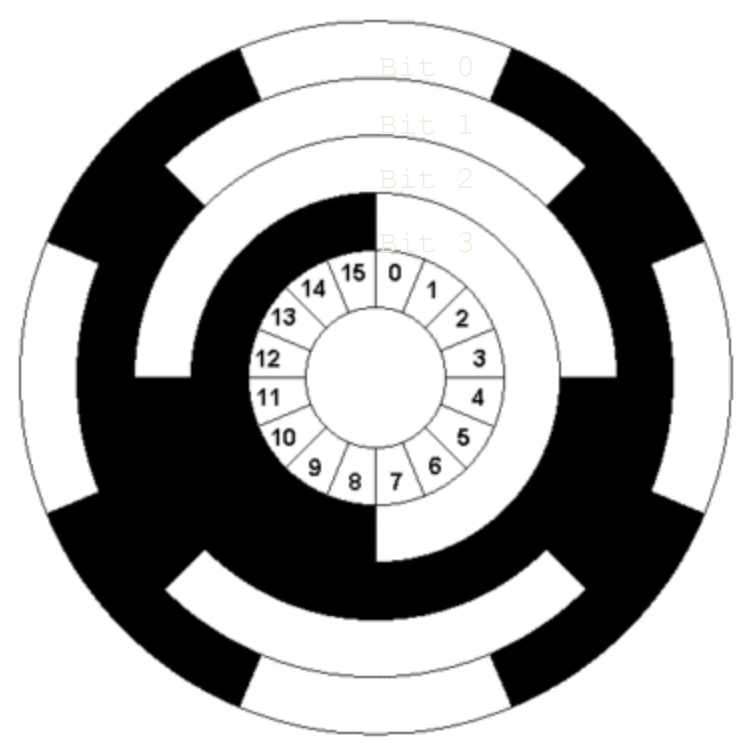
The Gray Code

- Gray-to-Binary Conversion
  - The MSB in the binary code is the same as the corresponding bit in the Gray code.
  - Add each binary code bit generated to the Gray code bit in the next adjacent position. Discard carries.

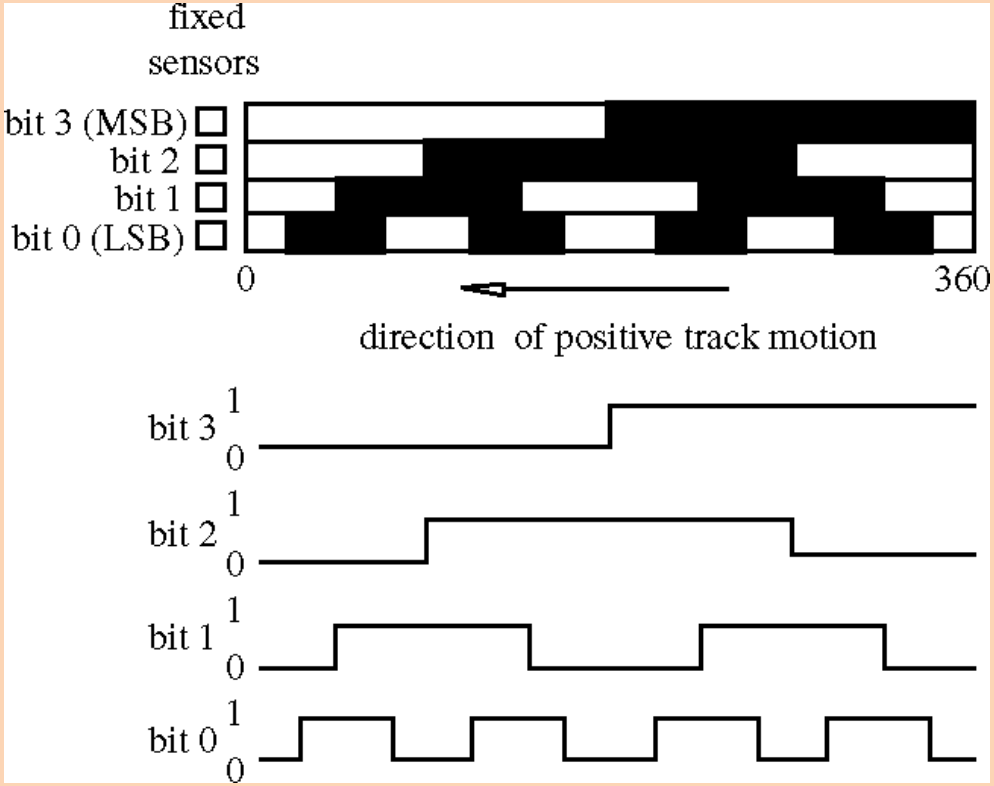
ex: convert the Gray code word `11011 to binary`



The Gray Code - Application



http://www.mipraso.de/enzyklopaedie/g/gray-code-scheibe.gif



http://www.engr.colostate.edu/~dga/mechatronics/figures/9-11.gif

Alphanumeric Codes

- Represent numbers and alphabetic characters.
  - Also represent other characters such as symbols and various instructions necessary for conveying information.
- The ASCII is the most common alphanumeric code.
  - ASCII = American Standard Code for Information Interchange

ASCII

- ASCII has 128 characters and symbols represented by a 7-bit binary code.
  - It can be considered an 8-bit code with the MSB always 0. (00h-7Fh)
    - 00h-1Fh (the first 32) – control characters
    - 20h-7Fh – graphics symbols (can be printed or displayed)

ASCII Table

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

http://ascii-table.com/img/table.gif

Extended ASCII

- There are an additional 128 characters that were adopted by IBM for use in their PCs. It's popular and is used in applications other than PCs → unofficial standard.
  - The extended ASCII characters are represented by an 8-bit code series from 80h-FFh

Extended ASCII Table

	8	9	A	B	C	D	E	F
0	Ç	É	á	▒	└	╨	α	≡
1	ü	æ	í	▓	┴	╤	ß	±
2	é	Æ	ó	▓	┬	╥	Γ	≥
3	â	ô	ú	│	├	╙	π	≤
4	ä	ö	ñ	┤	─	╘	Σ	⌠
5	à	ò	Ñ	╡	┼	╒	σ	⌡
6	å	û	ª	╢	╞	╓	µ	÷
7	ç	ù	º	╖	╟	╫	τ	≈
8	ê	ÿ	¿	╕	╚	╪	Φ	°
9	ë	Ö	⌐	╣	╔	┘	Θ	·
A	è	Ü	¬	║	╩	┌	Ω	·
B	ï	¢	½	╗	╦	▉	δ	√
C	î	£	¼	╝	╠	▄	∞	ⁿ
D	ì	¥	¡	╜	═	▌	φ	²
E	Ä	₧	«	╛	╬	▐	ε	■
F	Å	ƒ	»	┐	╧	▀	∩	

http://ascii-table.com/img/table-pc.gif

Binary Code

- Example:
  - Consider decimal 185 and its corresponding value in BCD and binary:

$$(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (10111001)_2$$

- BCD add

4	0100	4	0100	8	1000
+5	+0101	+8	+1000	+9	+1001
9	1001	12	1100	17	10001
			+0110		+0110
			10010		10111

Binary Code

Example:

– Consider the addition of 184 + 576 = 760 in BCD:

BCD	1	1		
	0001	1000	0100	184
	+0101	0111	0110	+576
Binary sum	0111	10000	1010	
Add 6		0110	0110	
BCD sum	0111	0110	0000	760

0	375
+9	760
0	135

- Decimal Arithmetic: (+375) + (-240) = +135

Hint 6: using 10's of BCD

Binary Codes

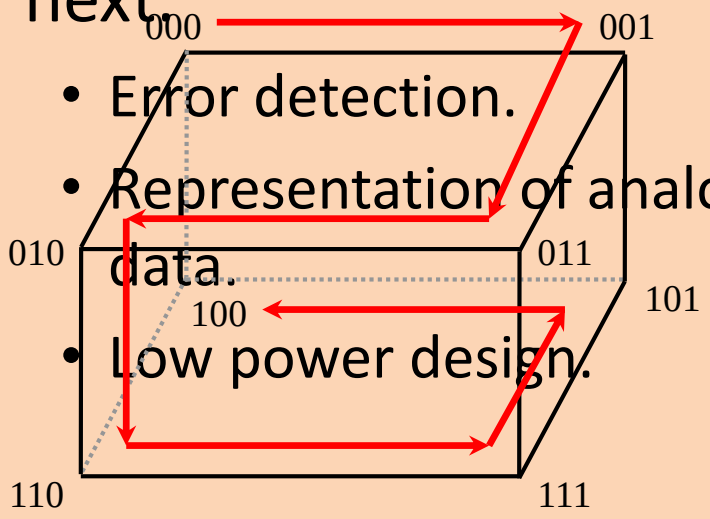
- Other

Table 1.5
Four Different Binary Codes for the Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
	1010	0101	0000	0001
Unused	1011	0110	0001	0010
bit	1100	0111	0010	0011
combi-	1101	1000	1101	1100
nations	1110	1001	1110	1101
	1111	1010	1111	1110

Binary Codes)

- Gray Code
  - The advantage is that only bit in the code group changes in going from one number to the next.
    - Error detection.
    - Representation of analog data.
    - Low power design.



1-1 and onto!!

Table 1.6
Gray Code

Gray Code	Decimal Equivalent
0000	0
0001	1
0011	2
0010	3
0110	4
0111	5
0101	6
0100	7
1100	8
1101	9
1111	10
1110	11
1010	12
1011	13
1001	14
1000	15

Binary Codes

- American Standard Code for Information Interchange (ASCII) Character Code

Table 1.7
American Standard Code for Information Interchange (ASCII)

	$b_7b_6b_5$							
$b_4b_3b_2b_1$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	-	o	DEL

Binary Codes

- ASCII Character Code

Control characters

NUL	Null	DLE	Data-link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End-of-transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

ASCII Character Codes

- American Standard Code for Information Interchange (Refer to Table 1.7)
- A popular code used to represent information sent as character-based data.
- It uses 7-bits to represent:
  - 94 Graphic printing characters.
  - 34 Non-printing characters.
- Some non-printing characters are used for text format (e.g. BS = Backspace, CR = carriage return).
- Other non-printing characters are used for record marking and flow control (e.g. STX and ETX start and end text areas).

Binary Codes

- Error-Detecting Code
  - To detect errors in data communication and processing, an eighth bit is sometimes added to the ASCII character to indicate its parity.
  - A parity bit is an extra bit included with a message to make the total number of 1's either even or

	With even parity	With odd parity
ASCII A = 1000001	01000001	11000001
ASCII T = 1010100	11010100	01010100

-
  - Consider the following two characters and their even and odd parity:

Binary Codes

- Error-Detecting Code
  - Redundancy (e.g. extra information), in the form of extra bits, can be incorporated into binary code words to detect and correct errors.
  - A simple form of redundancy is parity, an extra bit appended onto the code word to make the number of 1's odd or even. Parity can detect all single-bit errors and some multiple-bit errors.
  - A code word has even parity if the number of 1's in the code word is even.
  - A code word has odd parity if the number of 1's in the code word is odd.
  - Example:

Message A: 10001001**1** (even parity)

Message B: 10001001**0** (odd parity)

Binary Logic

- Definition of Binary Logic
  - Binary logic consists of binary variables and a set of logical operations.
  - The variables are designated by letters of the alphabet, such as *A*, *B*, *C*, *x*, *y*, *z*, etc, with each variable having two and only two distinct possible values: 1 and 0
  -

1. AND: This operation is represented by a dot or by the absence of an operator. For example, $x \cdot y = z$ or $xy = z$ is read "x AND y is equal to z," The logical operation AND is interpreted to mean that $z = 1$ if only $x = 1$ and $y = 1$; otherwise $z = 0$. (Remember that x, y, and z are binary variables and can be equal either to 1 or 0, and nothing else.)
2. OR: This operation is represented by a plus sign. For example, $x + y = z$ is read "x OR y is equal to z," meaning that $z = 1$ if $x = 1$ or $y = 1$ or if both $x = 1$ and $y = 1$. If both $x = 0$ and $y = 0$, then $z = 0$.
3. NOT: This operation is represented by a prime (sometimes by an overbar). For example, x' = z (or $\bar{x} = z$) is read "not x is equal to z," meaning that z is what z is not. In other words, if $x = 1$, then $z = 0$, but if $x = 0$, then $z = 1$, The NOT operation is also referred to as the complement operation, since it changes a 1 to 0 and a 0 to 1.

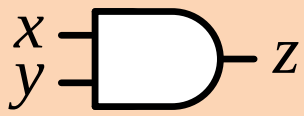
Binary Logic

- Truth Tables, Boolean Expressions, and Logic Gates

AND

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

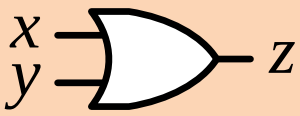
z = x • y = x y



OR

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

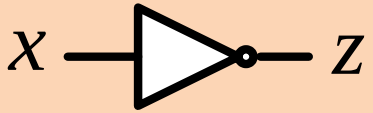
z = x + y



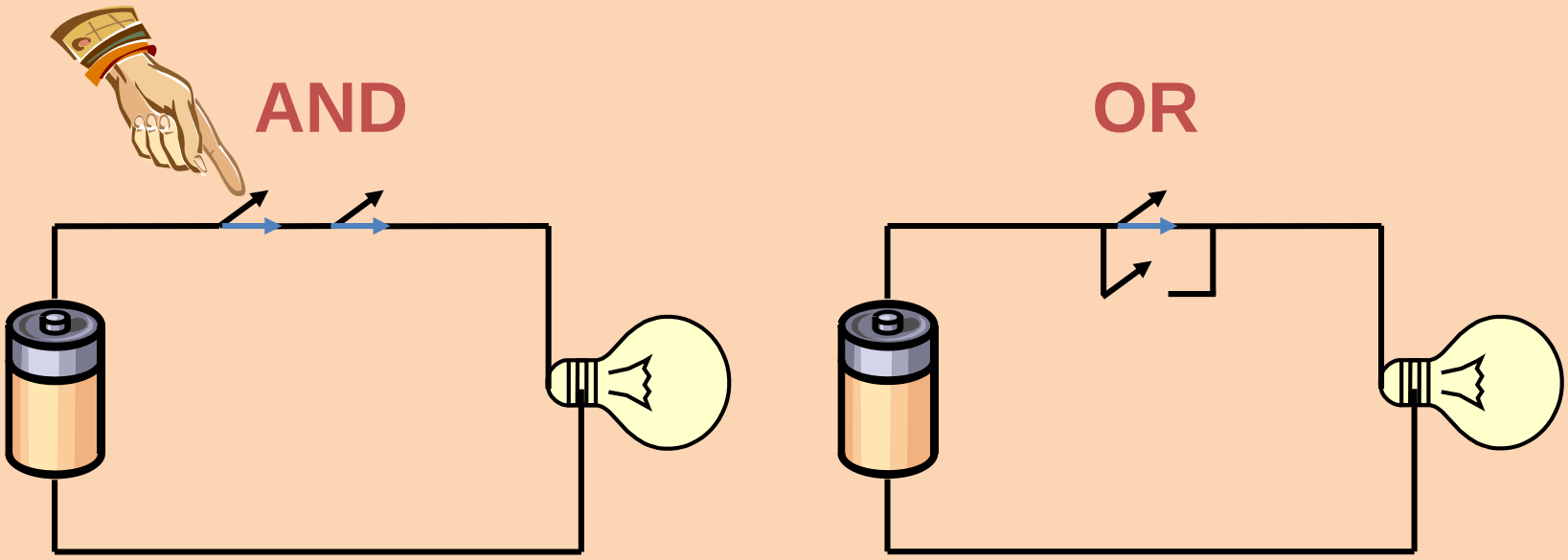
NOT

x	z
0	1
1	0

z = $\overline{x}$ = x'

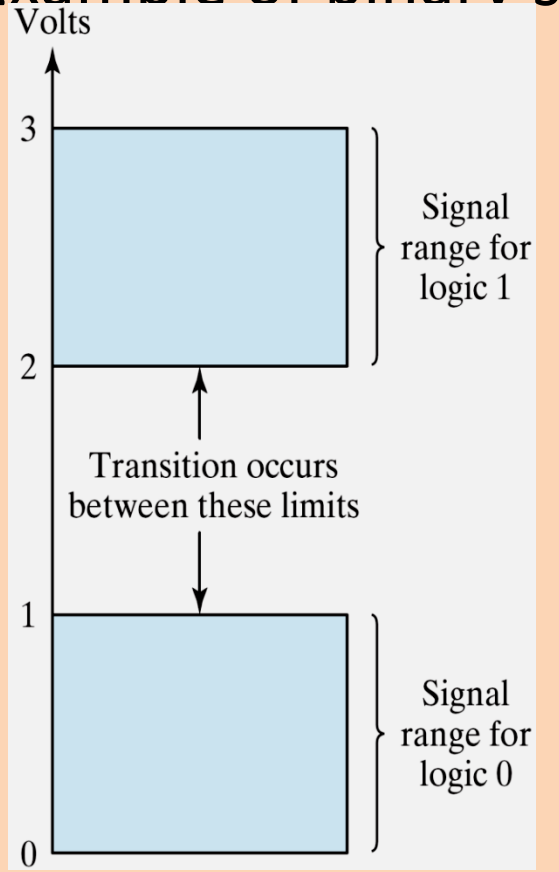


Switching Circuits



Binary Logic

- Logic gates
  - Example of binary signals



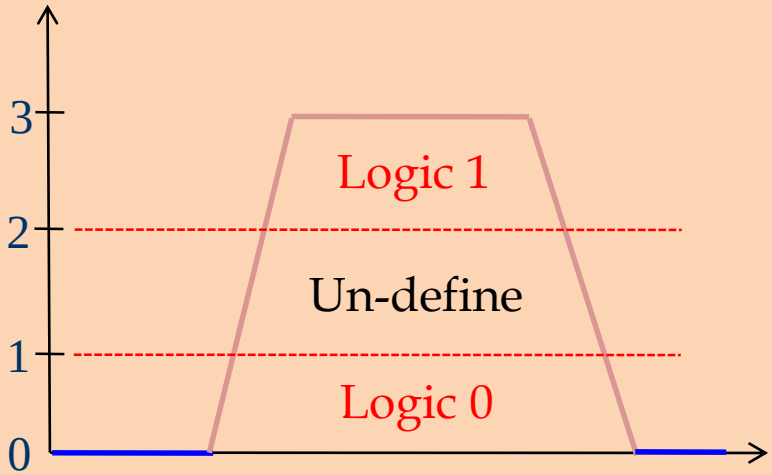
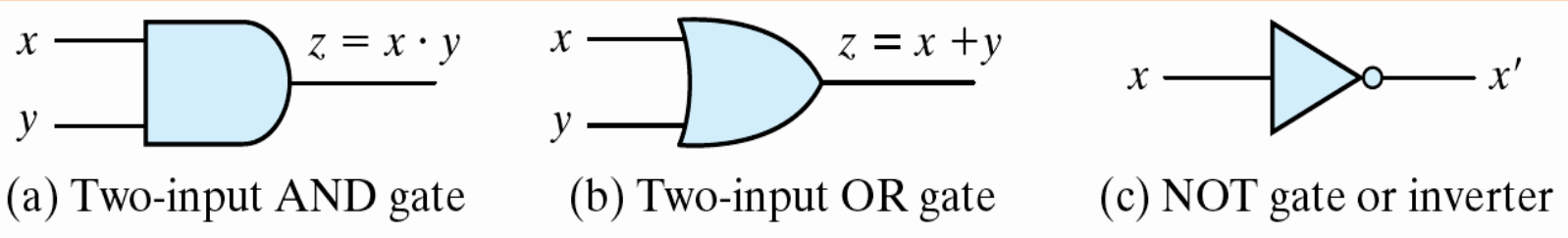


Figure 1.3 Example of binary signals

Binary Logic

- Logic gates



(a) Two-input AND gate (b) Two-input OR gate (c) NOT gate or inverter

Fig. 1.4 Symbols for digital logic circuits

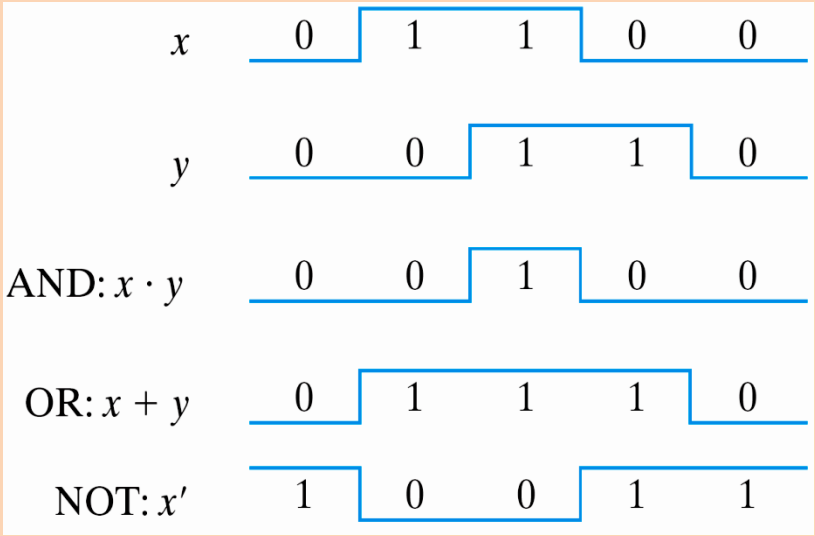


Fig. 1.5 Input-Output signals for gates

Binary Logic

- Logic gates
  - Graphic Symbols and Input-Output Signals for

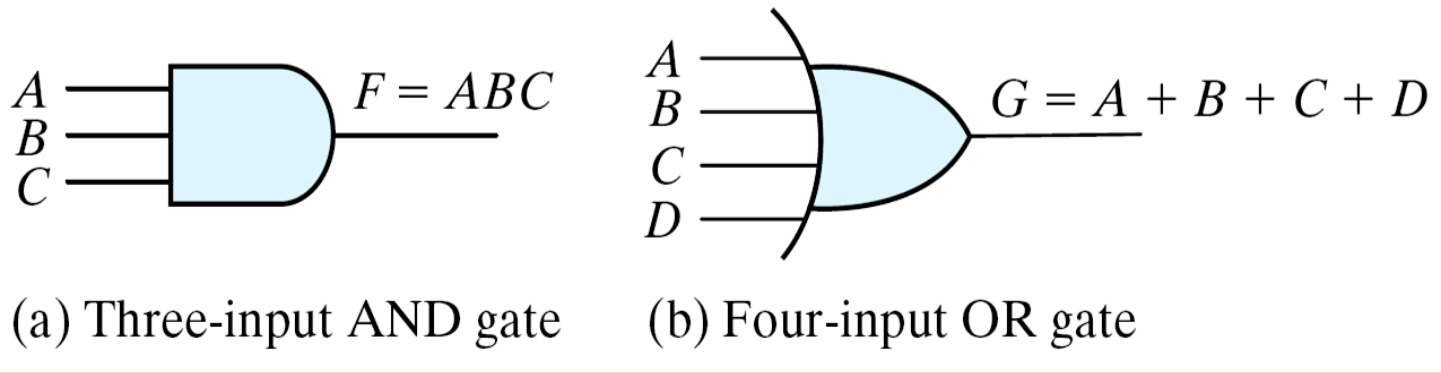


Fig. 1.6 Gates with multiple inputs

Boolean algebra Algebra

- Boolean algebra is a mathematical system for the manipulation of variables that can have one of two values.
  - In formal logic, these values are “true” and “false.”
  - In digital systems, these values are “on” and “off,” 1 and 0, or “high” and “low.”
- Boolean expressions are created by performing operations on Boolean variables.
  - Common Boolean operators include AND, OR, and NOT.

Boolean Algebra

- A Boolean operator can be completely described using a truth table.
- The truth table for the Boolean operators AND and OR are shown at the right.
- The AND operator is also known as a Boolean product. The OR operator is the Boolean sum.

X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

Boolean Algebra

- The truth table for the Boolean NOT operator is shown at the right.
- The NOT operation is most often designated by an overbar. It is sometimes indicated by a prime mark (‘) or an “elbow” (¬).

NOT x

x	$\overline{x}$
0	1
1	0

Boolean Algebra

- A Boolean function has:
  - At least one Boolean variable,
  - At least one Boolean operator, and
  - At least one input from the set {0,1}.
- It produces an output that is also a member of the set {0,1}.

Now you know why the binary numbering system is so handy in digital systems.

Boolean Algebra

- The truth table for the Boolean function:

$$F(x,y,z) = x\bar{z}+y$$

  is shown at the right.
- To make evaluation of the Boolean function easier, the truth table contains extra (shaded) columns to hold evaluations of subparts of the function.

$F(x,y,z) = x\bar{z}+y$

x	y	z	$\bar{z}$	$x\bar{z}$	$x\bar{z}+y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

Boolean Algebra

- As with common arithmetic, Boolean operations have rules of precedence.
- The NOT operator has highest priority, followed by AND and then OR.
- This is how we chose the (shaded) function subparts in our table.

$F(x,y,z) = x\bar{z}+y$

x	y	z	$\bar{z}$	$x\bar{z}$	$x\bar{z}+y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

Boolean Algebra

- Digital computers contain circuits that implement Boolean functions.
- The simpler that we can make a Boolean function, the smaller the circuit that will result.
  - Simpler circuits are cheaper to build, consume less power, and run faster than complex circuits.
- With this in mind, we always want to reduce our Boolean functions to their simplest form.
- There are a number of Boolean identities that help us to do this.

Boolean Algebra

- **Boolean Constants**
  - these are ‘0’ (false) and ‘1’ (true)
- **Boolean Variables**
  - variables that can only take the vales ‘0’ or ‘1’
- **Boolean Functions**
  - each of the logic functions (such as AND, OR and NOT) are represented by symbols as described above
- **Boolean Theorems**
  - a set of **identities** and **laws** – see text for details

- **Boolean identities**

AND Function	OR Function	NOT function
$0 \bullet 0 = 0$	$0 + 0 = 0$	$\overline{0} = 1$
$0 \bullet 1 = 0$	$0 + 1 = 1$	$\overline{1} = 0$
$1 \bullet 0 = 0$	$1 + 0 = 1$	$\overline{\overline{A}} = A$
$1 \bullet 1 = 1$	$1 + 1 = 1$	
$A \bullet 0 = 0$	$A + 0 = A$	
$0 \bullet A = 0$	$0 + A = A$	
$A \bullet 1 = A$	$A + 1 = 1$	
$1 \bullet A = A$	$1 + A = 1$	
$A \bullet A = A$	$A + A = A$	
$A \bullet \overline{A} = 0$	$A + \overline{A} = 1$	

- **Boolean laws**

Commutative law	**Absorption law**
$AB = BA$ $A + B = B + A$	$A + AB = A$ $A(A + B) = A$
Distributive law	**De Morgan's law**
$A(B + C) = AB + BC$ $A + BC = (A + B)(A + C)$	$\overline{A + B} = \overline{A} \bullet \overline{B}$ $\overline{A \bullet B} = \overline{A} + \overline{B}$
Associative law	**Note also**
$A(BC) = (AB)C$ $A + (B + C) = (A + B) + C$	$A + \overline{A}B = A + B$ $A(\overline{A} + B) = AB$

Boolean Algebra

- Most Boolean identities have an AND (product) form as well as an OR (sum) form. We give our identities using both forms. Our first group is rather intuitive:

Identity Name	AND Form	OR Form
Identity Law	$1x = x$	$0 + x = x$
Null Law	$0x = 0$	$1 + x = 1$
Idempotent Law	$xx = x$	$x + x = x$
Inverse Law	$x\bar{x} = 0$	$x + \bar{x} = 1$

Boolean Algebra

- Our second group of Boolean identities should be familiar to you from your study of algebra:

Identity Name	AND Form	OR Form
Commutative Law	$xy = yx$	$x+y = y+x$
Associative Law	$(xy)z = x(yz)$	$(x+y)+z = x + (y+z)$
Distributive Law	$x+yz = (x+y)(x+z)$	$x(y+z) = xy+xz$

Boolean Algebra

- Our last group of Boolean identities are perhaps the most useful.
- If you have studied set theory or formal logic, these laws are also familiar to you.

Identity Name	AND Form	OR Form
Absorption Law	$x(x+y) = x$	$x + xy = x$
DeMorgan's Law	$\overline{(xy)} = \bar{x} + \bar{y}$	$\overline{(x+y)} = \bar{x}\bar{y}$
Double Complement Law	$\overline{(\bar{x})} = x$	

Boolean Algebra

- We can use Boolean identities to simplify the function: $F(X,Y,Z) = (X + Y)(X + \overline{Y})\,\overline{(X\overline{Z})}$ as follows:

$(X + Y)(X + \overline{Y})\,\overline{(X\overline{Z})}$	Idempotent Law (Rewriting)
$(X + Y)(X + \overline{Y})(\overline{X} + Z)$	DeMorgan's Law
$(XX + X\overline{Y} + XY + Y\overline{Y})(\overline{X} + Z)$	Distributive Law
$((X + Y\overline{Y}) + X(Y + \overline{Y}))(\overline{X} + Z)$	Commutative & Distributive Laws
$((X + 0) + X(1))(\overline{X} + Z)$	Inverse Law
$X(\overline{X} + Z)$	Idempotent Law
$X\overline{X} + XZ$	Distributive Law
$0 + XZ$	Inverse Law
XZ	Idempotent Law

Boolean Algebra

- Sometimes it is more economical to build a circuit using the complement of a function (and complementing its result) than it is to implement the function directly.
- DeMorgan's law provides an easy way of finding the complement of a Boolean function.
- Recall DeMorgan's law states:

$$\overline{(xy)} = \bar{x} + \bar{y} \quad \text{and} \quad \overline{(x+y)} = \bar{x}\bar{y}$$

Boolean Algebra

- DeMorgan's law can be extended to any number of variables.
- Replace each variable by its complement and change all ANDs to ORs and all ORs to ANDs.
- Thus, we find the the complement of:

$$F(X,Y,Z) = (XY) + (\overline{X}Z) + (Y\overline{Z})$$

is:

$$\overline{F}(X,Y,Z) = \overline{(XY) + (\overline{X}Z) + (Y\overline{Z})}$$

$$= \overline{(XY)}\,\overline{(\overline{X}Z)}\,\overline{(Y\overline{Z})}$$

$$= (\overline{X}+\overline{Y})(X+\overline{Z})(\overline{Y}+Z)$$

Boolean Algebra

- Through our exercises in simplifying Boolean expressions, we see that there are numerous ways of stating the same Boolean expression.
  - These “synonymous” forms are *logically equivalent.*
  - Logically equivalent expressions have identical truth tables.
- In order to eliminate as much confusion as possible, designers express Boolean functions in *standardized* or *canonical* form.

Boolean Algebra

- There are two canonical forms for Boolean expressions: sum-of-products and product-of-sums.
  - Recall the Boolean product is the AND operation and the Boolean sum is the OR operation.
- In the sum-of-products form, ANDed variables are ORed together.
  - For example: $F(x,y,z) = xy + xz + yz$
- In the product-of-sums form, ORed variables are ANDed together:
  - For example: $F(x,y,z) = (x+y)(x+z)(y+z)$

Boolean Algebra

- It is easy to convert a function to sum-of-products form using its truth table.
- We are interested in the values of the variables that make the function true (=1).
- Using the truth table, we list the values of the variables that result in a true function value.
- Each group of variables is then ORed together.

$F(x,y,z) = x\bar{z}+y$

x	y	z	$x\bar{z}+y$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Boolean Algebra

- The sum-of-products form for our function is:

$$F(x,y,z) = \bar{x}y\bar{z} + \bar{x}yz + x\bar{y}\bar{z} + xy\bar{z} + xyz$$

We note that this function is not in simplest terms. Our aim is only to rewrite our function in canonical sum-of-products form.

$$F(x,y,z) = x\bar{z} + y$$

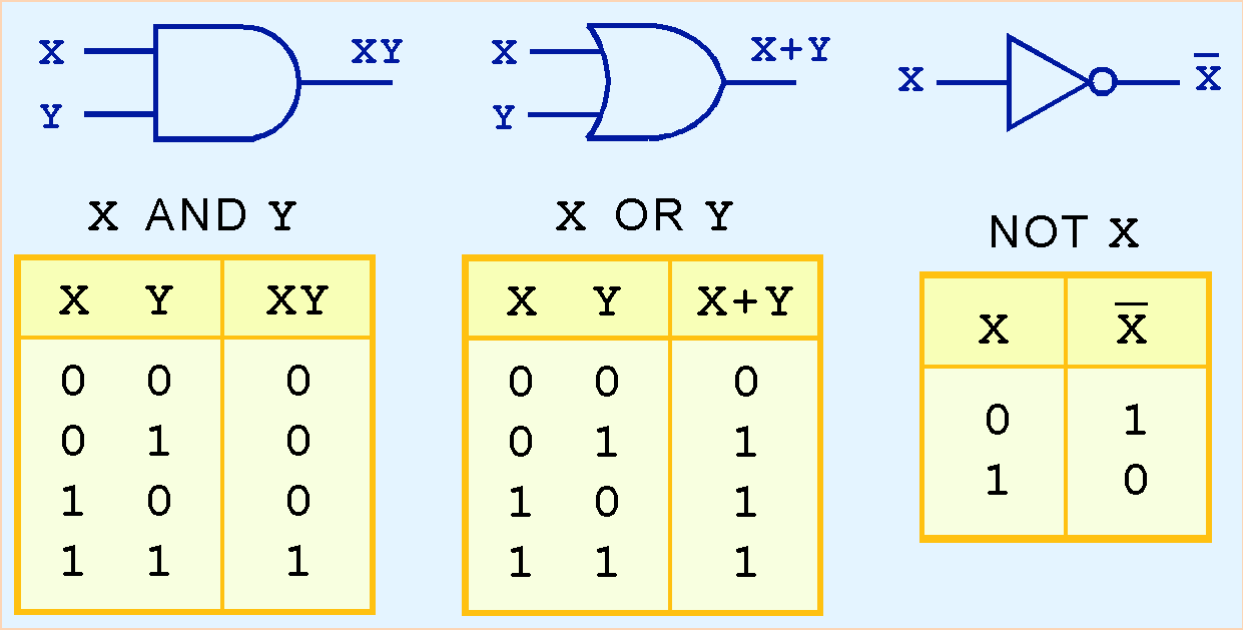
x	y	z	$x\bar{z}+y$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Logic Gates

- We have looked at Boolean functions in abstract terms.
- In this section, we see that Boolean functions are implemented in digital computer circuits called gates.
- A gate is an electronic device that produces a result based on two or more input values.
  - In reality, gates consist of one to six transistors, but digital designers think of them as a single unit.
  - Integrated circuits contain collections of gates suited to a particular purpose.

Logic Gates

- The three simplest gates are the AND, OR, and NOT gates.



X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT X

X	$\overline{X}$
0	1
1	0

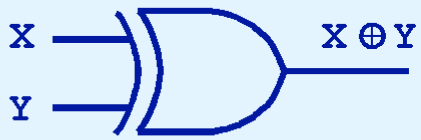
- They correspond directly to their respective Boolean operations, as you can see by their truth tables.

Logic Gates

- Another very useful gate is the exclusive OR (XOR) gate.
- The output of the XOR operation is true only when the values of the inputs differ.

X XOR Y

X	Y	X⊕Y
0	0	0
0	1	1
1	0	1
1	1	0



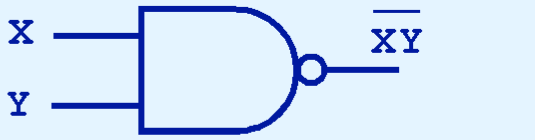
Note the special symbol ⊕ for the XOR operation.

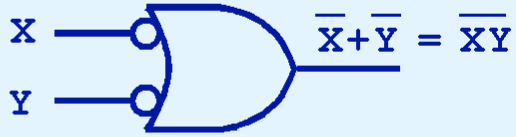
Logic Gates

- NAND and NOR are two very important gates. Their symbols and truth tables are shown at the right.

X NAND Y

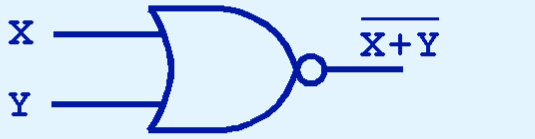
X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0

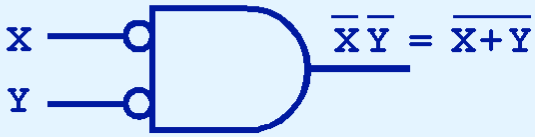




X NOR Y

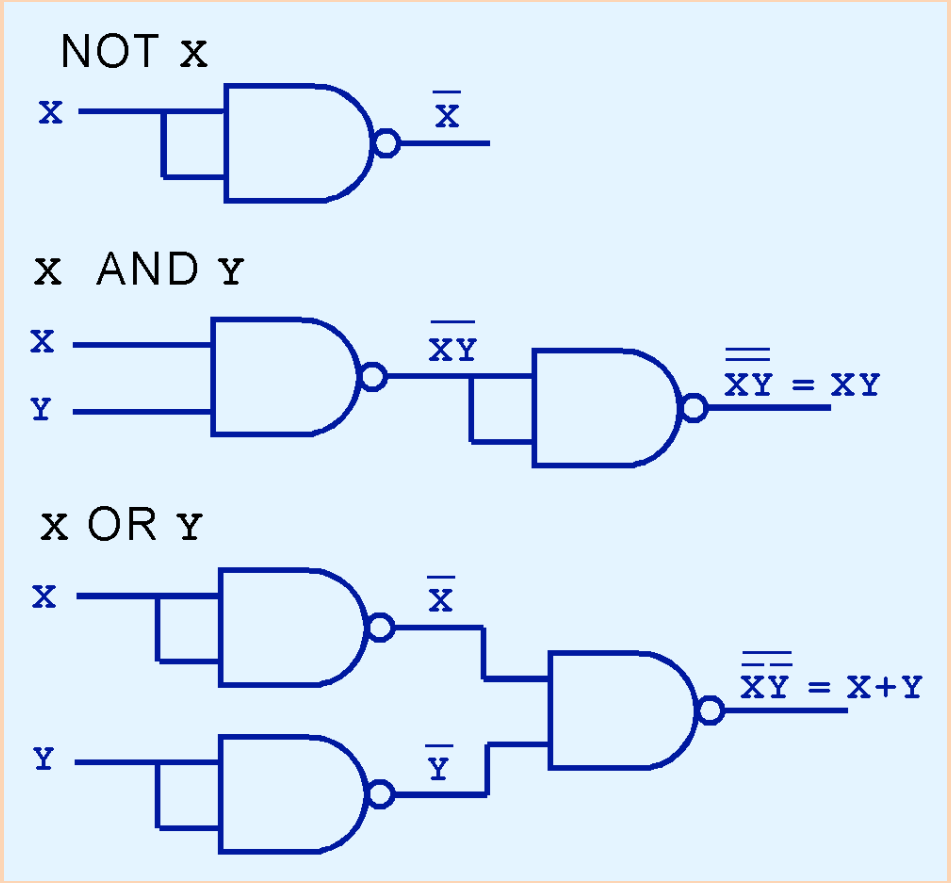
X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0





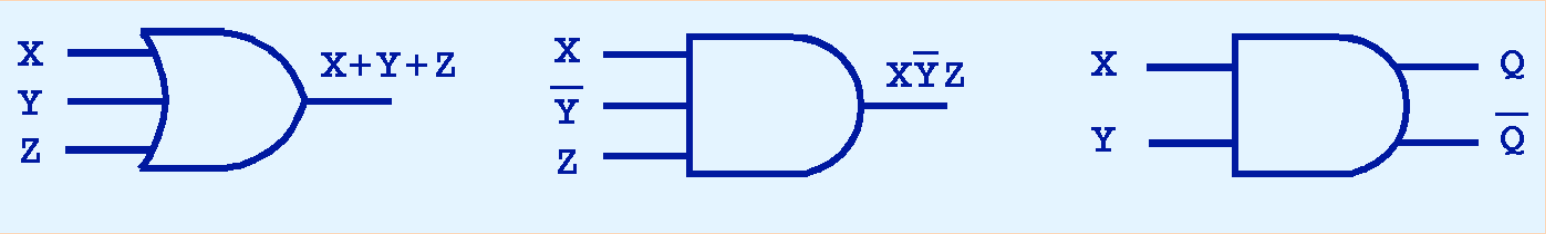
Logic Gates

- NAND and NOR are known as *universal gates* because they are inexpensive to manufacture and any Boolean function can be constructed using only NAND or only NOR gates.



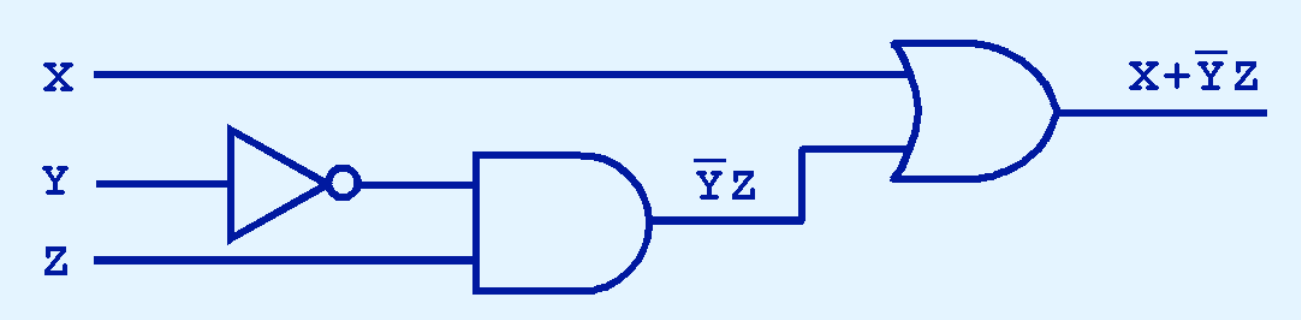
Logic Gates

- Gates can have multiple inputs and more than one output.
  - A second output can be provided for the complement of the operation.
  - We'll see more of this later.



Digital Components

- The main thing to remember is that combinations of gates implement Boolean functions.
- The circuit below implements the Boolean function: $F(X,Y,Z) = X+\overline{Y}Z$



We simplify our Boolean expressions so that we can create simpler circuits.