Digital Circuits and Logic Design (BCS-11)

By

Manish Kumar Srivastava

Assistant Professor

Department of Computer Science & Engineering

Madan Mohan Malaviya University of Technology Gorakhpur

(UP State Govt. University)

Email: mkscs@mmmut.ac.in

Syllabus

UNIT-I

Binary Codes - Weighted and Non-Weighted - Binary Arithmetic Conversion Algorithms - Error Detecting and Error Correcting Codes - Canonical and Standard Boolean Expressions - Truth Tables.

A **digital circuit** is a **circuit** where the signal must be one of two discrete levels. Each level is interpreted as one of two different states (for example, on/off, 0/1, true/false). **Digital circuits** use transistors to create logic gates in order to perform Boolean logic.

Logic Design

All digital computers are based on a two-valued logic system—1/0, on/off, yes/no . Computers perform calculations using components called logic gates, which are made up of integrated circuits that receive an input signal, process it, and change it into an output signal. There are three basic kinds of logic gates, called “and,” “or,” and “not.” By connecting logic gates together, a device can be constructed that can perform basic arithmetic functions.

- **Digital Signal** : Decimal values are difficult to represent in electrical systems. It is easier to use two voltage values than ten.

- Digital Signals have two basic states:

  1 (logic "high", or H, or "on")

  0 (logic "low", or L, or "off")

- Digital values are in a *binary* format. Binary means 2 states.

- A good example of binary is a light (only on or off)

Power switches have labels "1" for on and "0" for off.

- Bits and Pieces of DLD History

George Boole

- Mathematical Analysis of Logic (1847)
- An Investigation of Laws of Thoughts; Mathematical Theories of Logic and Probabilities (1854)

Claude Shannon

- Rediscovered the Boole
- “ A Symbolic Analysis of Relay and Switching Circuits “
- Boolean Logic and Boolean Algebra were Applied to Digital Circuitry

---------- Beginning of the Digital Age and/or Computer Age

World War II

Computers as Calculating Machines

Arlington (State Machines) “ Control “

Motivation

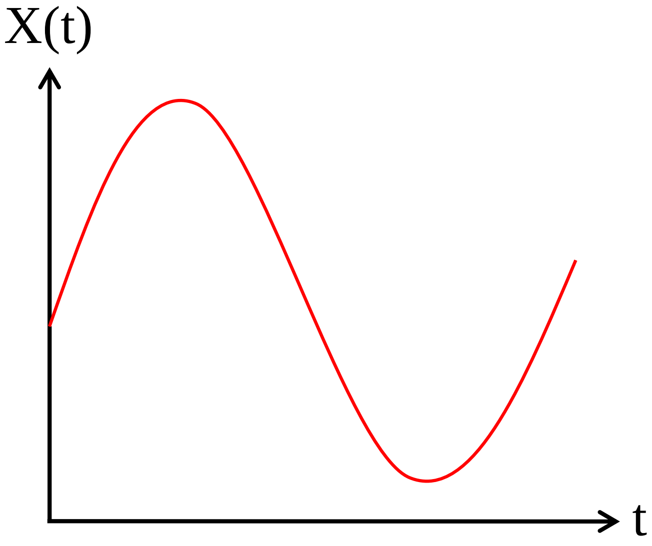
- Microprocessors/Microelectronics have revolutionized our world
  - Cell phones, internet, rapid advances in medicine, etc.
- The semiconductor industry has grown tremendously

Digital Systems and Binary Numbers

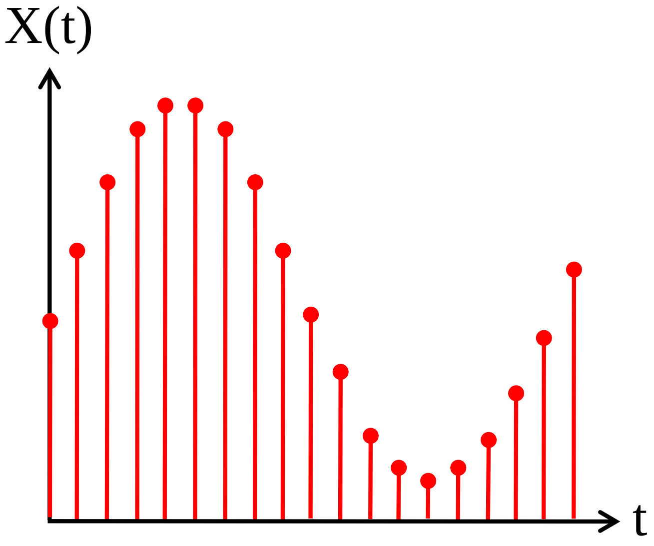
- Digital age and information age
- Digital computers
  - General purposes
  - Many scientific, industrial and commercial applications
- Digital systems
  - Telephone switching exchanges
  - Digital camera
  - Electronic calculators, PDA's
  - Digital TV
- Discrete information-processing systems
  - 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
  - The physical quantities or signals can assume only discrete values.
  - Greater accuracy



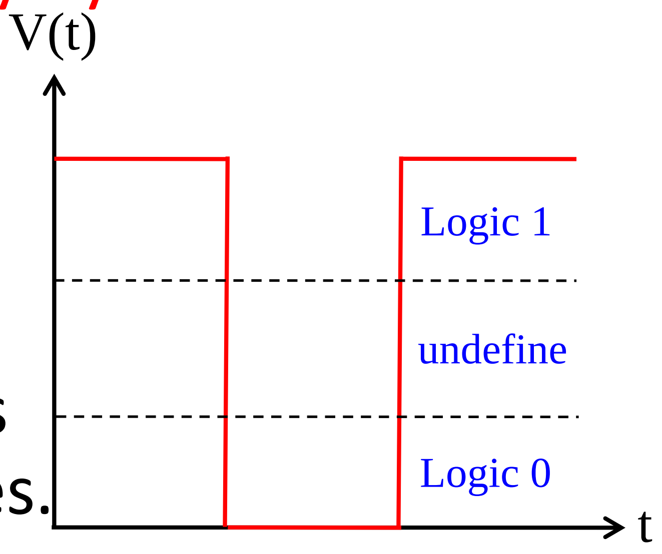
Analog signal



Digital signal

Binary Digital Signal

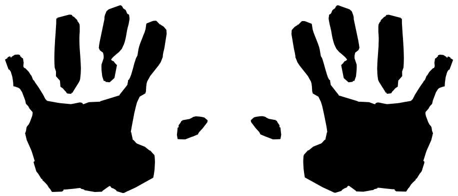
- An information variable represented by physical quantity.
- For digital systems, the variable takes on discrete values.
  - Two level, or binary values are the most prevalent values.
- Binary values are represented abstractly by:
  - Digits 0 and 1
  - Words (symbols) False (F) and True (T)
  - Words (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

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

64	8	1		1/8	1/64
5	1	2		7	4
2	1	0		-1	-2

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

$=(330.9375)_{10}$

$(512.74)_{8}$

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

256	16	1		1/16	1/256
1	E	5	.	7	A
2	1	0		-1	-2

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

$$= (485.4765625)_{10}$$

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

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

	1	1	
		5	5
+		5	5
	1	1	0

← Carry

= Ten ≥ Base

➔ Subtract a Base

Binary Addition

- Column Addition

$$
\begin{array}{cccccccccl}
 & 1 & 1 & 1 & 1 & 1 & 1 & & & \\
 & & 1 & 1 & 1 & 1 & 0 & 1 & & = 61 \\
+ & & & 1 & 0 & 1 & 1 & 1 & & = 23 \\
\hline
 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & & = 84
\end{array}
$$

$$\geq (2)_{10}$$

Binary Subtraction

- Borrow a “Base” when needed

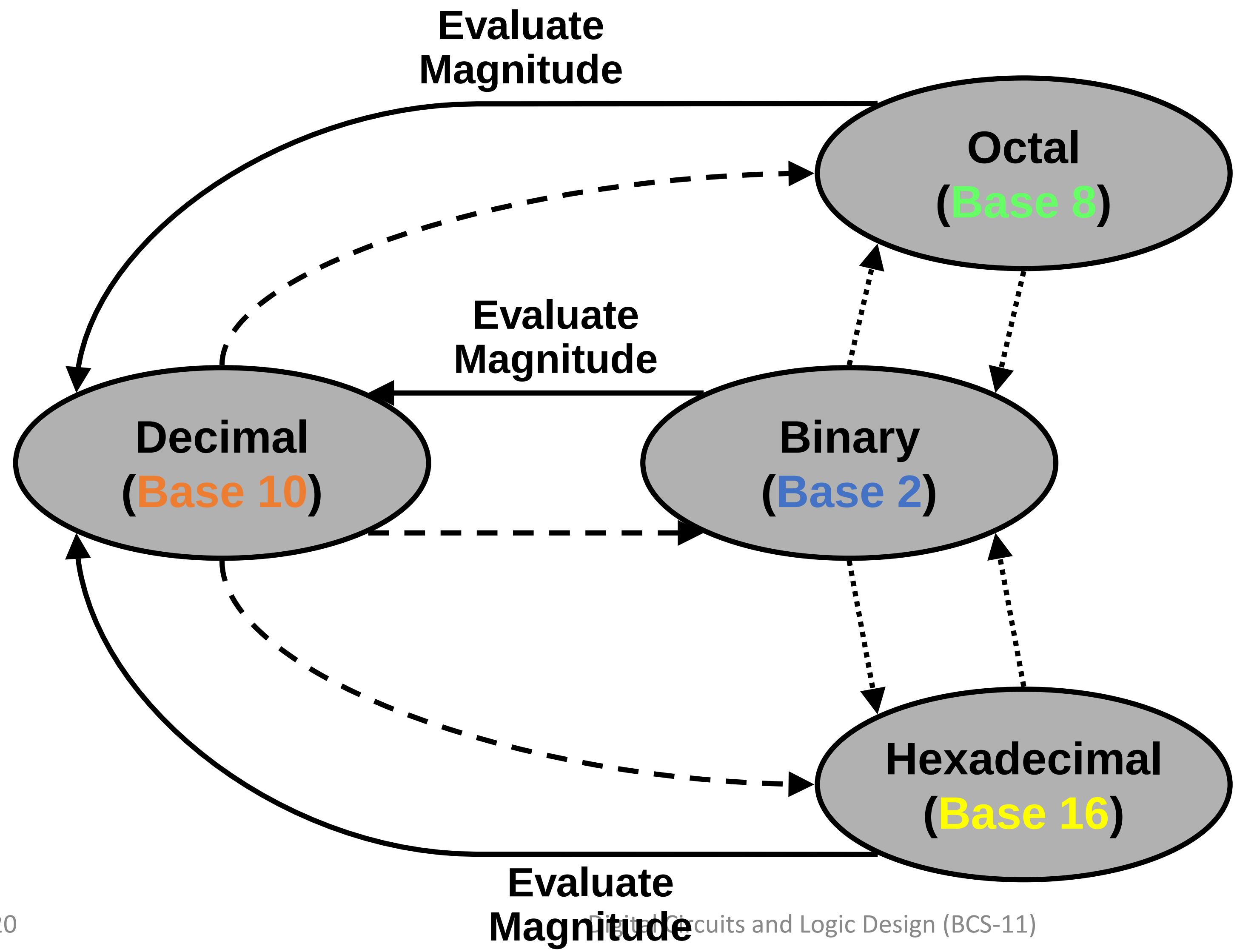
$$
\begin{array}{cccccccccl}
 & & 1 & & & 2 & & & & \\
 & 0 & \cancel{2} & 2 & 0 & 0 & 2 & & & = (10)_2 \\
 & \cancel{1} & 0 & 0 & \cancel{1} & \cancel{1} & 0 & 1 & & = 77 \\
- & & & 1 & 0 & 1 & 1 & 1 & & = 23 \\
\hline
 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & & = 54
\end{array}
$$

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:

Assume Zeros

$(\ 1\,0\,1\,1\,0\,.\,0\,1\ )_2$

$(\ 2\quad 6\ .\ 2\ )_8$

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

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

Assume Zeros

Example: $(10110.01)_2$

$(1\quad 6\ .\ 4)_{16}$

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

Octal – Hexadecimal Conversion

- Convert to Binary as an intermediate step

Example:

$$(2\ \ 6\ .\ 2)_8$$

Assume Zeros

$$(0\,1\,0\,1\,1\,0\,.\,0\,1\,0)_2$$

Assume Zeros

$$(1\ \ 6\ .\ 4)_{16}$$

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's Comp.:

	1 0 1 1 0 0 0 0	1 0 1 1 0 0 0 0
	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 $M - N$ in base *r* can be done as follows:

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 $M \geq N$, 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
  - Using 10's complement, subtract 72532 – 3250.

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

- Example 1.6
  - Using 10's complement, subtract 3250 – 72532.

	M =	03250
10's complement of	N =	$+\underline{27468}$
	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 (b) *Y* – *X*, by using 2's complement.

(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) $X - Y = 1010100 - 1000011$

$$
\begin{aligned}
X &= \quad 1010100\\
\text{1's complement of } Y &= \pm\, 0111100\\
\text{Sum} &= \; 10010000\\
\text{End-around carry} &= \underline{+ \qquad\quad 1}\\
\text{Answer. } X - Y &= \quad 0010001
\end{aligned}
$$

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

$$
\begin{aligned}
Y &= \quad 1000011\\
\text{1's complement of } X &= \underline{+\, 0101011}\\
\text{Sum} &= \quad 1101110
\end{aligned}
$$

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.
- Example:

+ 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

In 2's-complement form:

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)

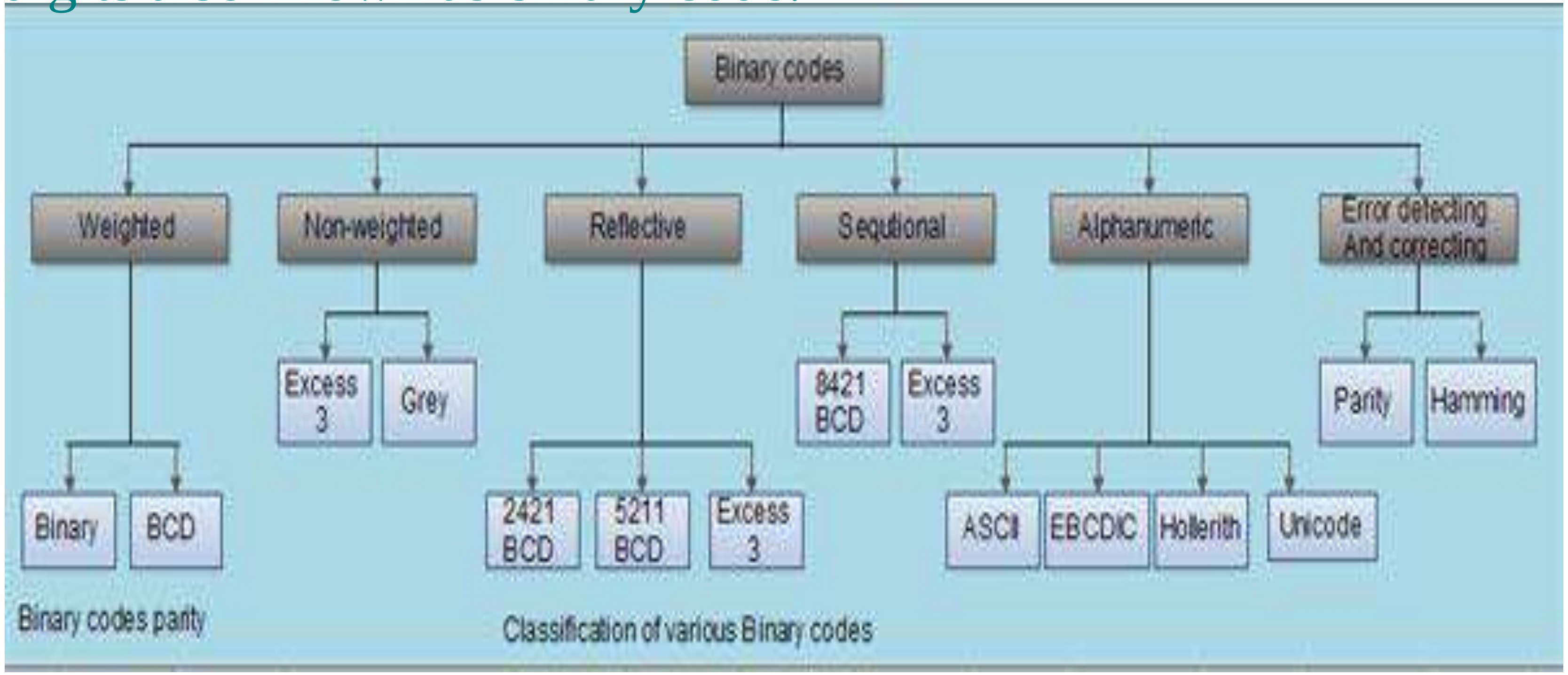
(11111010 – 11110011)

(11111010 + 00001101)

00000111 (+ 7)

Binary Codes

Digital data is represented, stored and transmitted as groups of binary digits also known as binary code.



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

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

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

BCD addition

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

Binary Codes

- Other Decimal Codes

Table 1.5
Four Different Binary Codes for the Decimal Digits

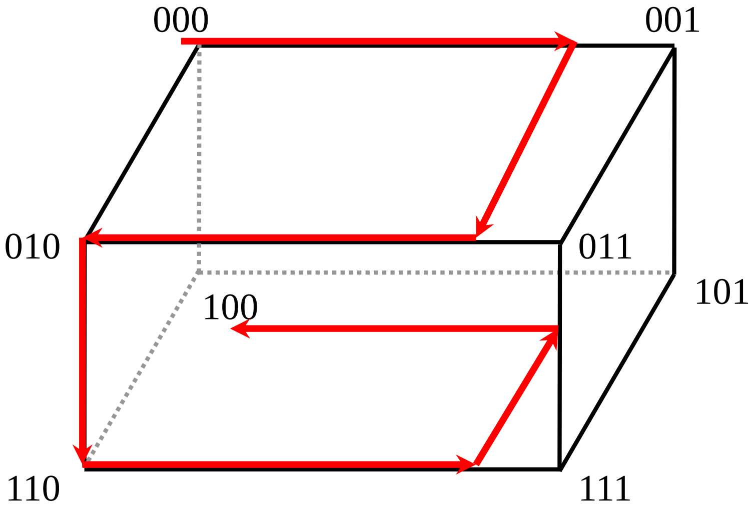
Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
	1010	0101	0000	0001
Unused	1011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

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



1-1 and onto!!

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

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 and Properties

- 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).
- ASCII has some interesting properties:
  - Digits 0 to 9 span Hexadecimal values 3016 to 3916
  - Upper case A-Z span 4116 to 5A16
  - Lower case a-z span 6116 to 7A16
    - Lower to upper case translation (and vice versa) occurs by flipping bit 6.

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 odd.

Example:

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

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

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)

Hamming Codes

- Invented W.B Hamming and Simple 1parity bit can tell us an error occurred
- Multiple parity bits can also tell us where it occurred
- O(lg(n)) bits needed to detect and correct one bit errors.
- In generally we use 7 bits hamming code
  - 4 data bits/message bit (m) and 3 parity bits ($2^{P}>= P+m+1$)

Example: Byte **1011 0001**

Two data blocks, **1011** and **0001**.

Expand the first block to 7 bits: **_ _ 1 _ 0 1 1**

Bit 1 is 0, because b3+b5+b7 is even.

Bit 2 is 1, b3+b6+b7 is odd.

bit 4 is 0, because b5+b6+b7 is even.

Our 7 bit block is: **0 1 1 0 0 1 1**

Bit position		1	2	3	4	5	6	7	
Encoded data bits		p1	p2	d1	p3	d2	d3	d4	
Parity bit coverage	p1	X		X		X		X	...
	p2		X	X			X	X	
	p3				X	X	X	X	

Repeat for right block giving **1 1 0 1 0 0 1**

Error detectings: 0 1 1 0 1 1 1

Re-Check each parity bit

Bits 1 and 4 are incorrect

1 + 4 = 5, so the error occurred in bit 5

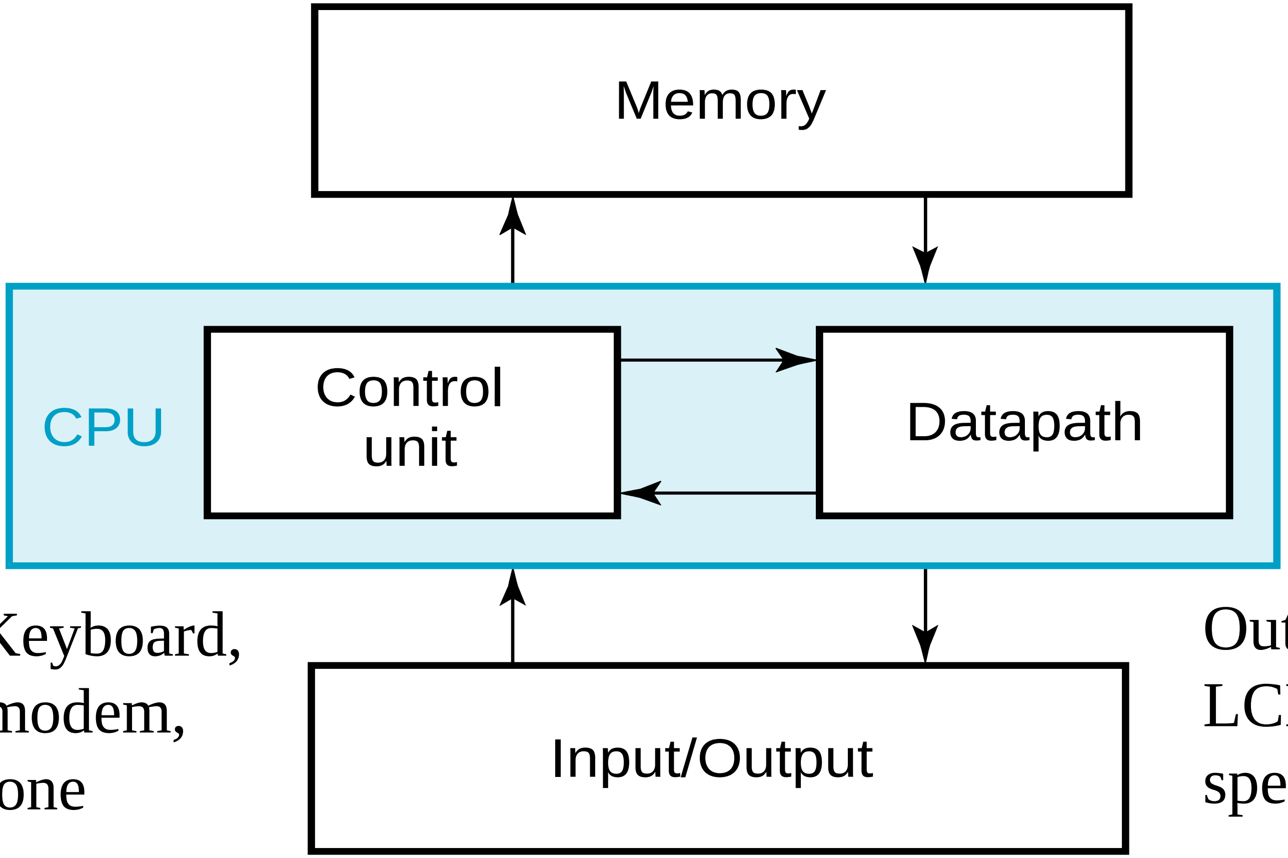
Binary Storage and Registers

- Registers
  - A binary cell is a device that possesses two stable states and is capable of storing one of the two states.
  - A register is a group of binary cells. A register with *n* cells can store any discrete quantity of information that contains *n* bits.

n cells → 2^n possible states

- A binary cell
  - Two stable state
  - Store one bit of information
  - Examples: flip-flop circuits, ferrite cores, capacitor
- A register
  - A group of binary cells
  - AX in x86 CPU
- Register Transfer
  - A transfer of the information stored in one register to another.
  - One of the major operations in digital system.
  - An example in next slides.

A Digital Computer Example

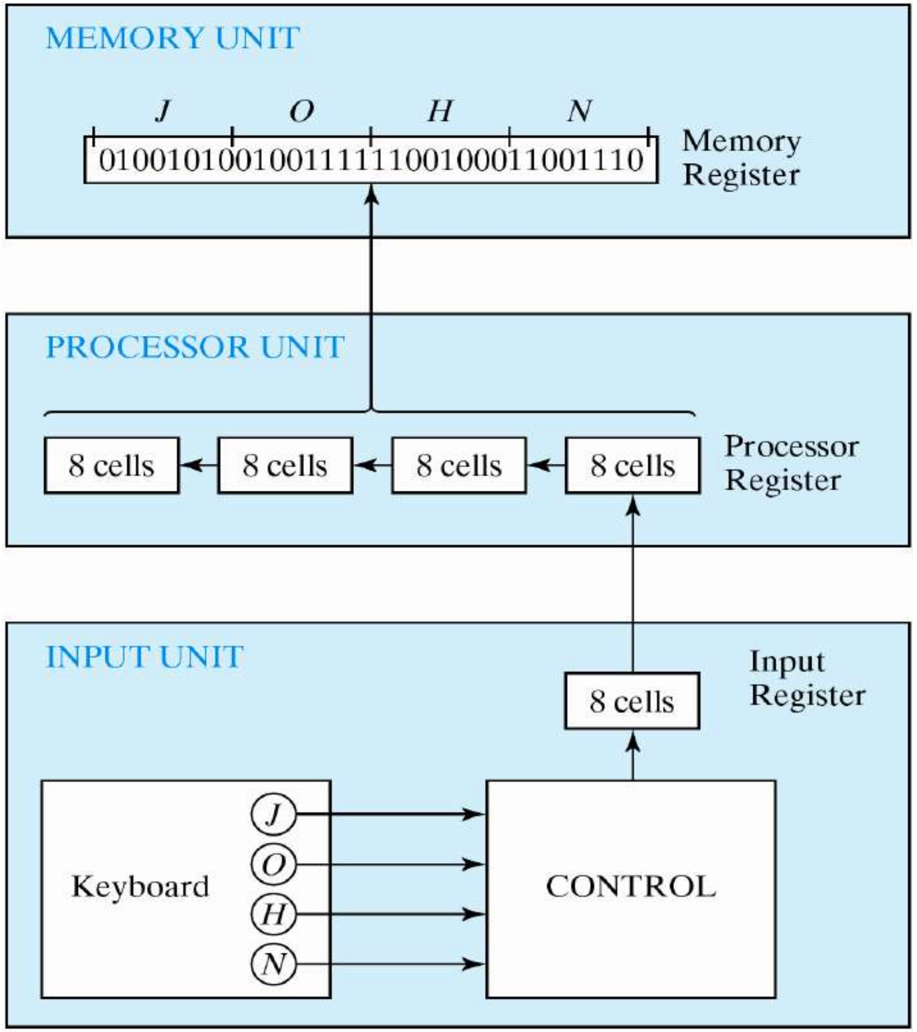


Inputs: Keyboard, mouse, modem, microphone

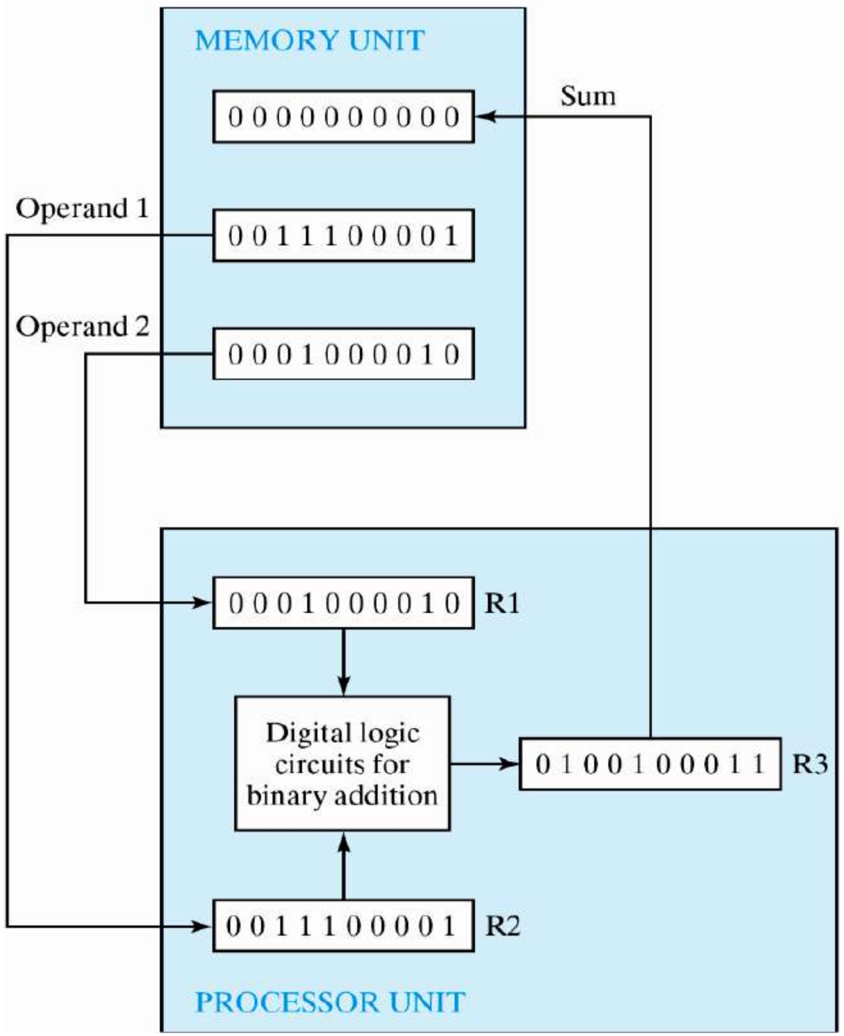
Outputs: CRT, LCD, modem, speakers

Synchronous or Asynchronous?

Transfer of information



Transfer of information



- The other major component of a digital system
  - Circuit elements to manipulate individual bits of information
  - Load-store machine

LD	R1;
LD	R2;
ADD	R3, R2, R1;
SD	R3;

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,
  - Three basic logical operations: AND, OR, and NOT.

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.

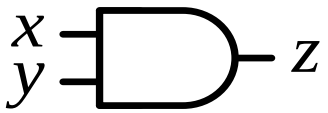
Binary Logic gates

- Truth Tables, Boolean Expressions, and Logic Gates

AND

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

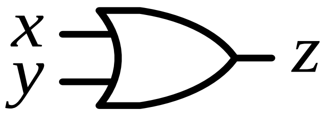
$$z = x \bullet y = x\,y$$



OR

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

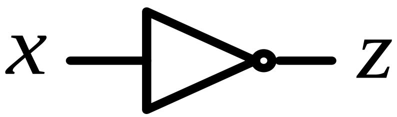
$$z = x + y$$

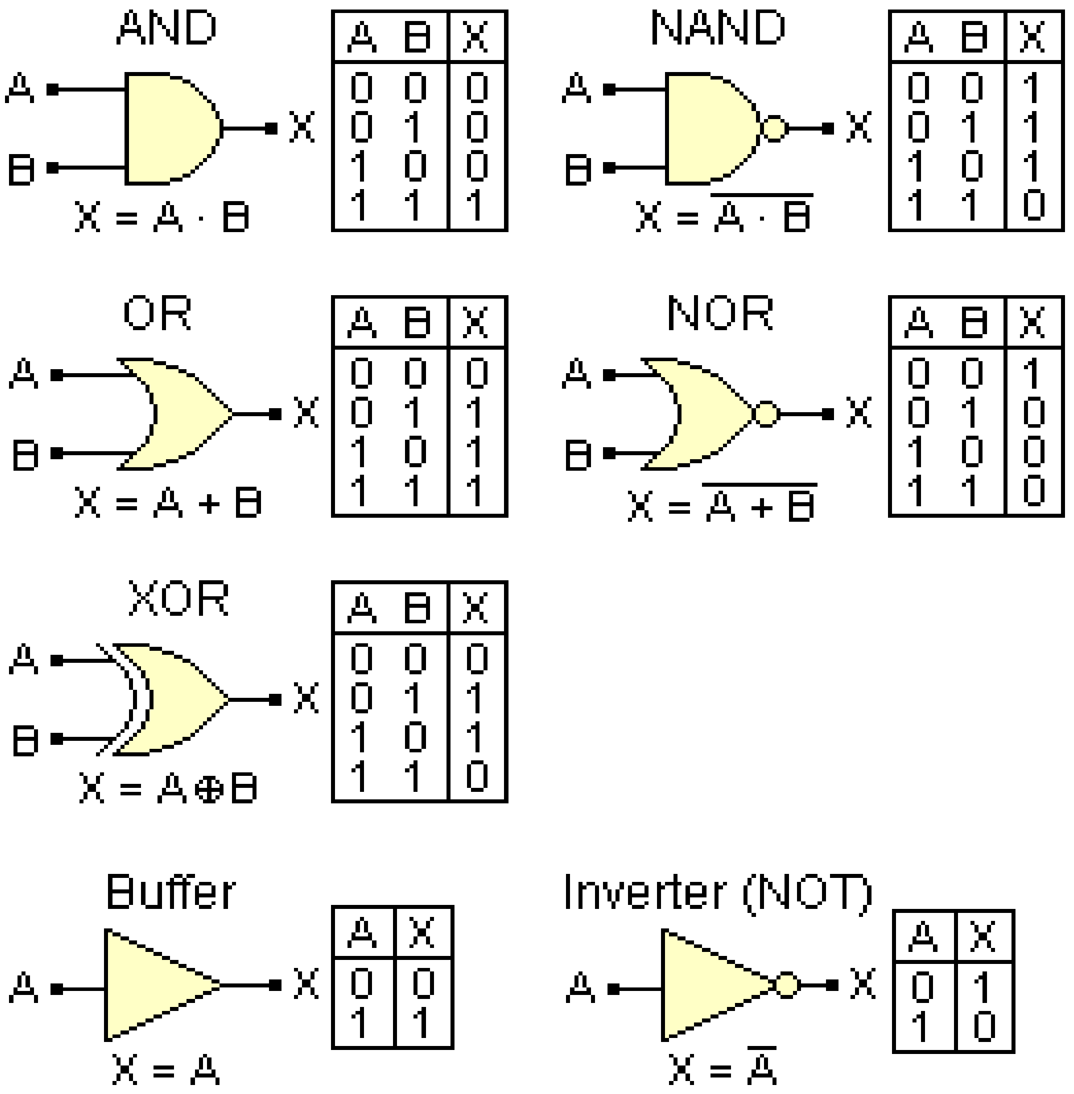


NOT

x	z
0	1
1	0

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





AND

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

NAND

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

OR

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

NOR

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

XOR

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Buffer

A	X
0	0
1	1

Inverter (NOT)

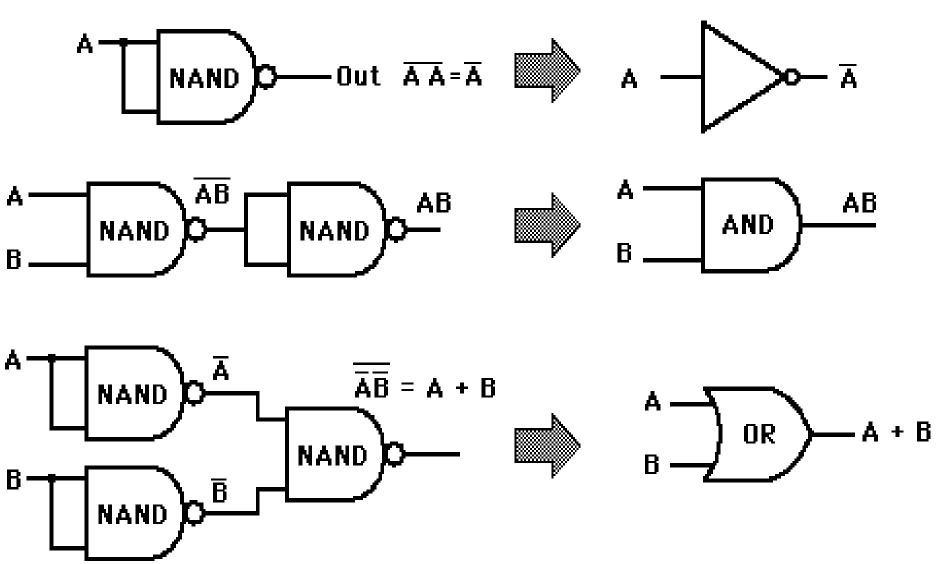
A	X
0	1
1	0

Logic Function	Boolean Notation
AND	A.B
OR	A+B
NOT	$\overline{A}$
NAND	$\overline{A.B}$
NOR	$\overline{A+B}$
EX-OR	$(A.\overline{B}) + (\overline{A}.B)$ or $A \oplus B$
EX-NOR	$(\overline{A}.\overline{B})+$ or $\overline{A \oplus B}$

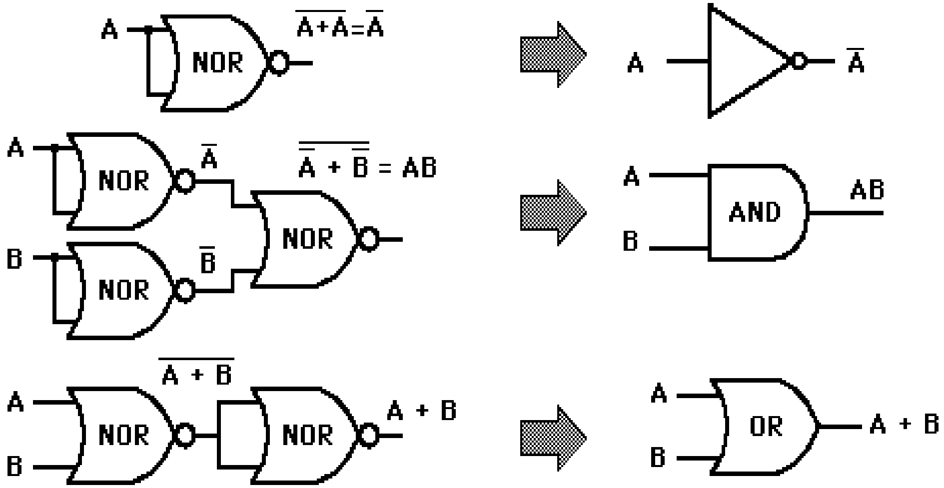
Universal Gate

- **NAND and NOR** Gates are called ***Universal Gates*** because AND, OR and NOT gates can be implemented &created by using these gates.

NAND Gate Implementations

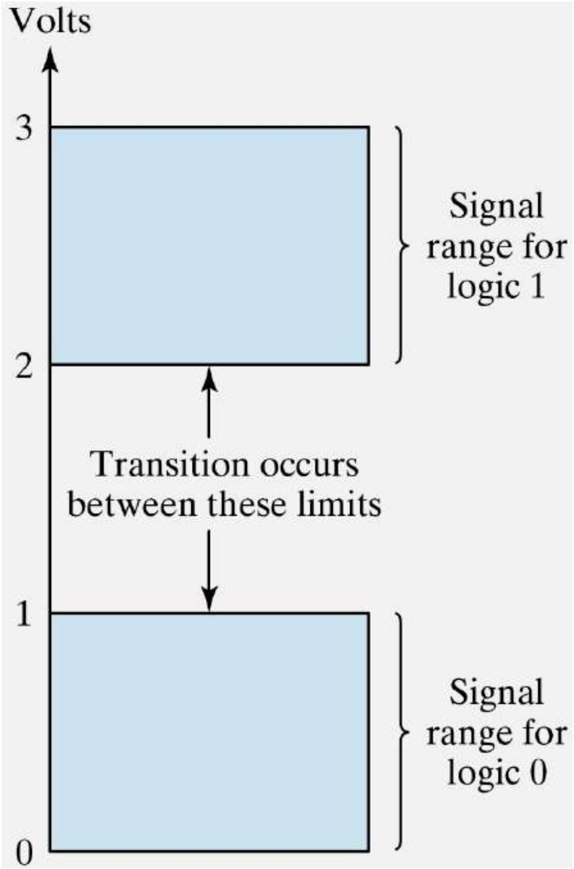


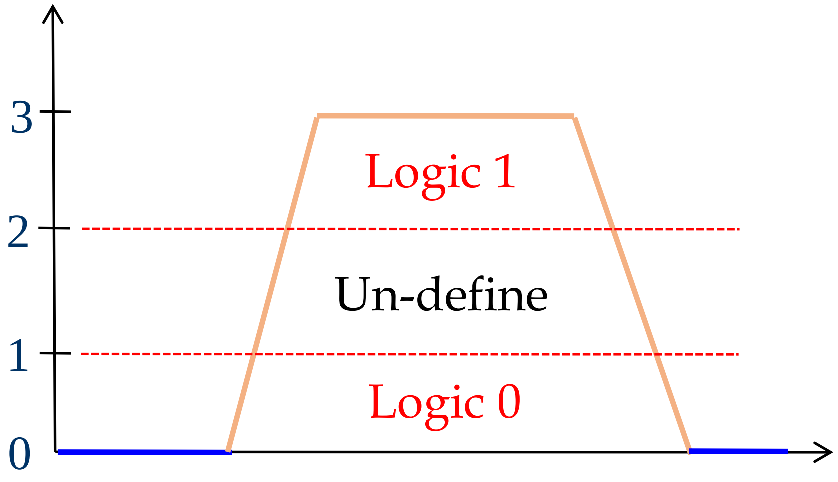
NOR Gate Implementations



Binary Logic

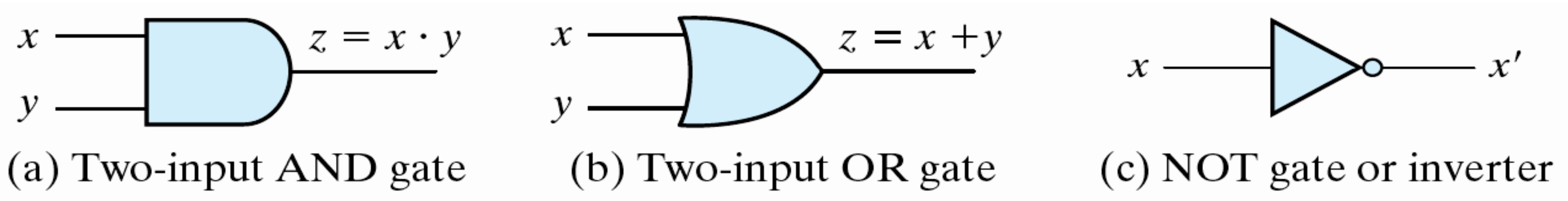
- Logic gates
  - Example of binary signals

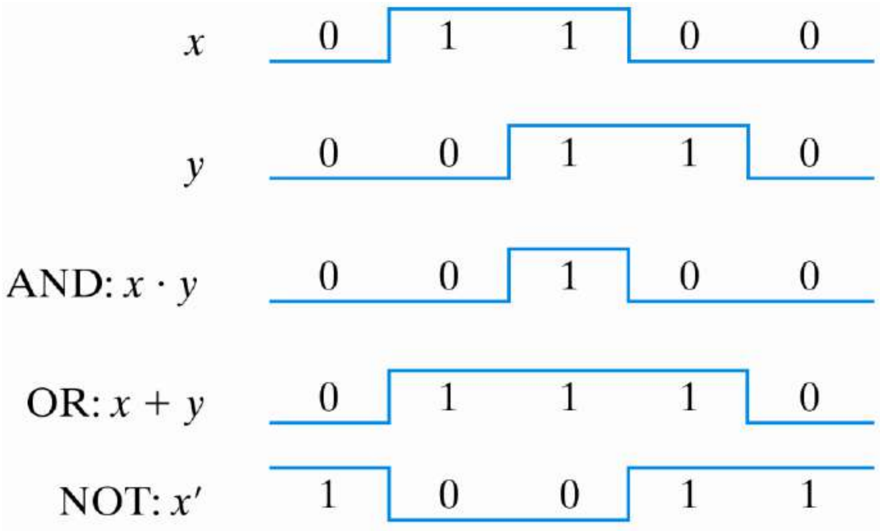




Binary Logic

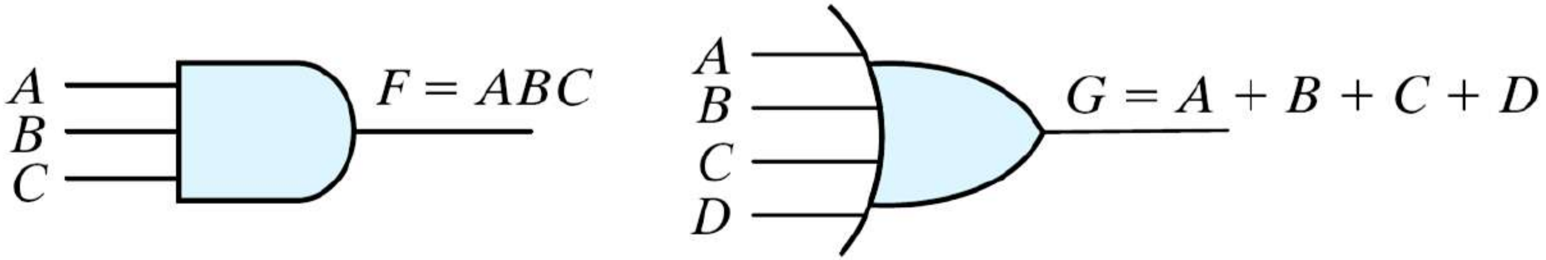
- Logic gates
  - Graphic Symbols and Input-Output Signals for Logic gates:





Binary Logic

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



(a) Three-input AND gate (b) Four-input OR gate

Fig. 1.6 Gates with multiple inputs

Boolean Algebra

Boolean Algebra : George Boole(English mathematician), 1854

- **Invented by George Boole in 1854**
- **An algebraic structure defined by a set B = {0, 1}, together with two binary operators (+ and ·) and a unary operator ( )**

"An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities"

Boolean Algebra

{(1,0), Var, (NOT, AND, OR), Thms}

- Mathematical tool to expression and analyze digital (logic) circuits
- Claude Shannon, the first to apply Boole's work, 1938
  - "A Symbolic Analysis of Relay and Switching Circuits" at MIT
- This chapter covers Boolean algebra, Boolean expression and its evaluation and simplification, and VHDL program

Basic Functions and Basic Functions

Boolean functions : NOT, AND, OR,

Exclusive OR(XOR) : Odd function   Exclusive NOR(XNOR) : Even function(equivalence)

Boolean functions for (a) AND, (b) OR, (c) XOR, and (d) NOT

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

(a)

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

(b)

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

(c)

x	x'
0	1
1	0

(d)

Basic functions

- AND $Z=X \cdot Y$ or $Z=XY$

  Z=1 if and only if X=1 and Y=1, otherwise Z=0

- OR $Z=X + Y$

  Z=1 if X=1 or if Y=1, or both X=1and Y=1. Z=0 if and only if X=0 and Y=0

- NOT $Z=X'$ or

  Z=1 if X=0, Z=0 if X=1

Boolean functions for (a) NAND, (b) NOR, and (c) XNOR

x	y	NAND
0	0	1
0	1	1
1	0	1
1	1	0

(a)

x	y	NOR
0	0	1
0	1	0
1	0	0
1	1	0

(b)

x	y	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

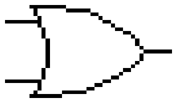
(c)

All possible binary boolean functions

x	y	0	$\wedge$	xy'	x	$x'y$	y	$\oplus$	$\vee$	NOR	XNOR	y'	$x + y'$	x'	$x' + y$	NAND	1
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Boolean Operations and Expressions

- Boolean Addition
  - Logical OR operation

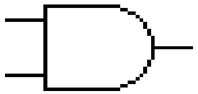
Ex 4-1) Determine the values of A, B, C, an  ke the sum term A+B’+C+D’

Sol) all literals must be ‘0’ for the sum term to be ‘0’

A+B’+C+D’=0+1’+0+1’=0$\rightarrow$ A=0, B=1, C=0, and D=1

- Boolean Multiplication
  - Logical AND operation

Ex 4-2) Determine the values of A, B, C, and D for AB’CD’=1

Sol) all literals must be ‘1’ for the produc 

AB’CD’=10’10’=1$\rightarrow$ A=1, B=0, C=1, and D=0

Basic Identities of Boolean Algebra

Basic Identities of Boolean Algebra

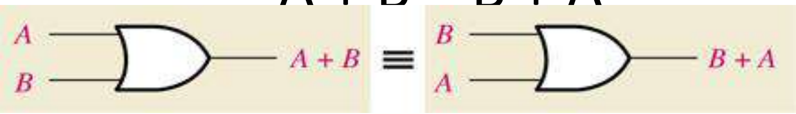
1.	$X + 0 = X$	2.	$X \cdot 1 = X$	
3.	$X + 1 = 1$	4.	$X \cdot 0 = 0$	
5.	$X + X = X$	6.	$X \cdot X = X$	
7.	$X + \overline{X} = 1$	8.	$X \cdot \overline{X} = 0$	
9.	$\overline{\overline{X}} = X$			
10.	$X + Y = Y + X$	11.	$XY = YX$	Commutative
12.	$X + (Y + Z) = (X + Y) + Z$	13.	$X(YZ) = (XY)Z$	Associative
14.	$X(Y + Z) = XY + XZ$	15.	$X + YZ = (X + Y)(X + Z)$	Distributive
16.	$\overline{X + Y} = \overline{X} \cdot \overline{Y}$	17.	$\overline{X \cdot Y} = \overline{X} + \overline{Y}$	DeMorgan's

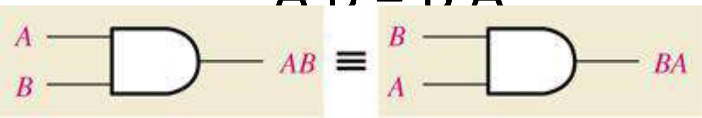
The relationship between a single variable X, its complement X′, and the binary constants 0 and 1

Laws of Boolean Algebra

Commutative Law: the order of literals does not matter

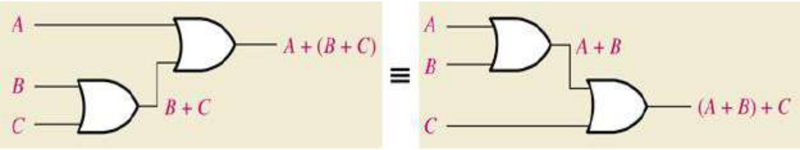
A + B = B + A　　　　A B = B A



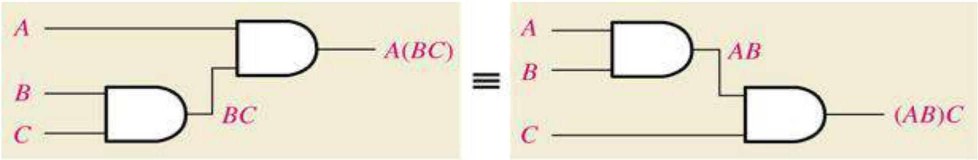


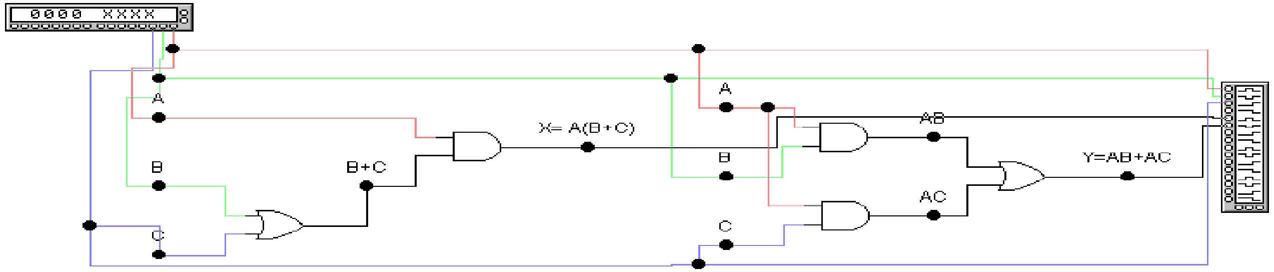
Associative Law: the grouping of literals does not matter

A + (B + C) = (A + B) + C (=A+B+C)　　　　A(BC) = (AB)C (=ABC)



B + AC





Rules of Boolean Algebra

- ✓ $A+0=A$ In math if you add 0 you have changed nothing in Boolean Algebra ORing with 0 changes nothing
- ✓ $A\cdot 0=0$ In math if 0 is multiplied with anything you get 0. If you AND anything with 0 you get 0
- ✓ $A\cdot 1=A$ ANDing anything with 1 will yield the anything

- ✓ $A+A=A$ ORing with itself will give the same result
- ✓ $A+A'=1$ Either A or A' must be 1 so A + A' =1
- ✓ $A\cdot A=A$ ANDing with itself will give the same result
- ✓ $A\cdot A'=0$ In digital Logic 1' =0 and 0' =1, so AA'=0 since one of the inputs must be 0.
- ✓ $A=(A')'$ If you not something twice you are back to the beginning

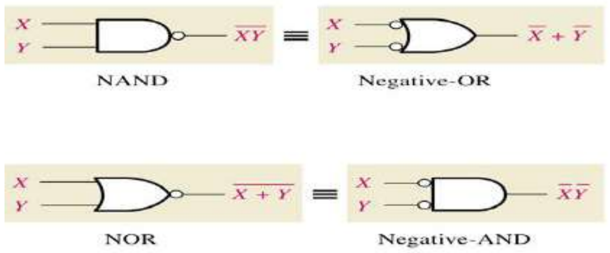
✓ $A + A'B = A + B$

If A is 1 the output is 1   If A is 0 the output is B

✓ $A + AB = A$

✓ $(A + B)(A + C) = A + BC$

- **DeMorgan's Theorem**
  - $F'(A, A', \cdot, +, 1, 0) = F(A', A, +, \cdot, 0, 1)$
  - $(A \bullet B)' = A' + B'$ and $(A + B)' = A' \bullet B'$
  - DeMorgan's theorem will help to simplify digital circuits using NORs and NANDs his theorem states

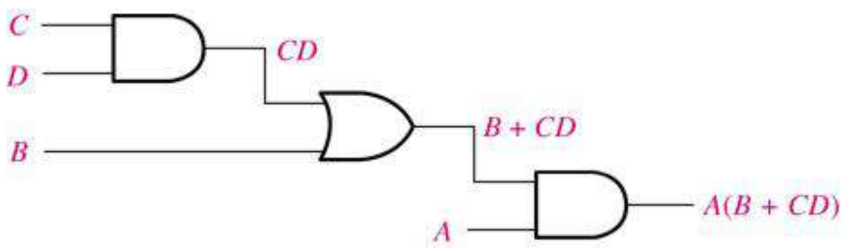


Inputs		Output	
X	Y	$\overline{XY}$	$\overline{X} + \overline{Y}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Inputs		Output	
X	Y	$\overline{X + Y}$	$\overline{X}\overline{Y}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Boolean Analysis of Logic Circuits

- Constructing a Truth Table for a Logic Circuit



- Convert the expression into the min-terms containing all the input literals
- Get the numbers from the min-terms
- Putting '1's in the rows corresponding to the min-terms and '0's in the remains

Ex) A(B+CD)=AB(C+C') (D+D') +A(B+B')CD =ABC(D+D') +ABC'(D+D') +ABCD+AB'CD =ABCD+ABCD'+ABC'D+ABC'D' +ABCD+AB'CD =ABCD+ABCD'+ABC'D+ABC'D' +AB'CD =m11+m12+m13+m14+m15=Σ(11,12,13,14,15)

A(B+CD) = m11+m12+m13+m14+m15 =Σ(11,12,13,14,15)

Input				Output
A	B	C	D	A(B+CD)
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Standard Forms of Boolean Expressions

- ❑The Sum-of-Products(SOP) Form   Ex) AB+ABC, ABC+CDE+B'CD'
- ❑The Product-of-Sums(POS) Form Ex) (A+B)(A+B+C), (A+B+C)(C+D+E)(B'+C+D')
- ❑Principle of Duality : SOP ⇔ POS
- ❑Domain of a Boolean Expression : The set of variables contained in the expression

  Ex) A'B+AB'C : the domain is {A, B, C}

✓Standard SOP Form (Canonical SOP Form)

- For all the missing variables, apply (x+x')=1 to the AND terms of the expression
- List all the min-terms in forms of the complete set of variables in ascending order

Ex : Convert the following expression into standard SOP form: AB'C+A'B'+ABC'D

Sol) domain={A,B,C,D}, AB'C(D'+D)+A'B'(C'+C)(D'+D)+ABC'D
=AB'CD'+AB'CD+A'B'C'D'+A'B'C'D+A'B'CD'+A'B'CD+ABC'D
=1010+1011+0000+0001+0010+0011+1101 =0+1+2+3+10+11+13 = $\Sigma(0,1,2,3,10,11,13)$

Standard POS Form (Canonical POS Form)

- For all the missing variables, apply (x'x)=0 to the OR terms of the expression
- List all the max-terms in forms of the complete set of variables in ascending order

Ex : Convert the following expression into standard POS form:

(A+B'+C)(B'+C+D')(A+B'+C'+D)

Sol) domain={A,B,C,D}, (A+B'+C)(B'+C+D')(A+B'+C'+D)

=(A+B'+C+D'D)(A'A+B'+C+D')(A+B'+C'+D)

=(A+B'+C+D')(A+B'+C+D)(A'+B'+C+D')(A+B'+C+D')(A+B'+C'+D)=(0100))(0101)(0110)(1101)= Π(4,5,6,13)

Converting Standard SOP to Standard POS

Step 1. Evaluate each product term in the SOP expression. Determine the binary numbers that represent the product terms

Step 2. Determine all of the binary numbers not included in the evaluation in Step 1

Step 3. Write in equivalent sum term for each binary number Step 2 and expression in POS form

Ex : Convert the following SOP to POS

Sol) SOP= A'B'C'+A'BC'+A'BC+AB'C+ABC=0+2+3+5+7 $=\Sigma(0,2,3,5,7)$

POS=(1)(4)(6) = $\Pi(1, 4, 6)$ (=(A+B+C')(A'+B+C)(A'+B'+C))

- **SOP and POS Observations**
  - Canonical Forms (Sum-of-minterms, Product-of-Maxterms), or other standard forms (SOP, POS) differ in complexity
  - Boolean algebra can be used to manipulate equations into simpler forms
  - Simpler equations lead to simpler implementations

Summary of Minterms and Maxterms

- There are 2^n minterms and maxterms for Boolean functions with n variables.
- Minterms and maxterms are indexed from 0 to $2^n - 1$
- Any Boolean function can be expressed as a logical sum of minterms and as a logical product of maxterms
- The complement of a function contains those minterms not included in the original function
- The complement of a sum-of-minterms is a product-of-maxterms with the same indices

Dual of a Boolean Expression

- **To changing 0 to 1 and + operator to – vise versa for a given boolean function**

❑ **Example: $F = (A + C) \cdot B + 0$**

dual $F = (A \cdot C + B) \cdot 1 = A \cdot C + B$

❑ **Example: $G = X \cdot Y + (W + Z)$**

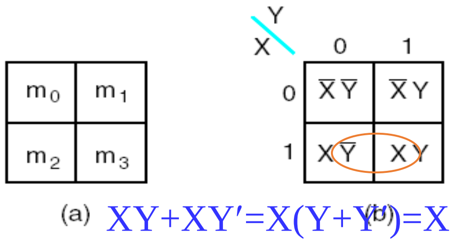
dual G =

✓ **Unless it happens to be self-dual, the dual of an expression does not equal the expression itself**

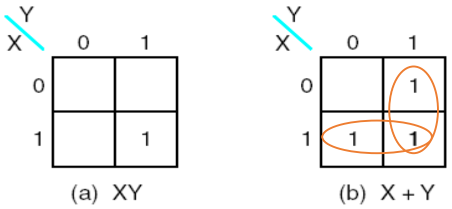
✓ **Are any of these functions self-dual?** **$(A+B)(A+C)(B+C)=(A+BC)(B+C)=AB+AC+BC$**

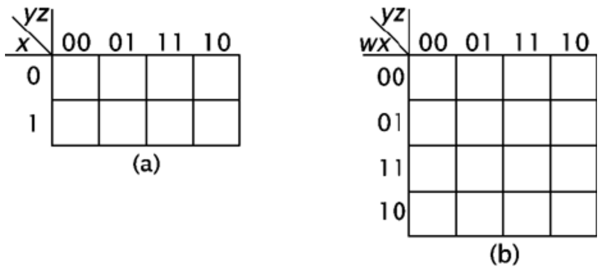
Karnaugh Map

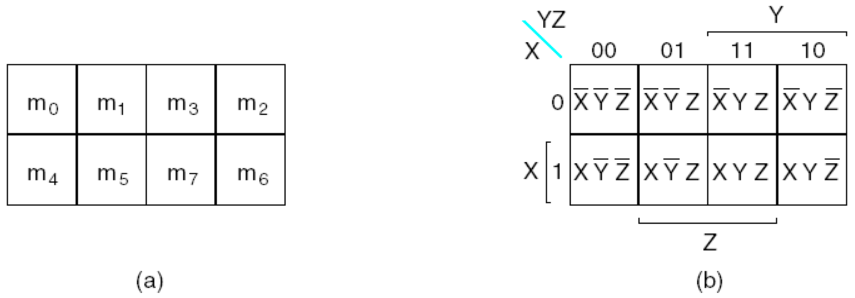
- Simplification methods
  - Boolean algebra(algebraic method)
  - Karnaugh map(map method))
  - Quine-McCluskey(tabular method)

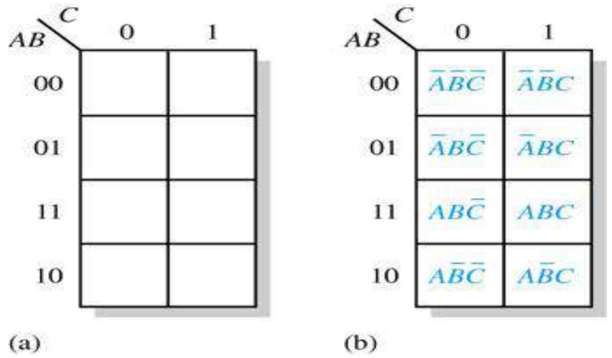


Two-Variable Map


Representation of Functions in the Map




Three-Variable Map



Syllabus

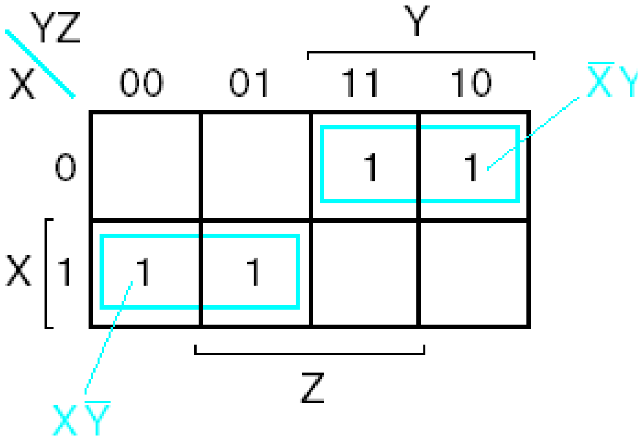
UNIT-II

K-Map Reduction - Don't Care Conditions - Adders / Subtractors- Carry Look-Ahead Adder - Code Conversion Algorithms - Design of Code Converters - Equivalence Functions. Binary/Decimal Parallel Adder/Subtractor for Signed Numbers - Magnitude Comparator - Decoders / Encoders - Multiplexers / Demultiplexers- Boolean Function Implementation using Multiplexers

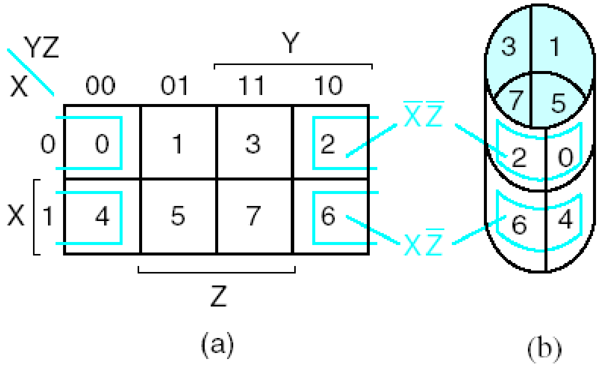
Karnaugh Map (K- Map) Steps

1. Sketch a Karnaugh map grid for the given problem.in power of 2^N Squares
2. Fill in the 1's and 0's from the truth table of sop or pos Boolean function
3. Circle groups of 1's.
   - Circle the largest groups of 2, 4, 8, etc. first.
   - Minimize the number of circles but make sure that every 1 is in a circle.
4. Write an equation using these circles.

Example) $F(X,Y,Z)=\Sigma m(2,3,4,5) = X'Y+XY'$



Example) $F(X,Y,Z)=\Sigma m(0,2,4,6) = X'Z'+XZ' = Z'(X'+X)=Z'$



Three-Variable Map: Flat and on a Cylinder to Show Adjacent Squares

Four-Variable K-Map : 16 minterms : $m_0 \sim m_{15}$

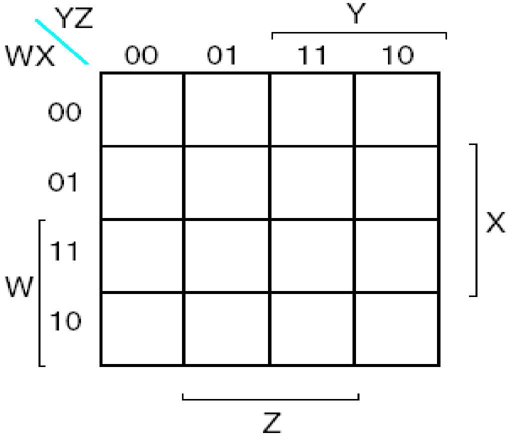
Rectangle group

- 2-squares(minterms) : 3-literals product term
- 4-squares : 2-literals product term
- 8-squares : 1-literals product term
- 16-squares : logic 1

AB \ CD	00	01	11	10
00	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}\bar{B}CD$	$\bar{A}\bar{B}C\bar{D}$
01	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}B\bar{C}D$	$\bar{A}BCD$	$\bar{A}BC\bar{D}$
11	$AB\bar{C}\bar{D}$	$AB\bar{C}D$	$ABCD$	$ABC\bar{D}$
10	$A\bar{B}\bar{C}\bar{D}$	$A\bar{B}\bar{C}D$	$A\bar{B}CD$	$A\bar{B}C\bar{D}$

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

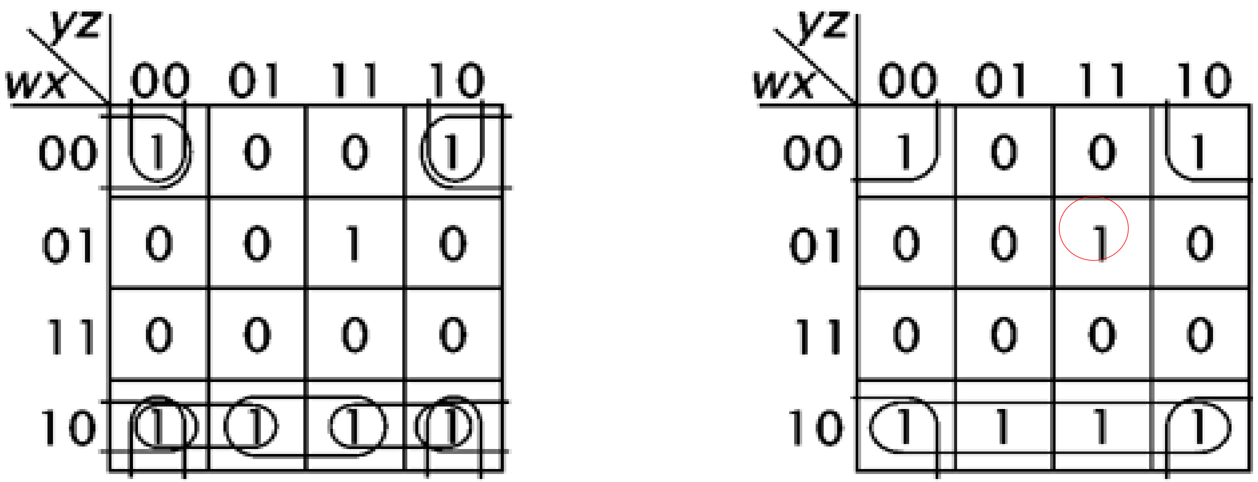
(a)



(b)

Fig. 2-17 Four-Variable Map

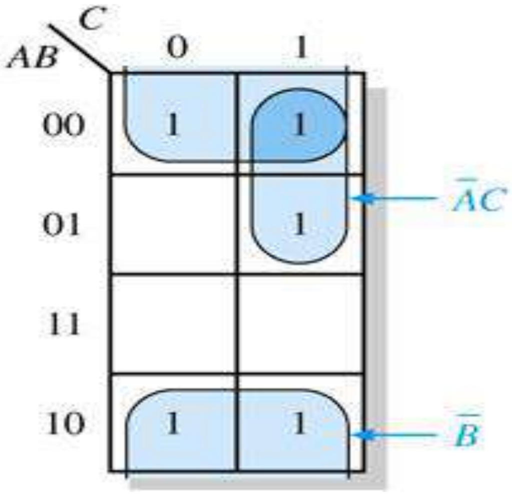
$F(W,X,Y,Z)=\Sigma m(0,2,7,8,9,10,11) = WX' + X'Z' + W'XYZ$



(a) (b)

Ex 4-28) Minimize the following expression

AB’C+A’BC+A’B’C+A’B’C’+AB’C’

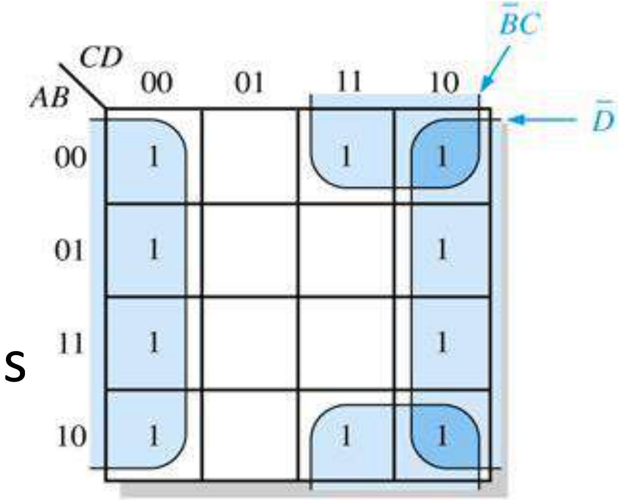


Sol) B’+A’C

Ex Minimize the following expression

B’C’D’+A’BC’D’+ABC’D’+A’B’CD+AB’CD+A’B’CD’+A’BCD’ +ABCD’+AB’CD’

Sol) D’+B’C



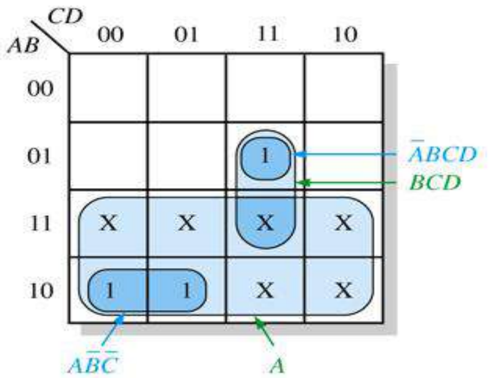
❑Don’t Care Conditions

- it really does not matter since they will never occur(its output is either ‘0’ or ‘1’)
- The don’t care terms can used to advantage on the Karnaugh map

Inputs A B C D	Output Y
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	0
0 1 0 0	0
0 1 0 1	0
0 1 1 0	0
0 1 1 1	1
1 0 0 0	1
1 0 0 1	1
1 0 1 0	X
1 0 1 1	X
1 1 0 0	X
1 1 0 1	X
1 1 1 0	X
1 1 1 1	X

Don’t cares

(a) Truth table



(b) Without “don’t cares” $Y = A\bar{B}\bar{C} + \bar{A}BCD$
With “don’t cares” $Y = A + BCD$

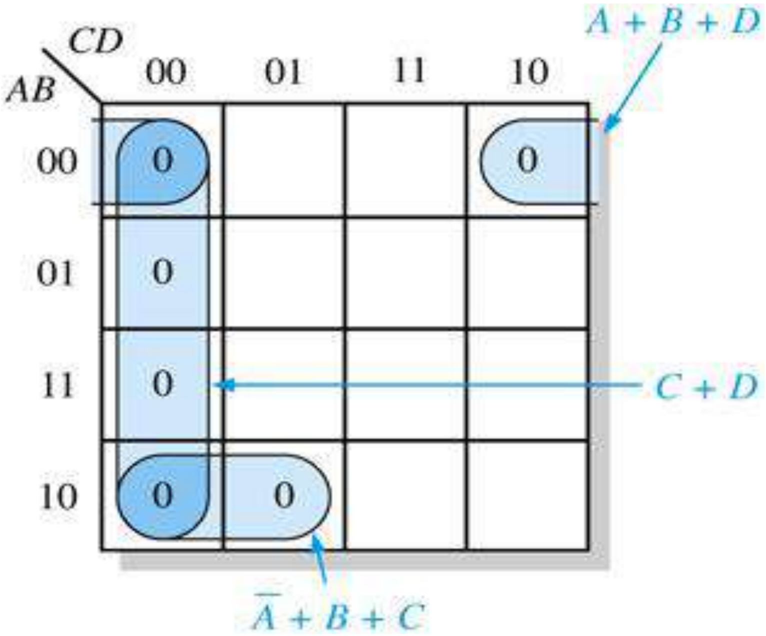
Ex K- Map for POS (B+C+D)(A+B+C’+D)(A’+B+C+D’)(A+B’+C+D)(A’+B’+C+D)

Sol) (B+C+D)=(A’A+B+C+D)=(A’+B+C

(1+0+0+0)(0+0+0+0)(0+0+1+0)

(1+0+0+1)(0+1+0+0)(1+1+0+0)

F=(C+D)(A’+B+C)(A+B+D)

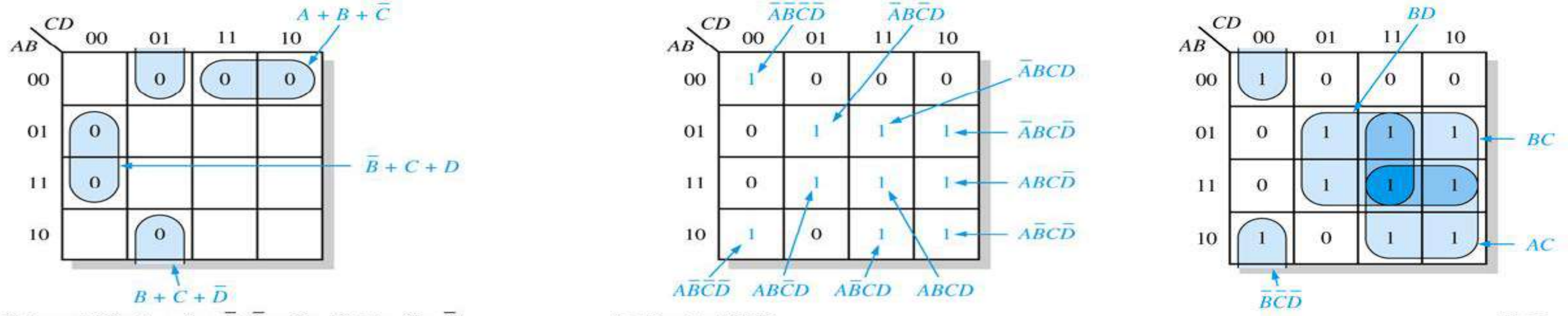


❑Converting Between POS and SOP Using the K–map

Ex 4-33) (A’+B’+C+D)(A+B’+C+D)

(A+B+C+D’)(A+B+C’+D’) (A’+B+C+D’)

(A+B+C’+D)



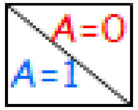
(a) Minimum POS: $(A + B + \bar{C})(\bar{B} + C + D)(B + C + \bar{D})$

(b) Standard SOP:
$\bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BCD + \bar{A}BC\bar{D} + ABC\bar{D} + A\bar{B}C\bar{D} +$
$A\bar{B}\bar{C}\bar{D} + AB\bar{C}D + A\bar{B}CD + ABCD$

(c) Minimum SOP: $AC + BC + BD + \bar{B}\bar{C}\bar{D}$

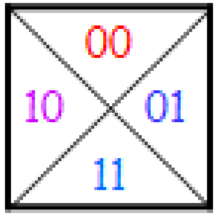
- Five Variable K-Map : {A,B,C,D,E}

DE \ BC	00	01	11	10
00	0 / 16	1 / 17	3 / 19	2 / 18
01	4 / 20	5 / 21	7 / 23	6 / 22
11	12 / 28	13 / 29	15 / 31	14 / 30
10	8 / 24	9 / 25	11 / 27	10 / 26



- Six Variable K-Map : {A,B,C,D,E,F}

AB



EF \ CD	00	01	11	10
00	0, 32, 16, 48	1, 33, 17, 49	3, 35, 19, 51	2, 34, 18, 50
01	4, 36, 20, 52	5, 37, 21, 53	7, 39, 23, 55	6, 38, 22, 54
11	12, 44, 28, 60	13, 45, 29, 61	15, 47, 31, 62	14, 46, 30, 63
10	8, 40, 24, 56	9, 41, 25, 57	11, 43, 27, 59	10, 42, 26, 58

- **Step 1** − Arrange the given min terms in an **ascending order** and make the groups based on the number of ones present in their binary representations. **- 'n+1' groups**

- **Step 2** − Compare the min terms present in **successive groups**. If there is a change in only one-bit position, then take the pair of those two min terms. Place this symbol '_' in the differed bit position and keep the remaining bits as it is.

- **Step 3** − Repeat step2 with newly formed terms till we get all **prime implicants**.

- **Step 4** − Formulate the **prime implicant table**. It consists of set of rows and columns. Place '1' in the cells corresponding to the min terms that are covered in each prime implicant.

- **Step 5** − Find the essential prime implicates by observing each column. Those essential prime implicants will be part of the simplified Boolean function.

- **Step 6** − Reduce the prime implicant table by removing the row of each essential prime implicant and the columns corresponding to the min terms that are covered in that essential prime implicant. Repeat step 5 for Reduced prime implicant table. Stop this process when all min terms of given Boolean function are over.

1. Simplify the following expression to sum of product using Tabulation Method

$$F(a,b,c,d) = \sum(0,1,2,3,4,6,7,11,12,15)$$

Solution:

a. **Determination of Prime Implicants**

Group 0	m0: 0000	√	(0,1) 000-	√	(0,1,2,3) 00--
			(0,2) 00-0	√	(0,2,4,6) 0--0
			(0,4) 0-00	√	(0,2,1,3) 00-- redundant
					(0,4,2,6) 0--0 redundant
Group 1	m1: 0001	√	(1,3) 00-1	√	(2,3,6,7) 0-1-
	m2: 0010	√	(2,3) 001-	√	(2,6,3,7) 0-1-
	m4: 0100	√	(2,6) 0-10	√	
			(4,6) 01-0	√	
			(4,12) -100		
Group 2	m3: 0011	√	(3,7) 0-11	√	(3,7,11,15) --11
	m6: 0110	√	(3,11) -011	√	(3,11,7,15) --11 redundant
	m12: 1100	√	(6,7) 011-	√	
Group 3	m7: 0111	√	(7,15) -111	√	
	m11: 1011	√	(11,15) 1-11	√	
Group 4	m15: 1111	√			

b. **Prime Implicant Chart:**

	PI	m0	m1	m2	m3	m4	m6	m7	m11	m12	m15
EPI	4 12					x				x	
EPI	0 1 2 3	x	x	x	x						
	0 2 4 6	x		x		x	x				
	2 3 6 7			x	x		x	x			
EPI	3 7 11 15				x			x	x		x
		√	√	√	√	√	√	√	√	√	√

$$f(a,b,c,d) = bc'd' + a'b' + cd + a'd'$$

3. Simplify the following expression to product of sum using Tabulation Method

$$F(a,b,c,d) = \prod(1,3,5,7,13,15)$$

Solution:

a. Determination of Prime Implicants

Group 0					
Group 1	M1: 0001	√	(1,3) 00-1	√	(1,3,5,7) 0--1
			(1,5) 0-01	√	(1,5,3,7) 0--1 redundant
Group 2	M3: 0011	√	(3,7) 0-11	√	(5,7,13,15) -1-1
	M5: 0101	√	(5,7) 01-1	√	(5,13,7,15) -1-1 redundant
			(5,13) -101	√	
Group 3	M7: 0111	√	(7,15) -111	√	
	M13: 1101	√	(13,15) 11-1	√	
Group 4	M15: 1111	√			

b. Prime Implicant Chart:

	PI	M1	M3	M5	M7	M13	M15
EPI	1 3 5 7	⊗	⊗	x	x		
EPI	5 7 13 15			x	x	⊗	⊗
		√	√	√	√	√	√

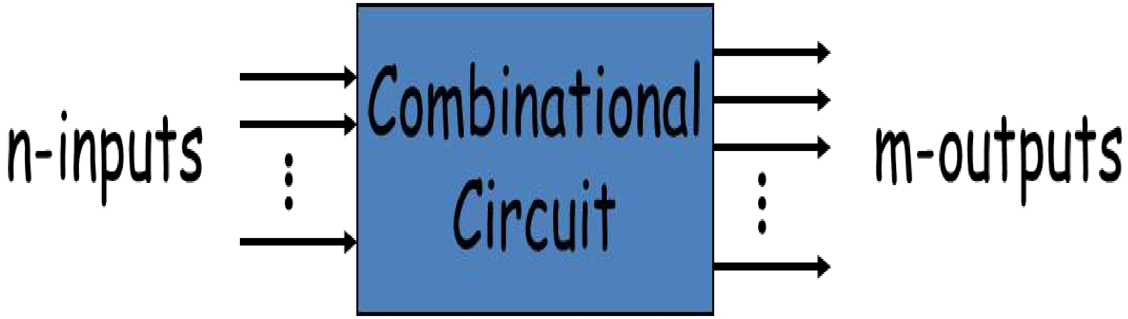
$$f(a,b,c,d) = (a + d')(b' + d')$$

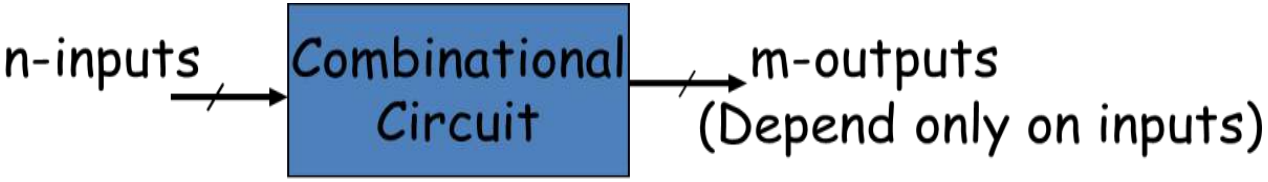
Digital Circuits

- ***Digital circuits are two types***

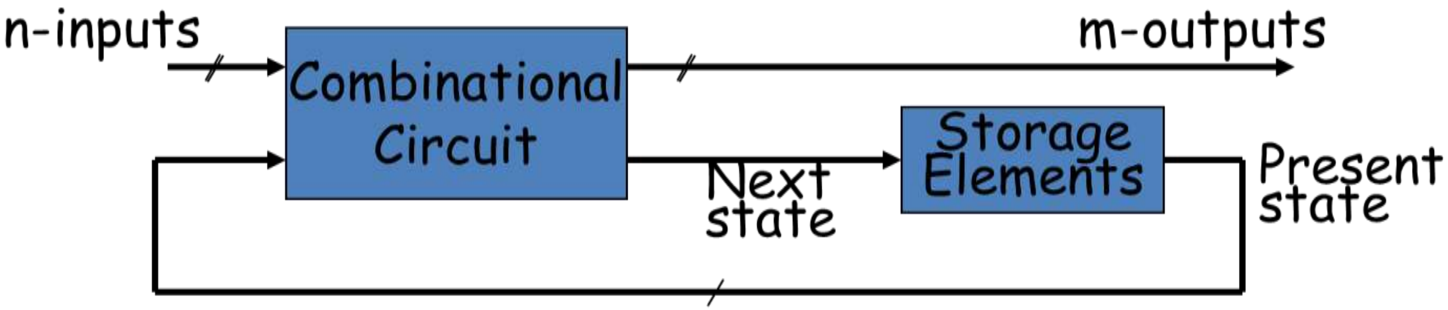
1. ***Combinational circuit*** **consists of logic gates whose outputs at any time are determined directly from the present combination of inputs without regard to previous inputs.**

2. ***Sequential Circuit*** **employ memory elements in addition to logic gates. Their outputs are a function of the inputs and the state of the memory elements.**



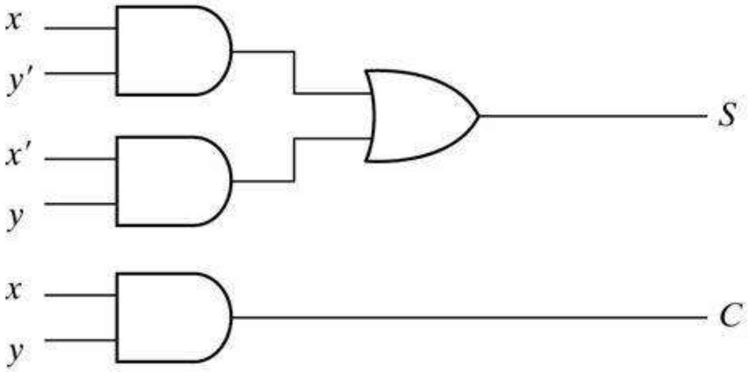


Combinational Circuit

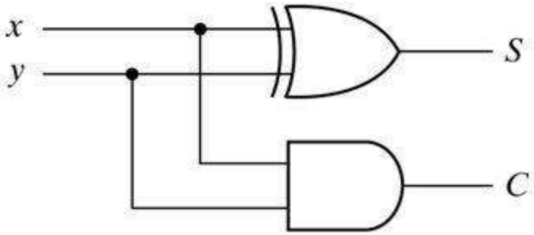


Sequential Circuit

x y	C S
0 0	0 0
0 1	0 1
1 0	0 1
1 1	1 0

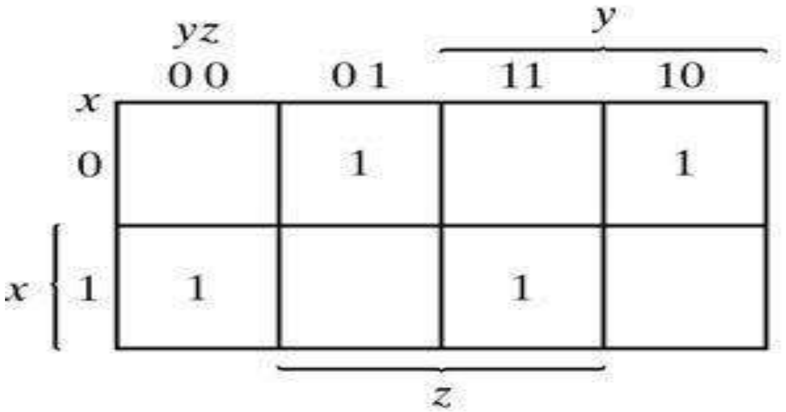


(a) $S = xy' + x'y$
$C = xy$

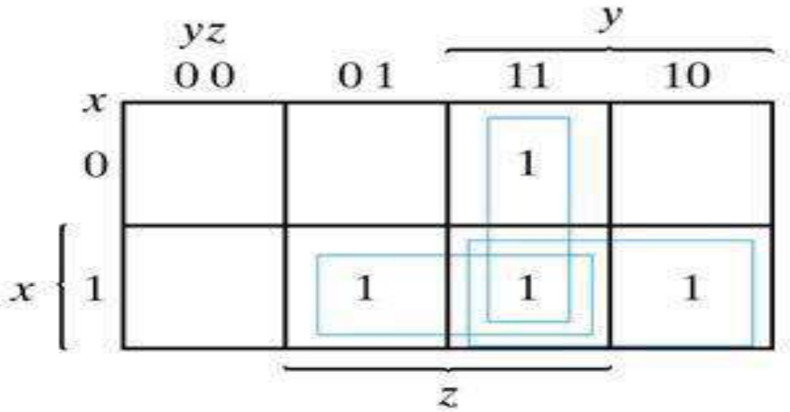


(b) $S = x \oplus y$
$C = xy$

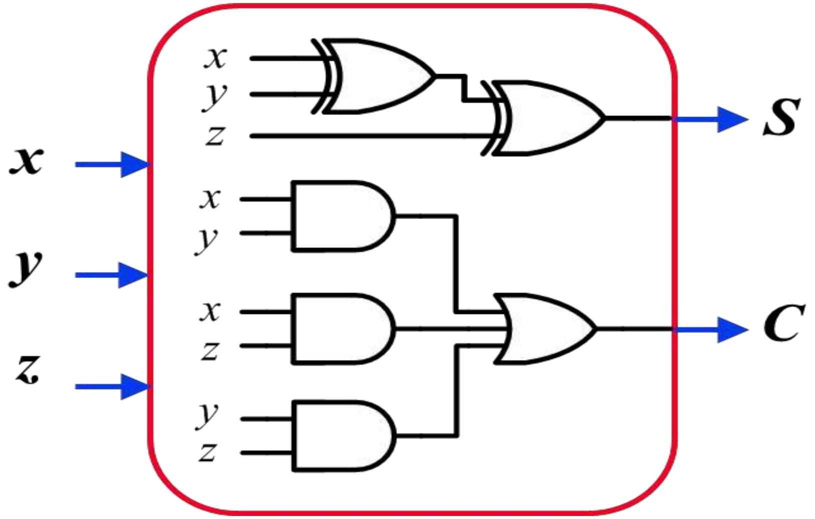
x y z	C S
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	0 1
1 0 1	1 0
1 1 0	1 0
1 1 1	1 1

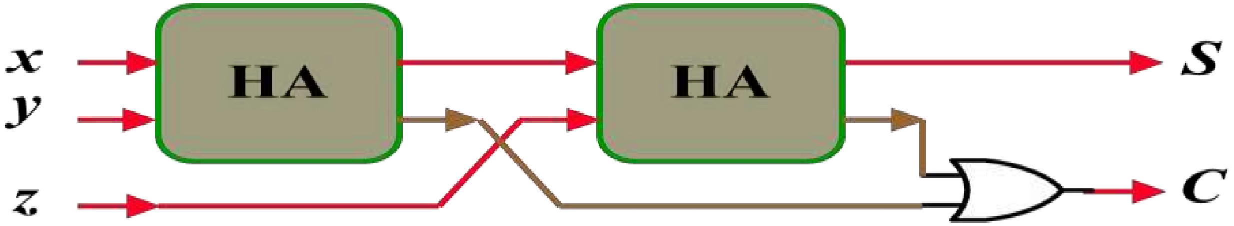


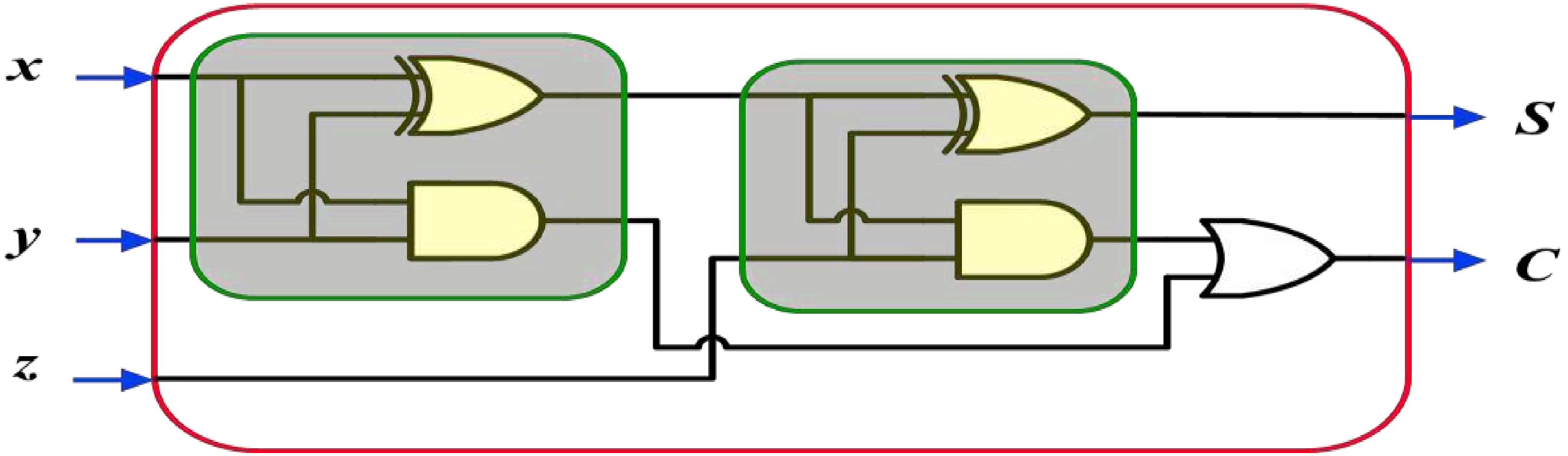
$S = x'y'z + x'yz' + xy'z' + xyz$



$$S = xy + xz + yz$$
$$= xy + xy'z + x'yz$$


x
y
z
x
y
z
x
y
x
z
y
z
S
C


x
y
z
HA
HA
S
C


x
y
z
S
C

Carry generation: output carry is produced internally by the FA.carry is generated only when both input bits are 1s.the generated carry C is expressed as the AND function of two input bits A and B so C=AB.

Carry propagation: occurs when the i/p carry is rippled to become the o/p carry.an i/p carrymay be propagated by the full adder when either or both of the i/p bits are 1s.the propagated carry Cp is expressed as the OR function of the i/p bits ie Cp=A+B

2- Bit Parallel Adder

- LSB of two binary numbers are represented by A_1 and B_1.The next higher bit are A_2 and B_2. The resulting three sum bits are Σ_1, Σ_2 and C_O, in which the C_O becomes MSB.
- The carry output C_O of each adder is connected as the carry input of the next higher order.

$$\begin{array}{r} A_2A_1 \\ +\ B_2B_1 \\ \hline C_0\Sigma_2\Sigma_1 \end{array}$$

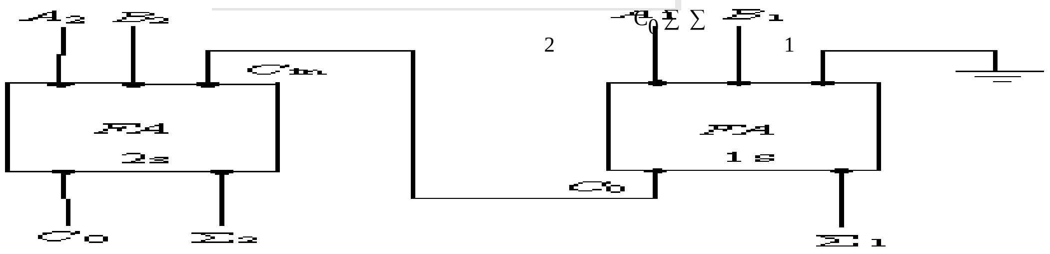
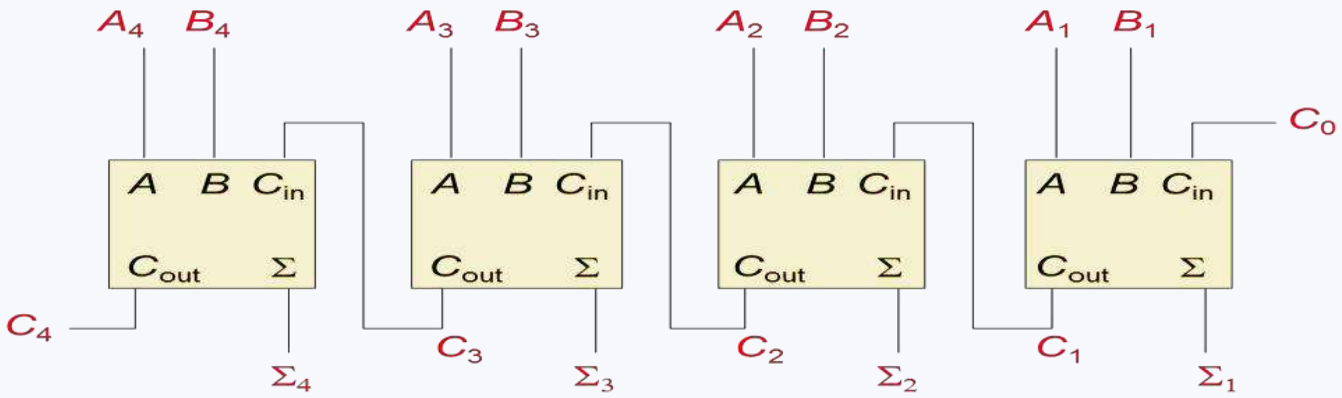


Fig : bit adder using two full adder

Four Bit Parallel Adders

- An n-bit adder requires n full adders with each output connected to the input carry of the next higher-order full adder.
- The carry output of each adder is connected to the carry input of next adder called as internal carries.



Input bit for number A	Input bit for number B	Carry bit input C_{IN}	Sum bit output S	Carry bit output C_{OUT}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Outputs: 3 output signals (GT, EQ, LT), where: GT = 1 IFF A > B EQ = 1 IFF A = B LT = 1 IFF A < B

Exactly One of these 3 outputs equals 1, while the other 2 outputs are 0`s.

Solution:

Inputs: 8-bits (A ⇒ 4-bits, B ⇒ 4-bits).A and B are two 4-bit numbers. Let $A = A_3A_2A_1A_0$, and Let $B = B_3B_2B_1B_0$.

Design of the EQ

- Define $X_i = A_i \text{ xnor } B_i = A_i B_i + A_i' B_i'$

$x_i = 1$ IFF $A_i = B_i \; \forall \; i = 0, 1, 2$ and 3

$x_i = 0$ IFF $A_i \neq B_i$

- Therefore the condition for A = B or EQ=1 IFF

$A_3 = B_3 \rightarrow (X_3 = 1)$, and $A_2 = B_2 \rightarrow (X_2 = 1)$, and $A_1 = B_1 \rightarrow (X_1 = 1)$, and $A_0 = B_0 \rightarrow (X_0 = 1)$.

- Thus, EQ=1 IFF $X_3 X_2 X_1 X_0 = 1$. In other words, $EQ = X_3 X_2 X_1 X_0$

Designing GT and LT:

- GT = 1 if A > B:

✓ If $A_3 > B_3$ $_3 = 1$ and $B_3 = 0$

✓ If $A_3 = B_3$ and $A_2 > B_2$

✓ If $A_3 = B_3$ and $A_2 = B_2$ and $A_1 > A_1$

✓ If $A_3 = B_3$ and $A_2 = B_2$ and $A_1 = B_1$ and $A_0 > B_0$

- Therefore,

$GT = A_3B_3' + X_3 A_2 B_2' + X_3 X_2 A_1 B_1' + X_3 X_2 X_1 A_0 B_0'$

Similarly, $LT = A_3'B_3 + X_3 A_2'B_2 + X_3 X_2 A_1'B_1 + X_3 X_2 X_1 A_0' B_0$

Outputs: 3 output signals (GT, EQ, LT), where: GT = 1 IFF A > B EQ = 1 IFF A = B LT = 1 IFF A < B

Exactly One of these 3 outputs equals 1, while the other 2 outputs are 0`s.

Solution:

Inputs: 8-bits (A ⇒ 4-bits, B ⇒ 4-bits).A and B are two 4-bit numbers. Let $A = A_3A_2A_1A_0$, and Let $B = B_3B_2B_1B_0$.

Design of the EQ

- Define $X_i = A_i$ xnor $B_i = A_i B_i + A_i' B_i'$

$x_i = 1$ IFF $A_i = B_i$ $\forall$ i =0, 1, 2 and 3

$x_i = 0$ IFF $A_i \neq B_i$

- Therefore the condition for A = B or EQ=1 IFF

$A_3= B_3 \rightarrow (X_3 = 1)$, and $A_2= B_2 \rightarrow (X_2 = 1)$, and $A_1= B_1 \rightarrow (X_1 = 1)$, and $A_0= B_0 \rightarrow (X_0 = 1)$.

- Thus, EQ=1 IFF $X_3 X_2 X_1 X_0 = 1$. In other words, $EQ = X_3 X_2 X_1 X_0$

Designing GT and LT:

- GT = 1 if A > B:

✓ If $A_3 > B_3$ $_3 = 1$ and $B_3 = 0$ If $A_3 = B_3$ and $A_2 > B_2$

✓ If $A_3 = B_3$ and $A_2 = B_2$ and $A_1 > A_1$

✓ If $A_3 = B_3$ and $A_2 = B_2$ and $A_1 = B_1$ and $A_0 > B_0$

- Therefore,

$GT = A_3B_3' + X_3 A_2 B_2' + X_3 X_2 A_1 B_1' + X_3 X_2 X_1A_0 B_0'$

Similarly, $LT = A_3'B_3 + X_3 A_2'B_2 + X_3 X_2 A_1'B_1 + X_3 X_2 X_1A_0' B_0$

Encoder

- Encoders typically have 2N inputs and N outputs.
- These are called 2N–to–N encoders.
- Encoders can also be devised to encode various symbols and alphabetic characters.
- The process of converting **from** familiar symbols or numbers to a coded format is called encoding.

Fig : Logical diagram of Encoder

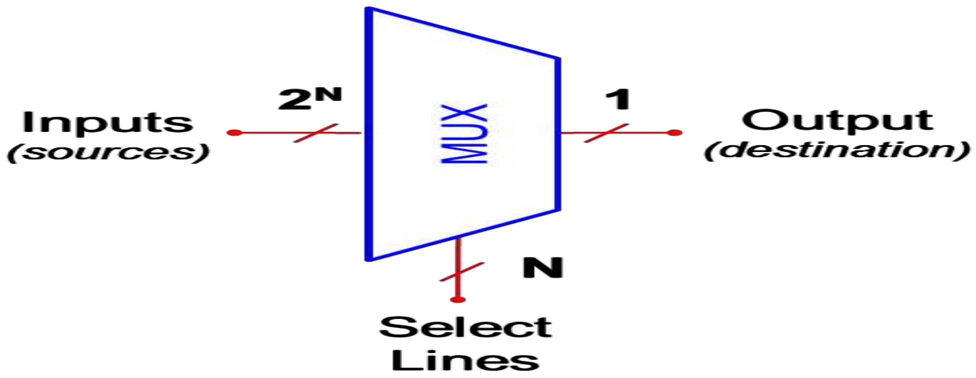
8 to-3 encoder Implementation

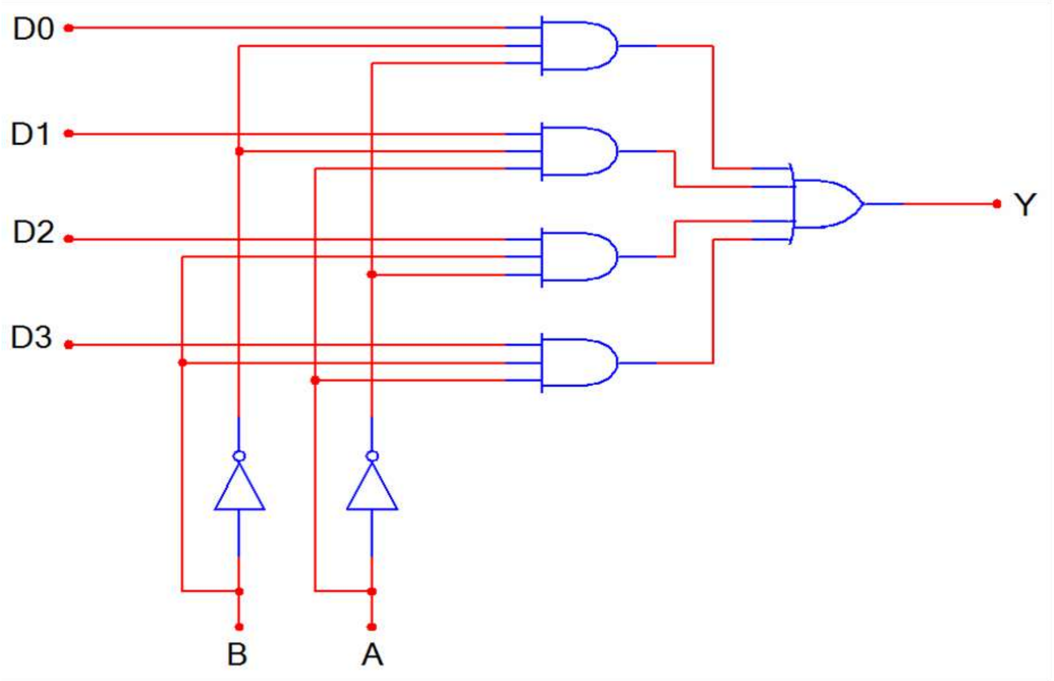
- Octal-to-Binary
- An octal to binary encoder has $2^3 = 8$ input lines D_0 to D_7 and 3 output lines Y_0 to Y_2. Below is the

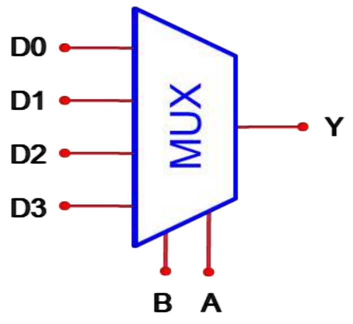
truth table for an octal to binary encoder.

Fig : Truth table for 8-3 encoder

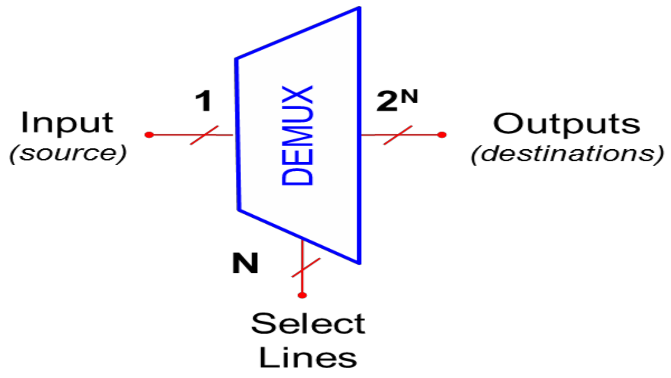
From the truth table, the outputs can be expressed by following Boolean Function. $Y_0 = D_1 + D_3 + D_5 + D_7$

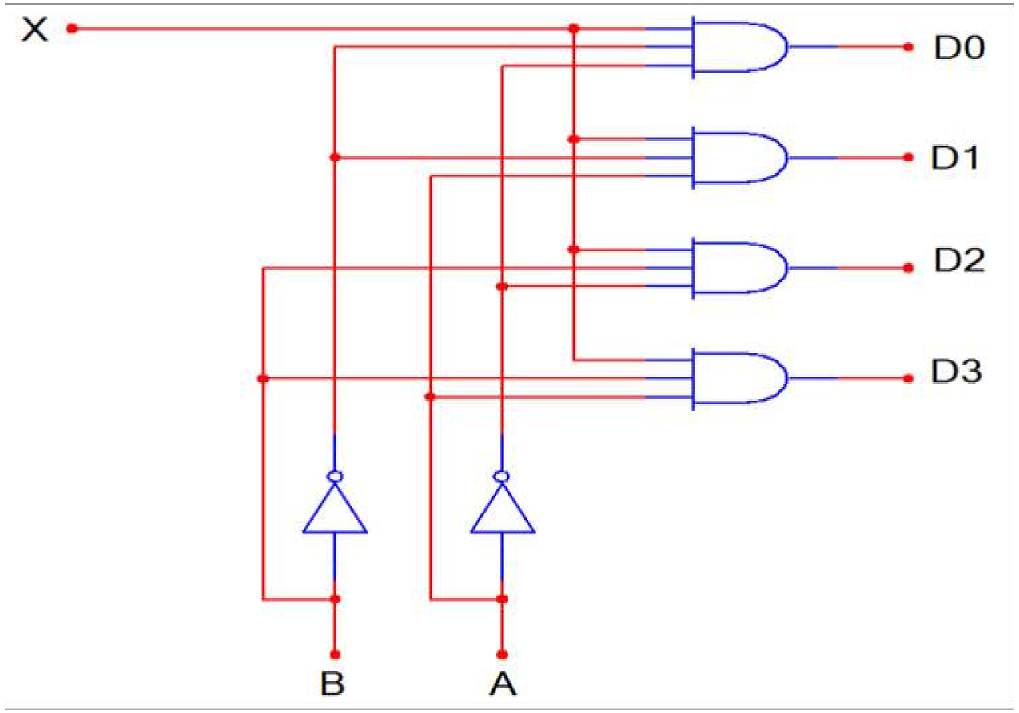


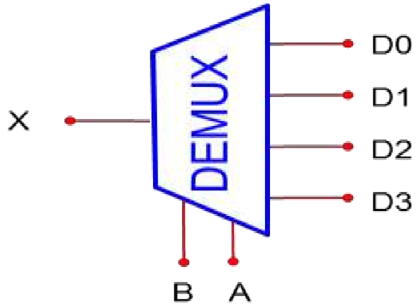




B	A	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3







B	A	D0	D1	D2	D3
0	0	X	0	0	0
0	1	0	X	0	0
1	0	0	0	X	0
1	1	0	0	0	X