Computer System Architecture

DR. Howida Youssry

Digital Computers

Computer Hardware(H/W)

- CPU
- Memory
  - Program Memory(ROM)
  - Data Memory(RAM)
- I/O Device
  - Input Device: Keyboard, Mouse, Scanner
  - Output Device: Printer, Plotter, Display
  - Storage Device(I/O): FDD, HDD, MOD

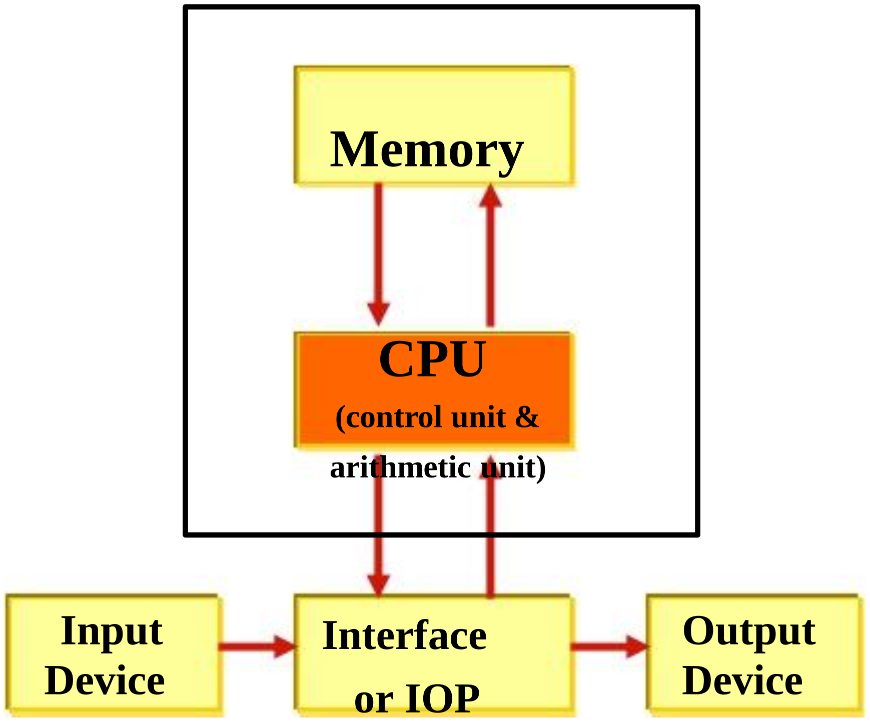


Figure 1-1 Block Diagram of a digital Computer

Logic Gates

ADC(Analog to Digital Conversion)

Signal $\longrightarrow$ Physical Quantity $\longrightarrow$ Binary Information

V, A, F, 거리   Discrete Value

$\begin{cases} 0 : 0.5v \downarrow \\ 1 : 3v \uparrow \end{cases}$

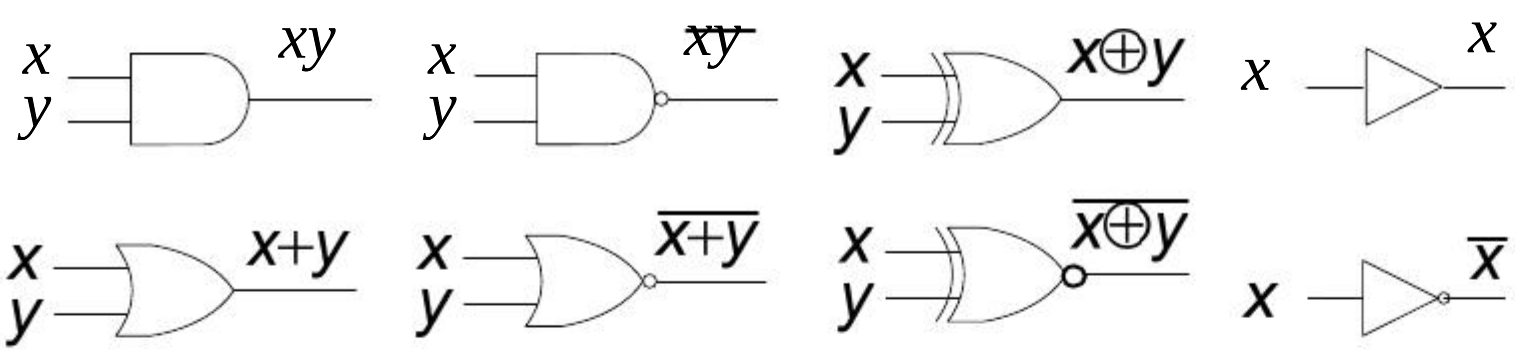
Gate

The manipulation of binary information is done by logic circuit called "gate".

Blocks of H/W that produce signals of binary 1 or 0 when input logic requirements are satisfied.

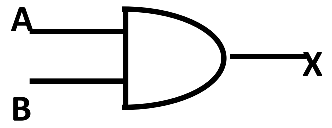
Digital Logic Gates : Fig. 1-2

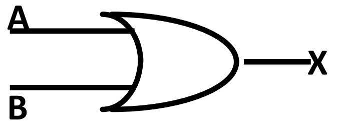
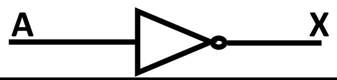
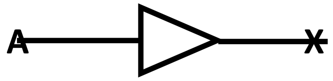
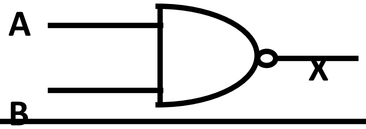
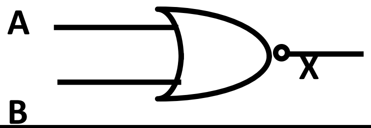
AND, OR, INVERTER, BUFFER, NAND, NOR, XOR, XNOR

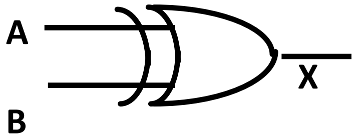


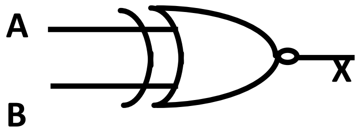
COMBINATIONAL GATES

Name	Symbol Function	Truth Table	

AND		X = A • B
		or
		X = AB
			A B | X
			0 0 | 0
			0 1 | 0
			1 0 | 0
			1 1 | 1

OR		X = A + B	A B | X 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
I		X = A	A | X 0 | 1 1 | 0
Buffer		X = A	A | X 0 | 0 1 | 1
NAND		X = (AB)’	A B | X 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		X = (A + B)’	A B | X 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0

XOR
Exclusive OR
		X = A ⊕ B
		or
		X = A’B + AB’
			A B | X
			0 0 | 0
			0 1 | 1
			1 0 | 1
			1 1 | 0

XNOR
Exclusive NOR
or Equivalence
		X = (A ⊕ B)’
		or
		X = A’B’+ AB
			A B | X
			0 0 | 1
			0 1 | 0
			1 0 | 0
			1 1 | 1

Boolean Algebra

Boolean Algebra

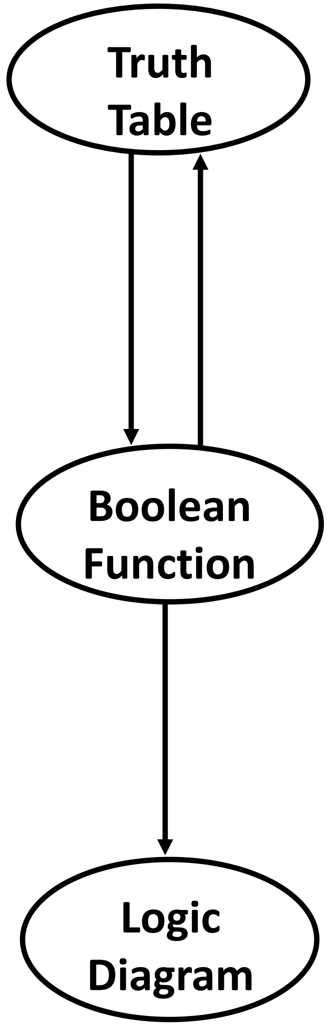
* Algebra with Binary(Boolean) Variable and Logic Operations
* Boolean Algebra is useful in Analysis and Synthesis of Digital Logic Circuits

  - Input and Output signals can be represented by Boolean Variables, and
  - Function of the Digital Logic Circuits can be represented by Logic Operations, i.e., Boolean Function(s)
  - From a Boolean function, a logic diagram can be constructed using AND, OR, and I

Truth Table

* The most elementary specification of the function of a Digital Logic Circuit is the Truth Table

  - Table that describes the Output Values for all the combinations of the Input Values, called *MINTERMS*
  - n input variables --> 2^n minterms



Boolean Algebra

Boolean Function: variable + operation

F(x, y, z) = x + y'z

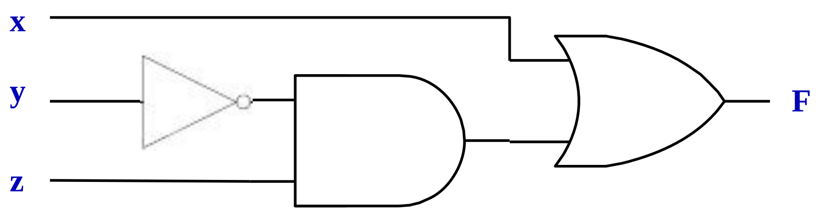
Truth Table: *Fig. 1-3(a)*
Relationship between a function and variable

2^n Combination

Variable n = 3

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

Logic Diagram: *Fig. 1-3(b)*
Algebraic Expression → Logic Diagram



Purpose of Boolean Algebra

To facilitate the analysis and design of digital circuit

Boolean function = Algebraic form = convenient tool

Truth table (relationship between binary variables : *Fig 1-3a*) $\rightarrow$ Algebraic form

Logic diagram (input-output relationship : *Fig. 1-3b*) $\rightarrow$ Algebraic form

Find simpler circuits for the same function : *by using Boolean algebra rules*

Boolean Algebra Rule : *Tab. 1-1*

[1] $x + 0 = x$	**[2]** $x \bullet 0 = 0$
[3] $x + 1 = 1$	**[4]** $x \bullet 1 = x$
[5] $x + x = x$	**[6]** $x \bullet x = x$
[7] $x + x' = 1$	**[8]** $x \bullet X' = 0$
[9] $x + y = y + x$	**[10]** $xy = yx$
[11] $x + (y + z) = (x + y) + z$	**[12]** $x(yz) = (xy)z$
[13] $x(y + z) = xy + xz$	**[14]** $x + yz = (x + y)(x + z)$
[15] $(x + y)' = x'y'$	**[16]** $(xy)' = x' + y'$
[17] $(x')' = x$	

[15] and [16] : De Morgan's Theorem

[ex.1]

F= AB' + C'D + AB' + C'D
= x + x (let x= AB' + C'D)
=x
= AB' + C'D

[ex.2]

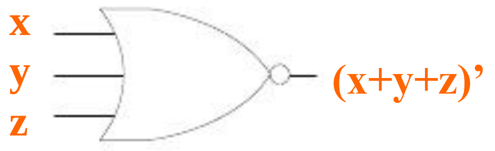
F= ABC + ABC' + A'C
= AB(C + C') + A'C
= AB + A'C

Fig. 1-6(a)

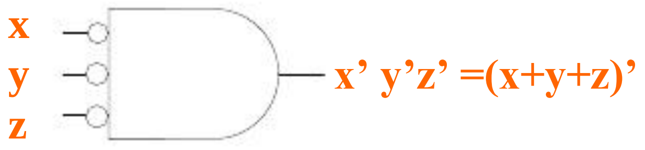
Fig. 1-6(b)

1 inverter, 1 AND gate

Fig. 1-4 2 graphic symbols for NOR gate

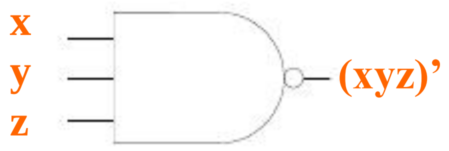


(a) OR-invert

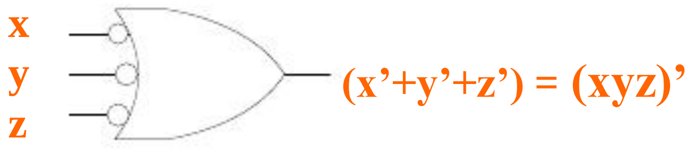


(b) invert-AND

Fig. 1-5 2 graphic symbols for NAND gate



(a) AND-invert



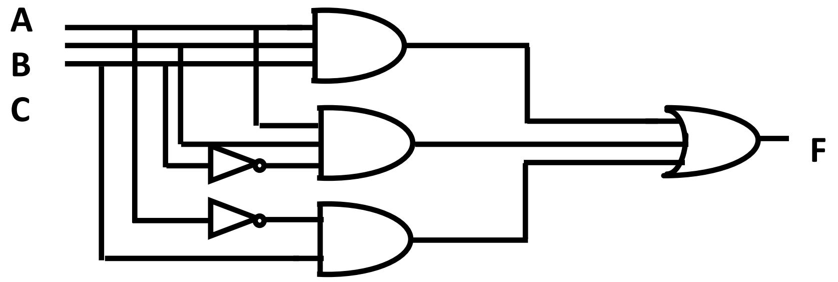
(b) invert-OR

EQUIVALENT CIRCUITS

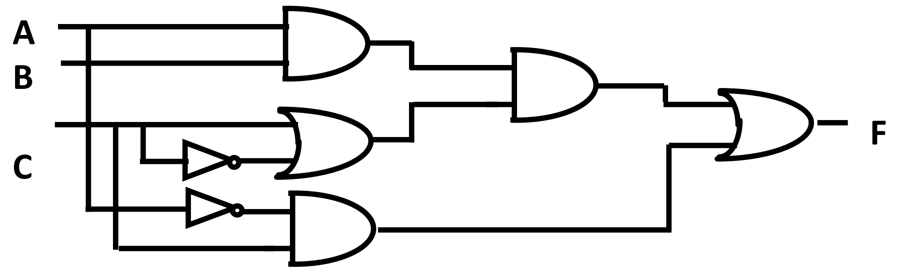
Many different logic diagrams are possible for a given Function

F = ABC + ABC' + A'C .... (1)
= AB(C + C') + A'C [13] ...... (2)
= AB • 1 + A'C [7]
= AB + A'C [4] .... (3)

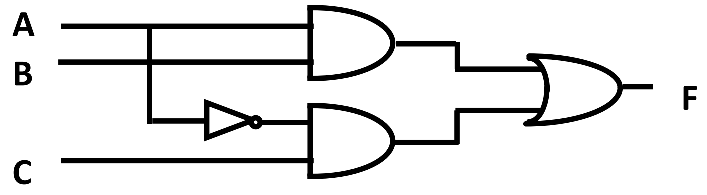
(1)



(2)



(3)



COMPLEMENT OF FUNCTIONS

A Boolean function of a digital logic circuit is represented by only using logical variables and AND, OR, and Invert operators.

--> Complement of a Boolean function

- Replace all the variables and subexpressions in the parentheses appearing in the function expression with their respective complements

$$A,B,\ldots,Z,a,b,\ldots,z \Rightarrow A',B',\ldots,Z',a',b',\ldots,z'$$

$$(p + q) \Rightarrow (p + q)'$$

- Replace all the operators with their respective complementary operators

AND $\Rightarrow$ OR

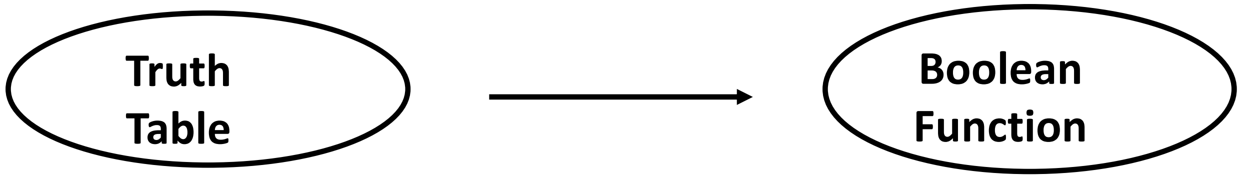
OR $\Rightarrow$ AND

- Basically, extensive applications of the DeMorgan's theorem

$$(x_1 + x_2 + \ldots + x_n)' \Rightarrow x_1'x_2'\ldots x_n'$$

$$(x_1x_2 \ldots x_n)' \Rightarrow x_1' + x_2' + \ldots + x_n'$$

SIMPLIFICATION



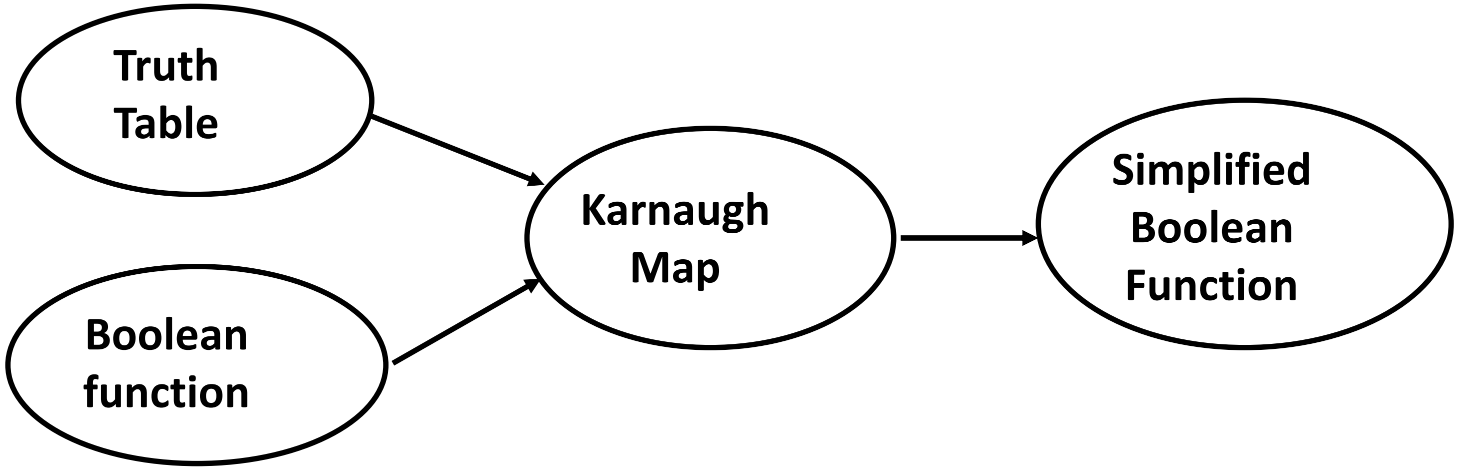
Unique

Many different expressions exist

Simplification from Boolean function

- Finding an equivalent expression that is least expensive to implement
- For a simple function, it is possible to obtain
  a simple expression for low cost implementation
- But, with complex functions, it is a very difficult task

Karnaugh Map(K-map) is a simple procedure for simplifying Boolean expressions.



Map Simplification

Karnaugh Map(K-Map)

- Map method for simplifying Boolean expressions

Minterm / Maxterm

- Minterm : n variables ***product*** (x=1, x'=0)
- Maxterm : n variables ***sum*** (x=0, x'=1)

2 variables example

x	y	Minterm		Maxterm	
0	0	$x'y'$	m_0	$x + y$	M_0
0	1	$x'y$	m_1	$x + y'$	M_1
1	0	$x\ y'$	m_2	$x' + y$	M_2
1	1	$x\ y$	m_3	$x' + y'$	M_3

$m_0 + m_1 + m_2 + m_3$

M_0 □ M_1 □ M_2 □ M_3

$F = \underbrace{x'y}_{m_1} + \underbrace{xy}_{m_3}$

= ©(1,3) **(*m_1 + m_3*)**

= ∠(0,2) **(Complement = *M_0* □ *M_2*)**

KARNAUGH MAP

Karnaugh Map for an n-input digital logic circuit (n-variable sum-of-products form of Boolean Function, or Truth Table) is

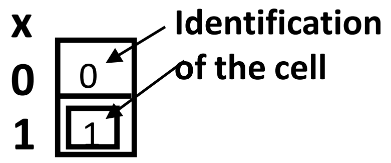
- **Rectangle divided into 2^n cells**
- **Each cell is associated with a *Minterm***
- **An output(function) value for each input value associated with a mintern is written in the cell representing the minterm**

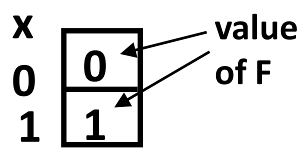
--> 1-cell, 0-cell

Each Minterm is identified by a decimal number whose binary representation is identical to the binary interpretation of the input values of the minterm.

Karnaugh Map

x	F
0	1
1	0

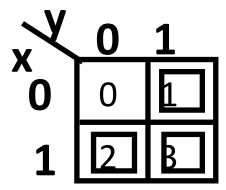


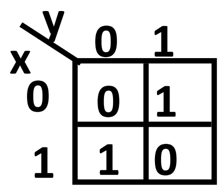


$F(x) = \sum(1)$

1-cell

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

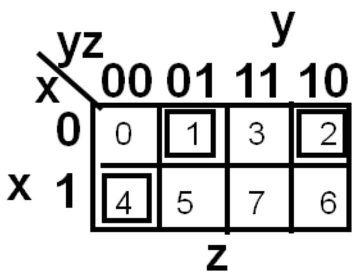




$F(x,y) = \sum(1,2)$

KARNAUGH MAP

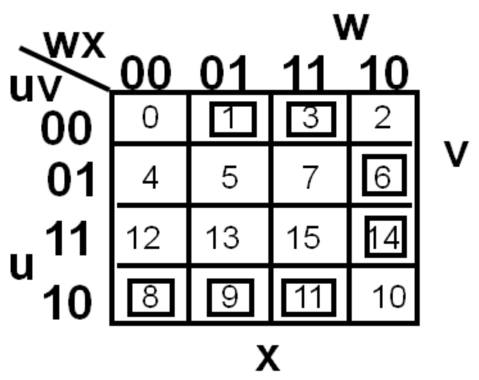
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0



x \ yz	00	01	11	10
0	0	1	0	1
1	1	0	0	0

$F(x,y,z) = \sum (1,2,4)$

u	v	w	x	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0



uv \ wx	00	01	11	10
00	0	1	1	0
01	0	0	0	1
11	0	0	0	1
10	1	1	1	0

$F(u,v,w,x) = \sum (1,3,6,8,9,11,14)$

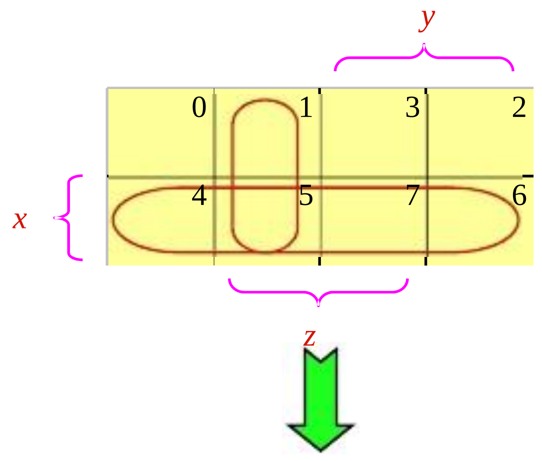
Ex) F= x + y’z

(1) Truth Table

x	y	z	F	Minterm
0	0	0	0	m_0
0	0	1	1	m_1
0	1	0	0	m_2
0	1	1	0	m_3
1	0	0	1	m_4
1	0	1	1	m_5
1	1	0	1	m_6
1	1	1	1	m_7

(2) *F*(*x*, *y*,*z*) = ©(1,4,5,6,7)

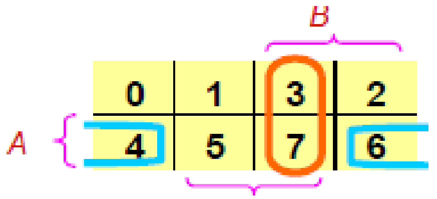
(3)



F= x + y’z

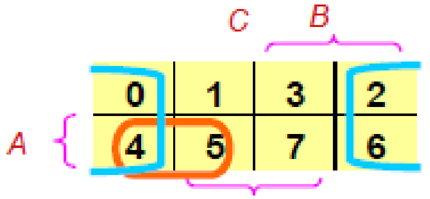
[ex.] *F*(*A*,*B*,*C*) = ©(3,4,6,7)

F=AC’ + BC



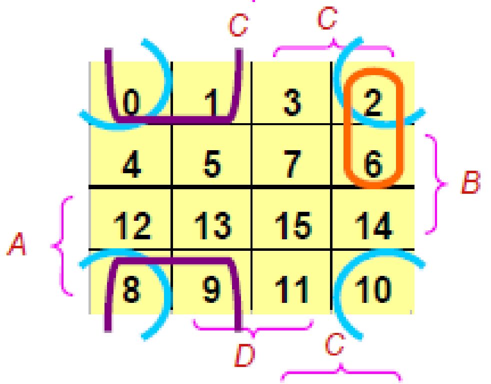
[ex.] *F*(*A*,*B*,*C*) = ©(0,2,4,5,6)

F=C’ + AB’



[ex.] *F*(*A*,*B*,*C*,*D*) = ©(0,1,2,6,8,9,10)

F=B’D’ + B’C’ + A’CD’



Product-of-Sums Simplification

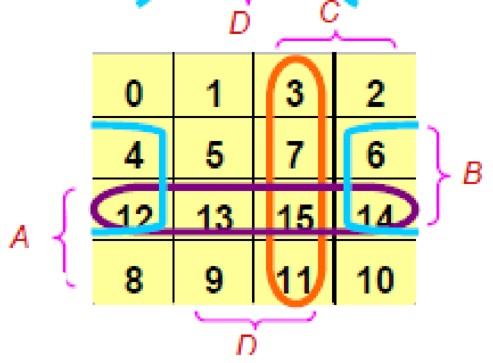
F(*A*,*B*,*C*,*D*) = ©(0,1,2,5,8,9,10)

F=B’D’ + B’C’ + A’C’D *Sum of product*

F’=AB + CD + BD’(square marked 0’s)

F’’(F)=(A’ + B’)(C’ + D’)(B’ + D) 전개

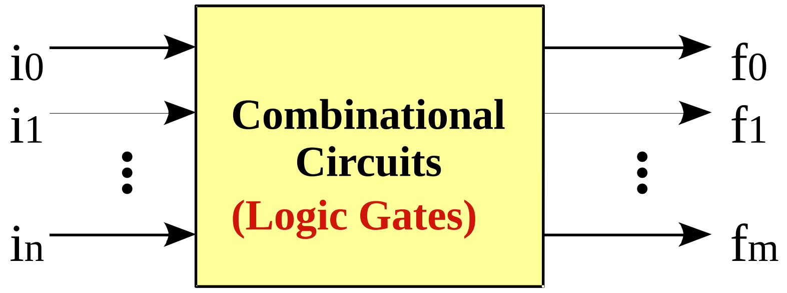
Product of Sum



Combinational Circuits

A connected arrangement of *logic gates* with a set of inputs and outputs

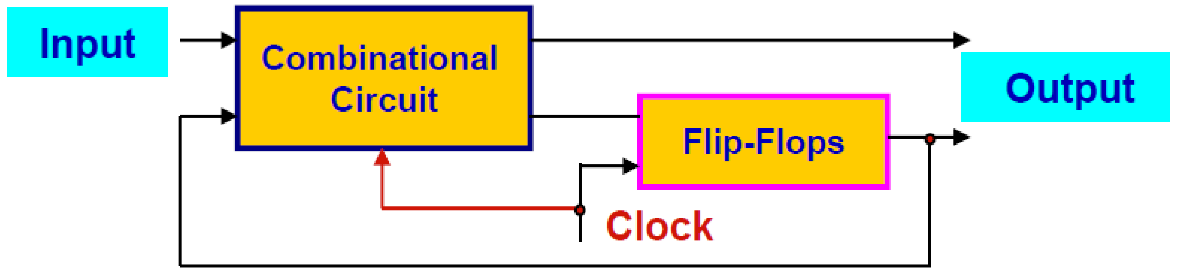
Fig. 1-15 Block diagram of a combinational circuit



A sequential circuit

is an interconnection of F/F and Gate

Clocked synchronous sequential circuit



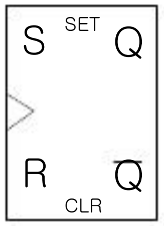
Flip-Flops

Combinational Circuit = Gate
Sequential Circuit = Gate + F/F

Flip-Flop

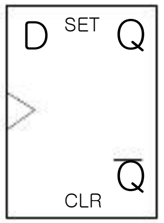
The *storage elements* employed in clocked *sequential circuit*
A binary cell capable of storing one bit of information

SR*(Set/Reset)* F/F



S	R	Q(t+1)	
0	0	Q(t)	no change
0	1	0	clear to 0
1	0	1	set to 1
1	1	?	Indeterminate

D*(Data)* F/F

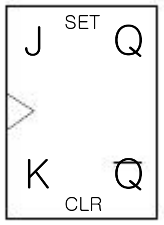


D	Q(t+1)	
0	0	clear to 0
1	1	set to 1

"no change" condition : Q(t+1)=D

1) Disable Clock
2) Feedback output into input p.52

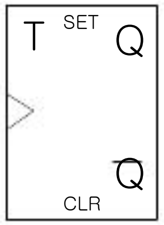
JK*(Jack/King)* F/F



J	K	Q(t+1)	
0	0	Q(t)	no change
0	1	0	clear to 0
1	0	1	set to 1
1	1	Q(t)'	Complement

JK F/F is a refinement of the SR F/F
The indeterminate condition of the SR type is defined in complement

T*(Toggle)* F/F



T	Q(t+1)	
0	Q(t)	no change
1	Q'(t)	Complement

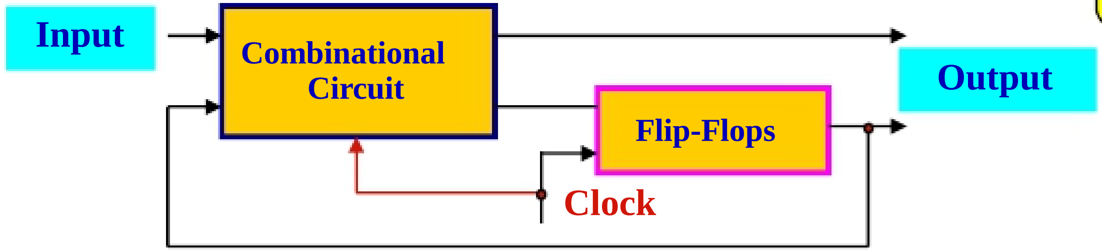
T=1(J=K=1), T=0(J=K=0) 이면 JK F/F
수식 표현 : Q(t+1)= Q(t) □ T xor

1-7 Sequential Circuits

A sequential circuit is an interconnection of F/F and Gate

Clocked synchronous sequential circuit

Combinational Circuit = Gate
Sequential Circuit = Gate + F/F



Input Equation

$D_A = Ax + Bx$, $D_B = A'x$

Output Equation

$y = Ax' + Bx'$

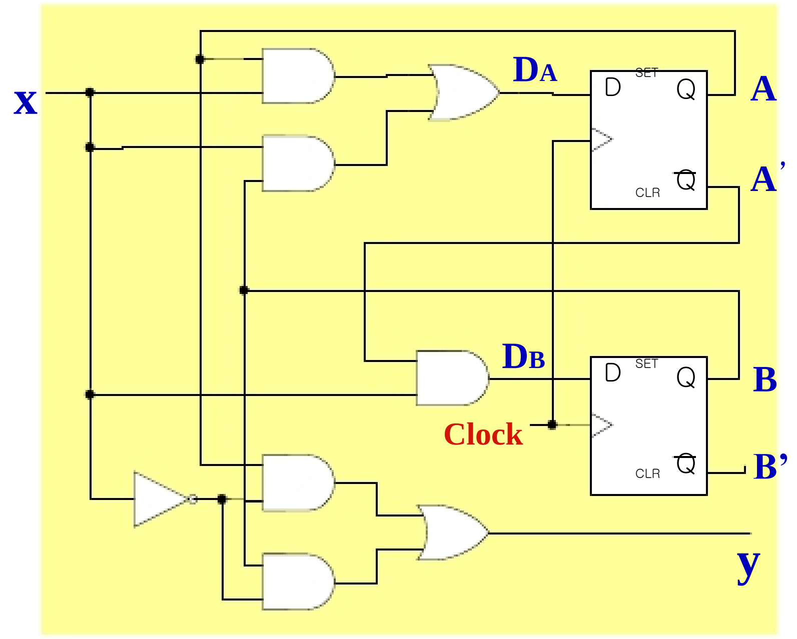


Fig. 1-25 Example of a sequential circuit