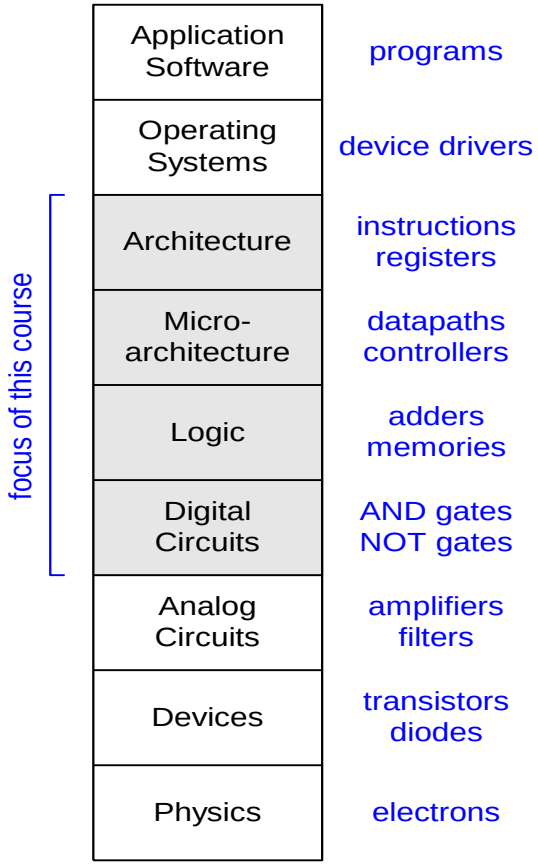
Introduction to Digital Logic Design

Subhasis Bhattacharjee

Scope

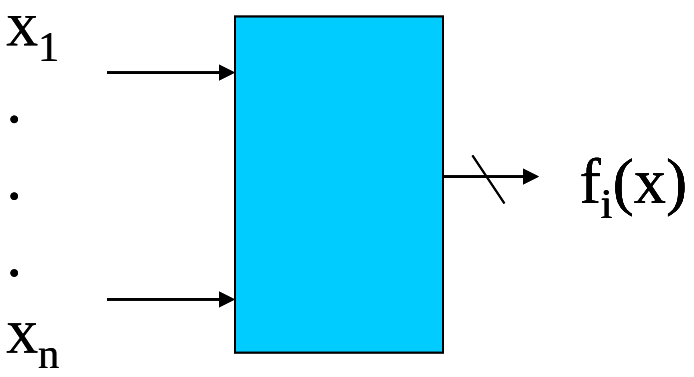

Application Software
programs
Operating Systems
device drivers
Architecture
instructions
registers
Micro-architecture
datapaths
controllers
Logic
adders
memories
Digital Circuits
AND gates
NOT gates
Analog Circuits
amplifiers
filters
Devices
transistors
diodes
Physics
electrons
focus of this course

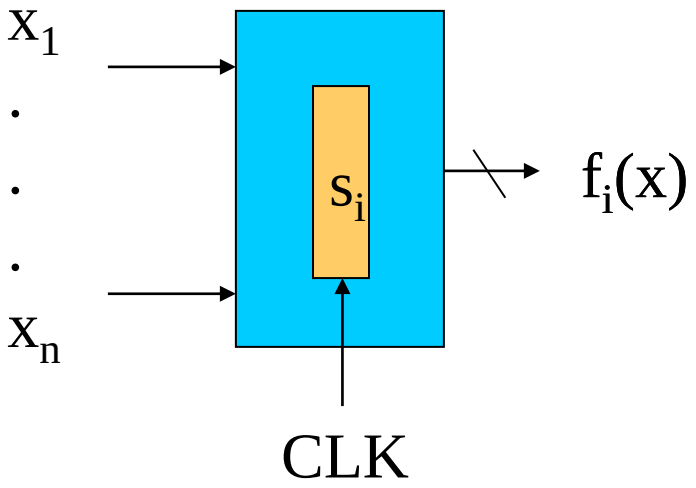
Major Parts in this course

We will cover the following major things in this course:

- Combinational Logic
- Sequential Networks
- Standard Modules

Combinational Logic vs Sequential Network





Combinational logic:

$y_i = f_i(x_1,..,x_n)$

Sequential Networks

1) Memory 2) Time Steps (Clock)

$y_i^t = f_i(x_1^t,\ldots,x_n^t, s_1^t, \ldots,s_m^t)$

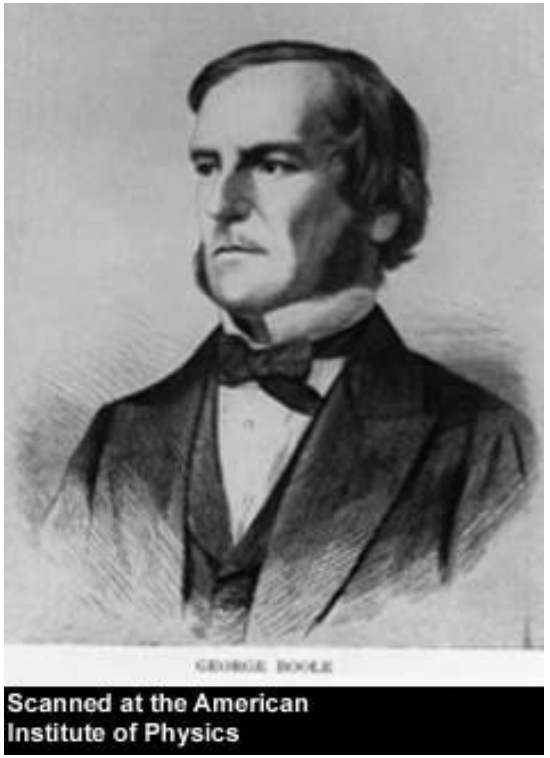
$S_i^{t+1} = g_i(x_1^t,\ldots,x_n^t, s_1^t,\ldots,s_m^t)$

Scope

Subjects	**Building Blocks**	**Theory**
Combinational Logic	AND, OR, NOT,XOR	Boolean Algebra
Sequential Network	FF, Counter, Registers	Finite State Machine
Standard Modules	Operators, Interconnects, Memory	Arithmetics, Universal Logic

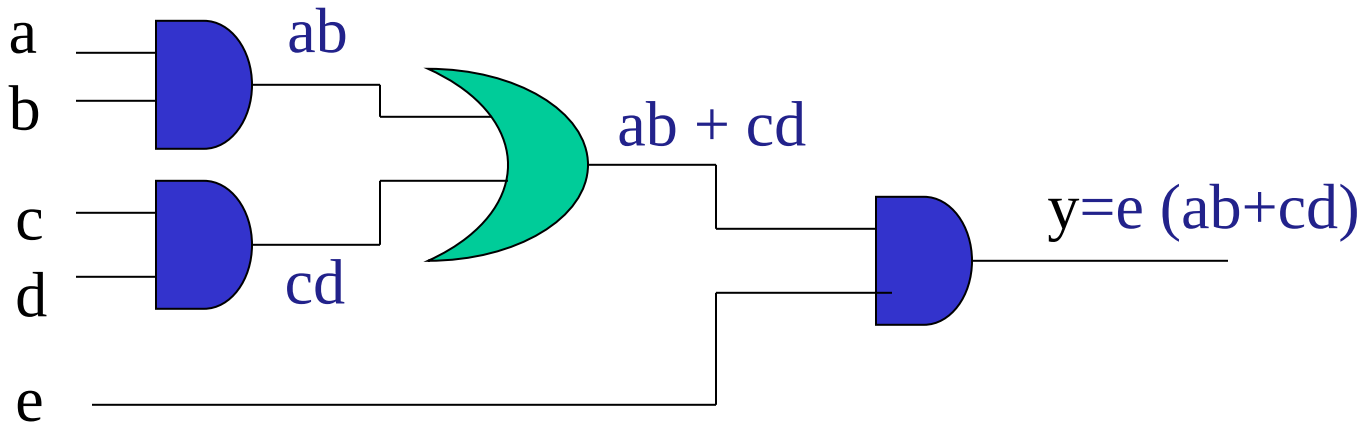
George Boole, 1815 - 1864

- Born to working class parents
- Taught himself mathematics and joined the faculty of Queen's College in Ireland.
- Wrote *An Investigation of the Laws of Thought* (1854)
- Introduced binary variables
- Introduced the three fundamental logic operations: AND, OR, and NOT.



Combinational Logic

Combinational Logic vs Boolean Algebra Expression



Schematic Diagram:

5 primary inputs

4 components

9 signal nets

12 pins

Boolean Algebra:

5 literals

4 operators

Some Definitions

- Complement: variable with a bar over it
  $\bar{A}$, $\bar{B}$, $\bar{C}$
- Literal: variable or its complement
  A, $\bar{A}$, B, $\bar{B}$, C, $\bar{C}$
- Implicant: product of literals
  $A\bar{B}C$, $A\bar{C}$, BC
- Minterm: product that includes all input variables
  $\bar{A}BC$, $AB\bar{C}$, $A\bar{B}C$
- Maxterm: sum that includes all input variables
  $(A+\bar{B}+C)$, $(\bar{A}+B+C)$, $(\bar{A}+\bar{B}+\bar{C})$

Digital Discipline: Binary Values

- Typically consider only two discrete values:
  - 1's and 0's
  - 1, TRUE, HIGH
  - 0, FALSE, LOW
- 1 and 0 can be represented by specific voltage levels, rotating gears, fluid levels, etc.
- Digital circuits usually depend on specific voltage levels to represent 1 and 0
- *Bit*: *B*inary dig*it*

Digital (logic) Elements: Gates

- Digital devices or gates have one or more inputs and produce an output that is a function of the current input value(s).
- All inputs and outputs are binary and can only take the values 0 or 1
- A gate is called a *combinational circuit* because the output only depends on the current input combination.
- Digital circuits are created by using a number of connected gates such as the output of a gate is connected to the input of one or more gates in such a way to achieve specific outputs for input values.
- Digital or logic design is concerned with the design of such circuits.

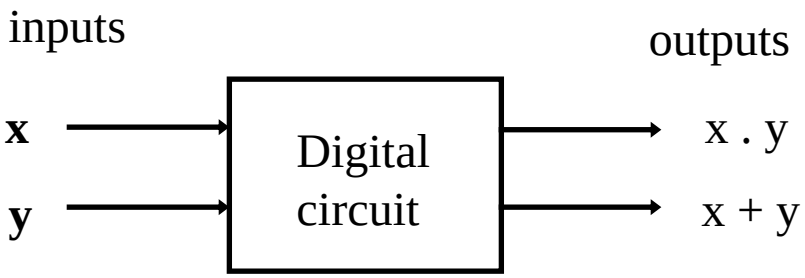
Truth Tables

Provide a listing of every possible combination of values of binary inputs to a digital circuit and the corresponding outputs.

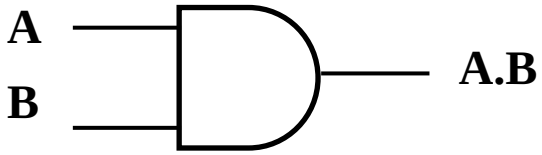
INPUTS	OUTPUTS
…	…
…	…

Truth Table			
Inputs		Outputs	
x	y	x.y	x+y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

- Example (2 inputs, 2 outputs):



AND Gate

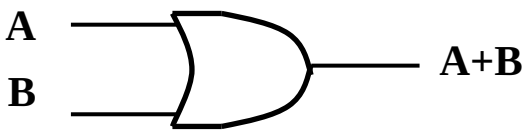


Symbol

Truth Table		
Inputs		Output
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

SHARED TERMINALS (We will not mention them - Usually)	
GND	Connected to ground
VCC	Connected to positive voltage to provide power to all four gates

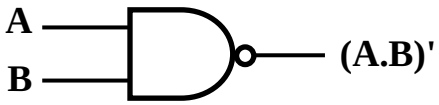
OR Gate



Symbol

Truth Table		
Inputs		Output
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

NAND Gate



Symbol

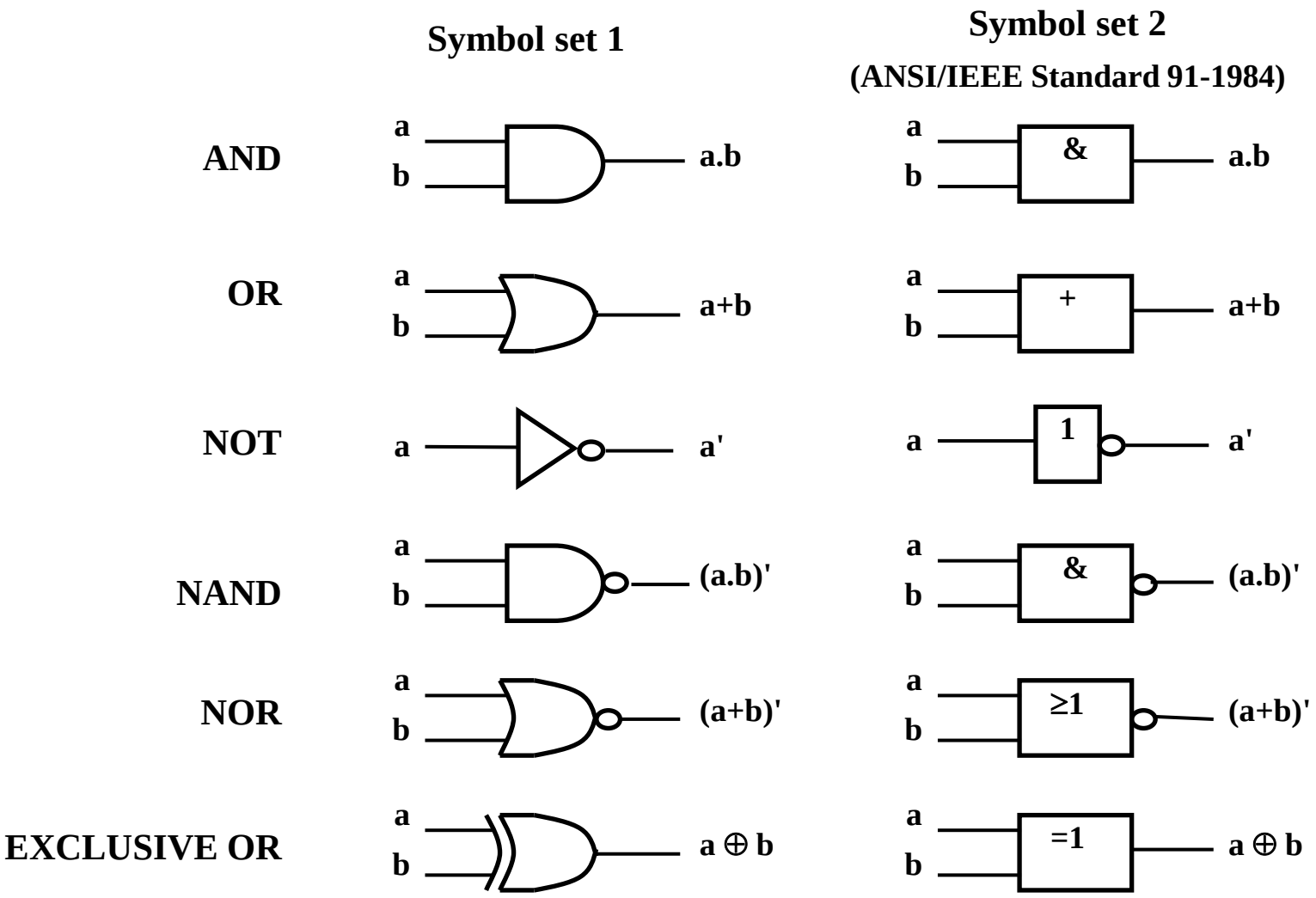
Truth Table		
Inputs		Output
A	B	(A . B)’
0	0	1
0	1	1
1	0	1
1	1	0

The NAND Gate

- Multiple equivalent symbols / logical representations
- NAND gate is self-sufficient (can build any logic circuit with it).
- Can be used to implement AND/OR/NOT

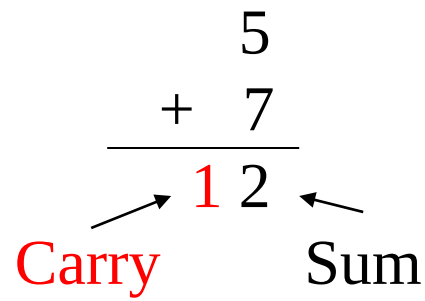
Equivalent Symbols

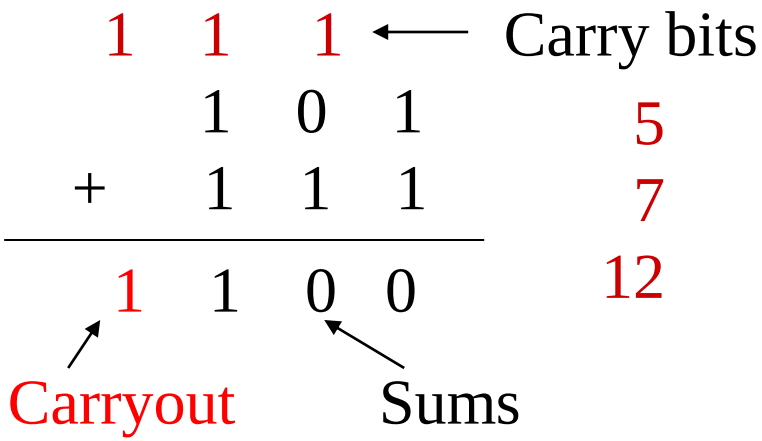
Logic Gates


Symbol set 1
Symbol set 2
(ANSI/IEEE Standard 91-1984)
AND
a
b
a.b
&
OR
a+b
+
NOT
a'
1
NAND
(a.b)'
NOR
(a+b)'
≥1
EXCLUSIVE OR
a ⊕ b
=1

Useful Circuits using Logic Gates

Binary Addition



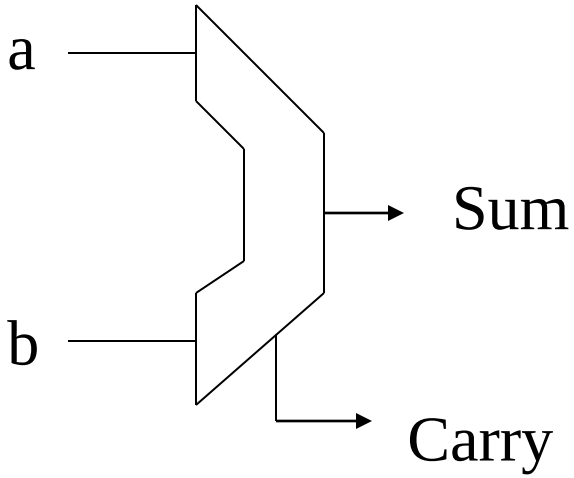


Binary Addition: Hardware

- Half Adder: Two inputs (a,b) and two outputs (carry, sum).

- Full Adder: Three inputs (a,b,c) and two outputs (carry, sum).

Half Adder



Truth Table

a	b	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Switching Function

Switching Expressions:

Sum (a,b) = a'b + ab'

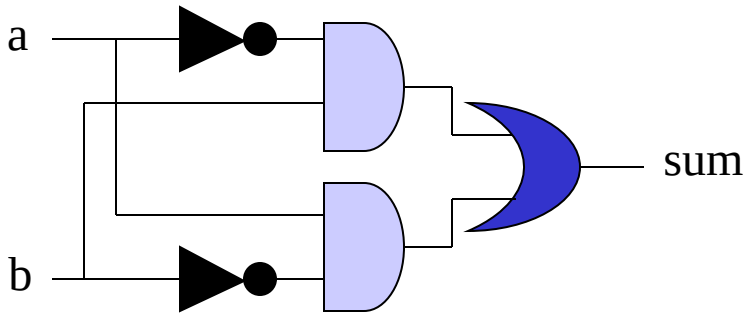
Carry (a, b) = a*b

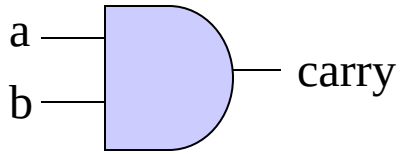
Ex:

Sum (0,0) = 0'0 + 0*0' = 0 + 0 = 0

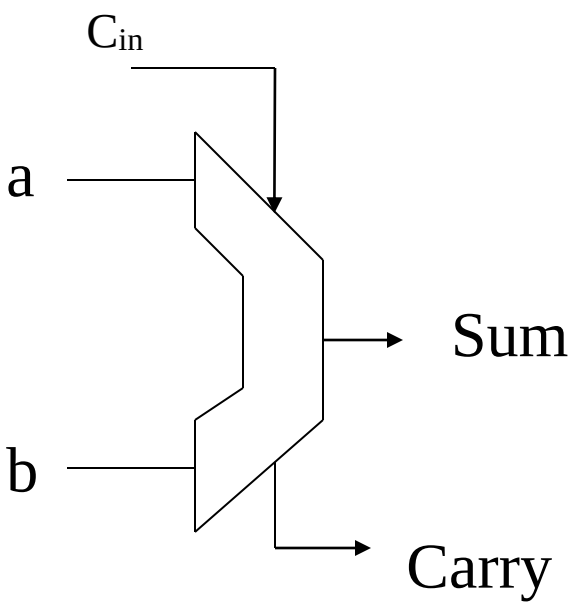
Sum (0,1) = 0'1 + 0*1' = 1 + 0 = 1

Sum (1,1) = 1'1 + 1*1' = 0 + 0 = 0





Full Adder



Truth Table

Id	a	b	c_{in}	carry	sum
0	0	0	0	0	0
1	0	0	1	0	1
2	0	1	0	0	1
3	0	1	1	1	0
4	1	0	0	0	1
5	1	0	1	1	0
6	1	1	0	1	0
7	1	1	1	1	1

Minterm and Maxterm

Id	a	b	c_{in}	carryout		
0	0	0	0	0		a+b+c
1	0	0	1	0		a+b+c’
2	0	1	0	0		a+b’+c
3	0	1	1	1	a’ b c	
4	1	0	0	0		a’+b+c
5	1	0	1	1	a b’c	
6	1	1	0	1	a b c’	
7	1	1	1	1	a b c	

maxterm

minterm

Minterms

$f_1(a,b,c) = a'bc + ab'c + abc' + abc$

$a'bc = 1$ iff $(a,b,c,) = (0,1,1)$
$ab'c = 1$ iff $(a,b,c,) = (1,0,1)$
$abc' = 1$ iff $(a,b,c,) = (1,1,0)$
$abc = 1$ iff $(a,b,c,) = (1,1,1)$

$f_1(a,b,c) = 1$ iff $(a,b,c) = (0,1,1), (1,0,1), (1,1,0),$ or $(1,1,1)$

Ex: $f_1(1,0,1) = 1'01 + 10'1 + 101' + 101 = 1$
$f_1(1,0,0) = 1'00 + 10'0 + 100' + 100 = 0$

Maxterms

$f_2(a,b,c) = (a+b+c)(a+b+c')(a+b'+c)(a'+b+c)$

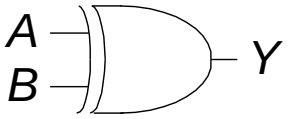
a + b + c = 0 iff (a,b,c,) = (0,0,0)
a + b + c' = 0 iff (a,b,c,) = (0,0,1)
a + b' + c = 0 iff (a,b,c,) = (0,1,0)
a' + b + c = 0 iff (a,b,c,) = (1,0,0)

$f_2(a,b,c) = 0$ iff (a,b,c) = (0,0,0), (0,0,1), (0,1,0), (1,0,0)

Ex: $f_2(1,0,1) = (1+0+1)(1+0+1')(1+0'+1)(1'+0+1) = 1$
$f_2(0,1,0) = (0+1+0)(0+1+0')(0+1'+0)(0'+1+0) = 0$

Other - Two-Input Logic Gates

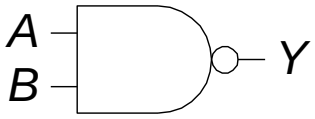
XOR



$Y = A \oplus B$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

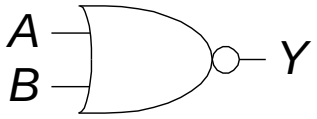
NAND



$Y = \overline{AB}$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

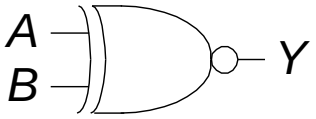
NOR



$Y = \overline{A + B}$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

XNOR



$Y = \overline{A \oplus B}$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Which Gates are Important

Universal Set

Universal Set: A set of gates such that every switching function can be implemented with gates in this set.

Ex:

{AND, OR, NOT}

{AND, NOT}

{OR, NOT}

Universal Set

Universal Set: A set of gates such that every Boolean function can be implemented with gates in this set.

Ex:

{AND, OR, NOT}

{AND, NOT} OR can be implemented with AND & NOT gates a+b = (a'b')'

{OR, NOT} AND can be implemented with OR & NOT gates ab = (a'+b')'

{AND, OR} This is not universal.

Standard Combinational Modules (Combinational Building Blocks)

Some (Common) Building Blocks?

- Decoder: Decode address
- Encoder: Encode address
- Multiplexer (Mux): Select data by address
- Demultiplexier (DeMux): Direct data by address
- Shifter: Shift bit location
- Adder: Add two binary numbers

Part III. Standard Modules

Interconnect Modules:

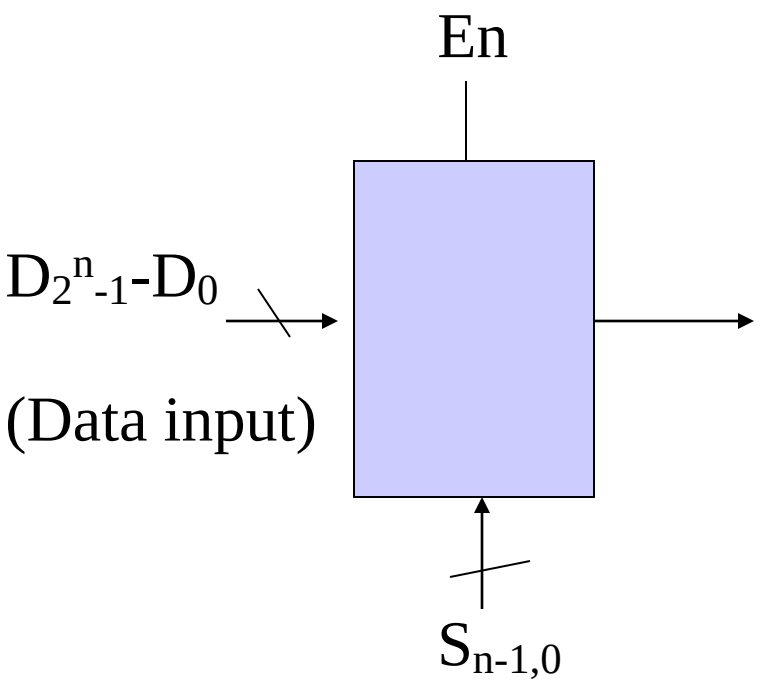
1. Decoder, 2. Encoder

3. Multiplexer, 4. Demultiplexer

Multiplexer

- Definition
- Logic Diagram
- Application

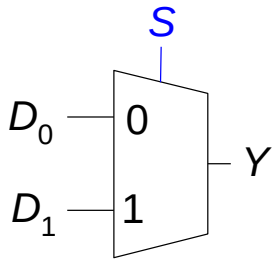
3. Mux (Multiplexer): Definition



Description
If En = 1
  $y = D_i$ where $i = (S_{n-1}, .. , S_0)$
Else
  y = 0

Multiplexer (Mux): Definition

- Selects between one of N inputs to connect to the output.
- $\log_2 N$-bit select input – control input
- Example:

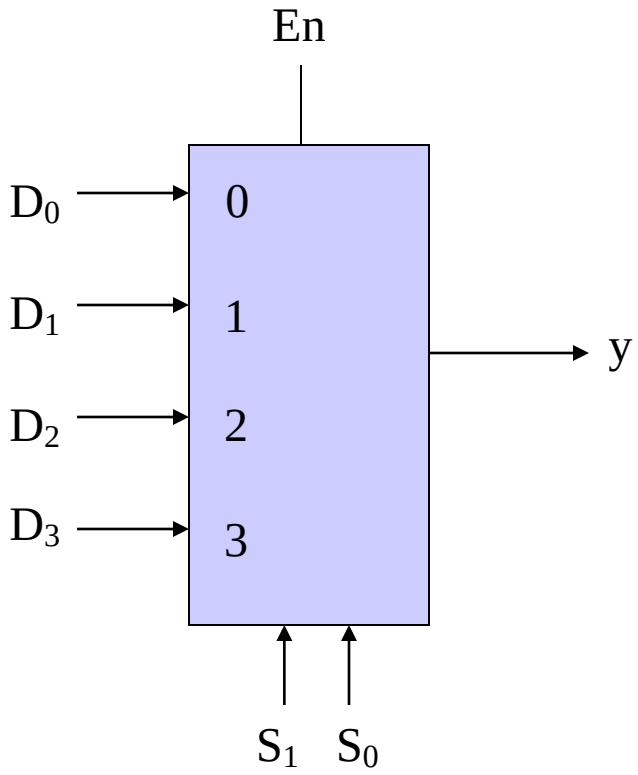


2:1 Mux

S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

S	Y
0	D_0
1	D_1

Multiplexer Definition: Example

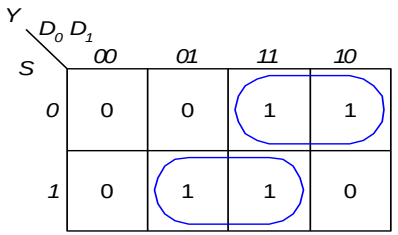


If $D_0 = 0$ and $S_1S_0 = 00$ => $y = 0$

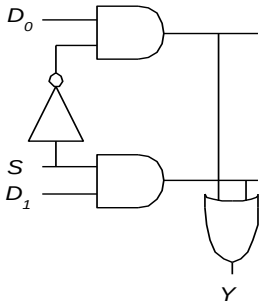
If $D_0 = 1$ and $S_1S_0 = 00$ => $y = 1$

Multiplexer: Logic Diagram

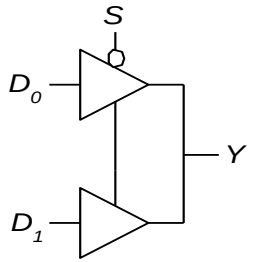
- Logic gates
  - Sum-of-products form



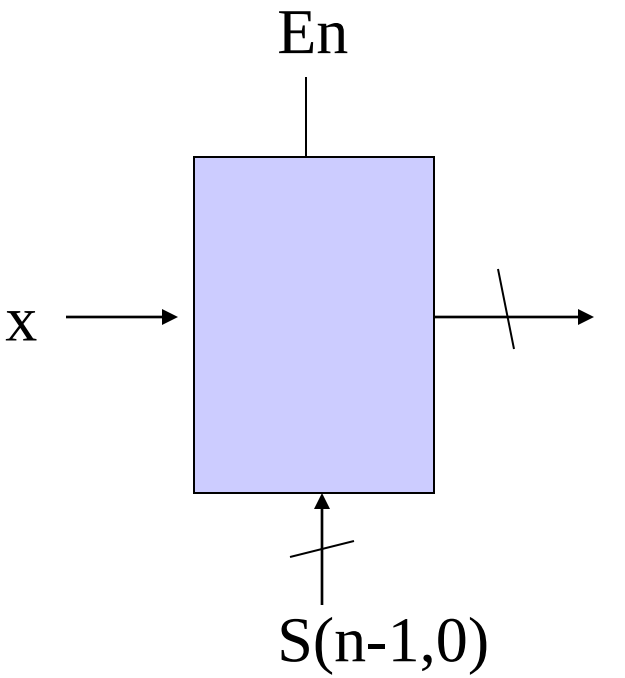
$Y = D_0\overline{S} + D_1S$



- Tristates
  - For an N-input mux, use N tristates
  - Turn on exactly one to select the appropriate input



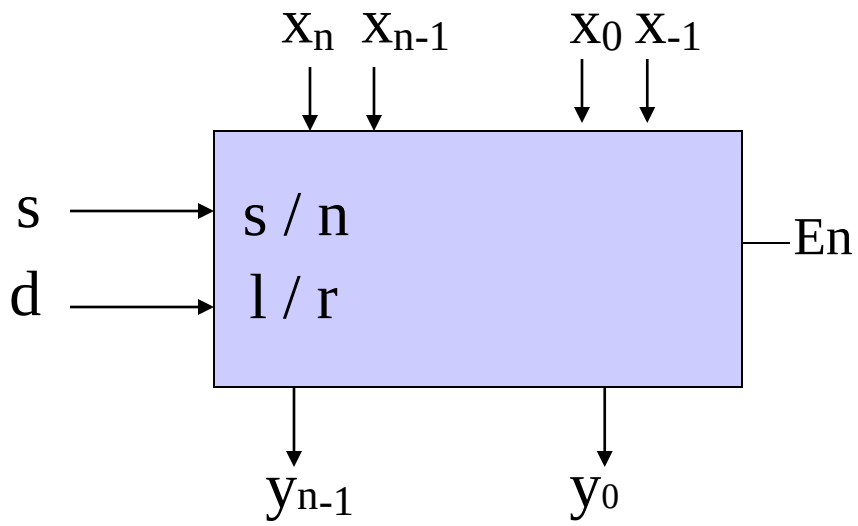
4. Demultiplexers



$y_i = x$ if $i = (S_{n-1}, .. , S_0)$ & En = 1

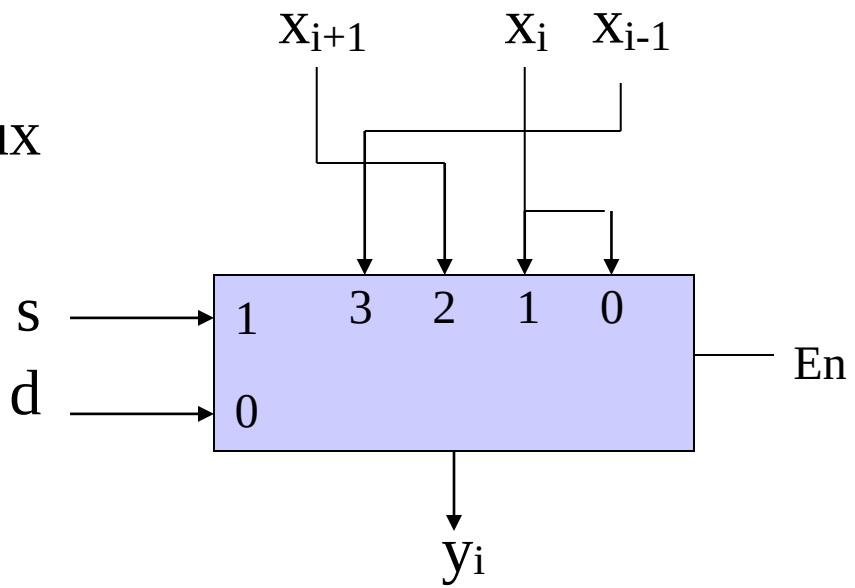
$y_i = 0$ otherwise

Shifter

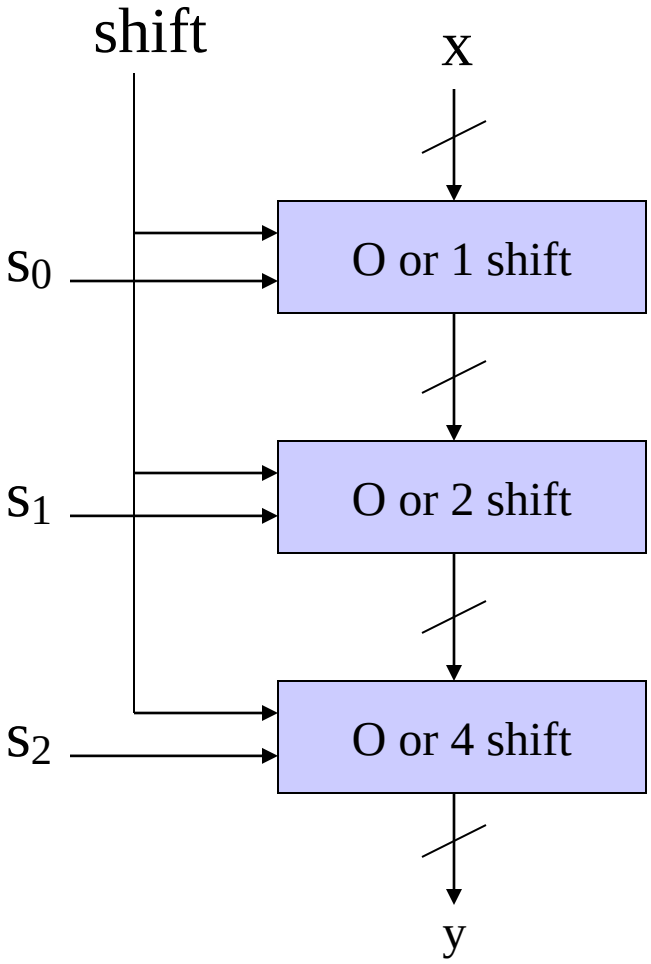


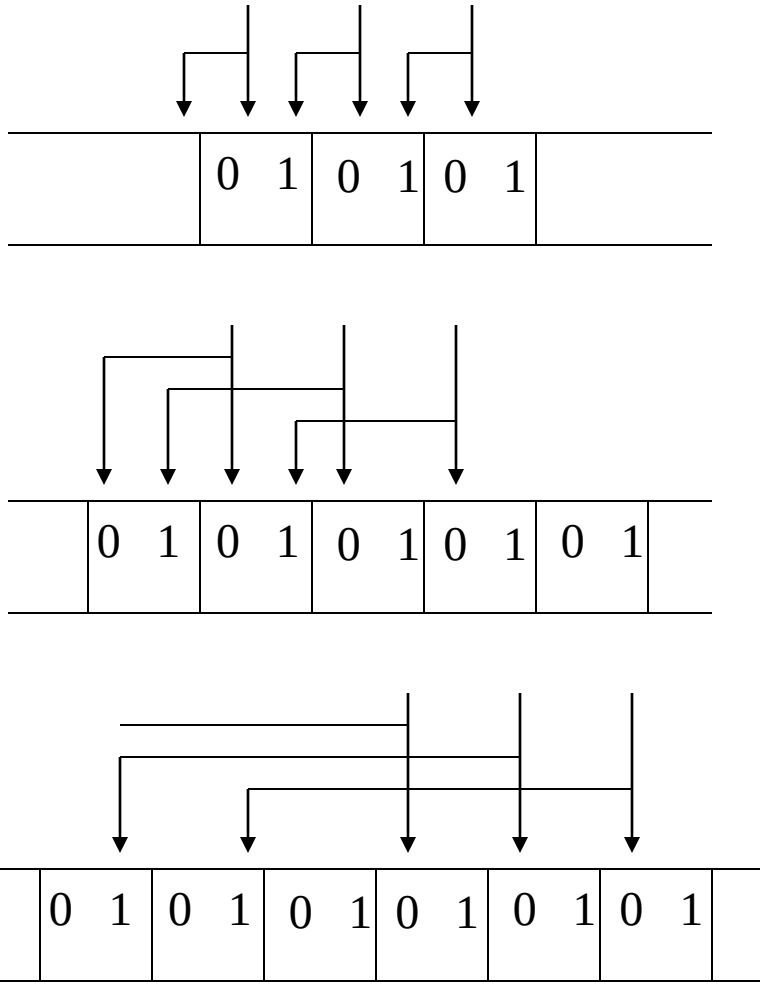
$y_i = x_{i-1}$ if En = 1, s = 1, and d = L
$= x_{i+1}$ if En = 1, s = 1, and d = R
$= x_i$ if En = 1, s = 0
$= 0$ if En = 0

Can be implemented with a mux

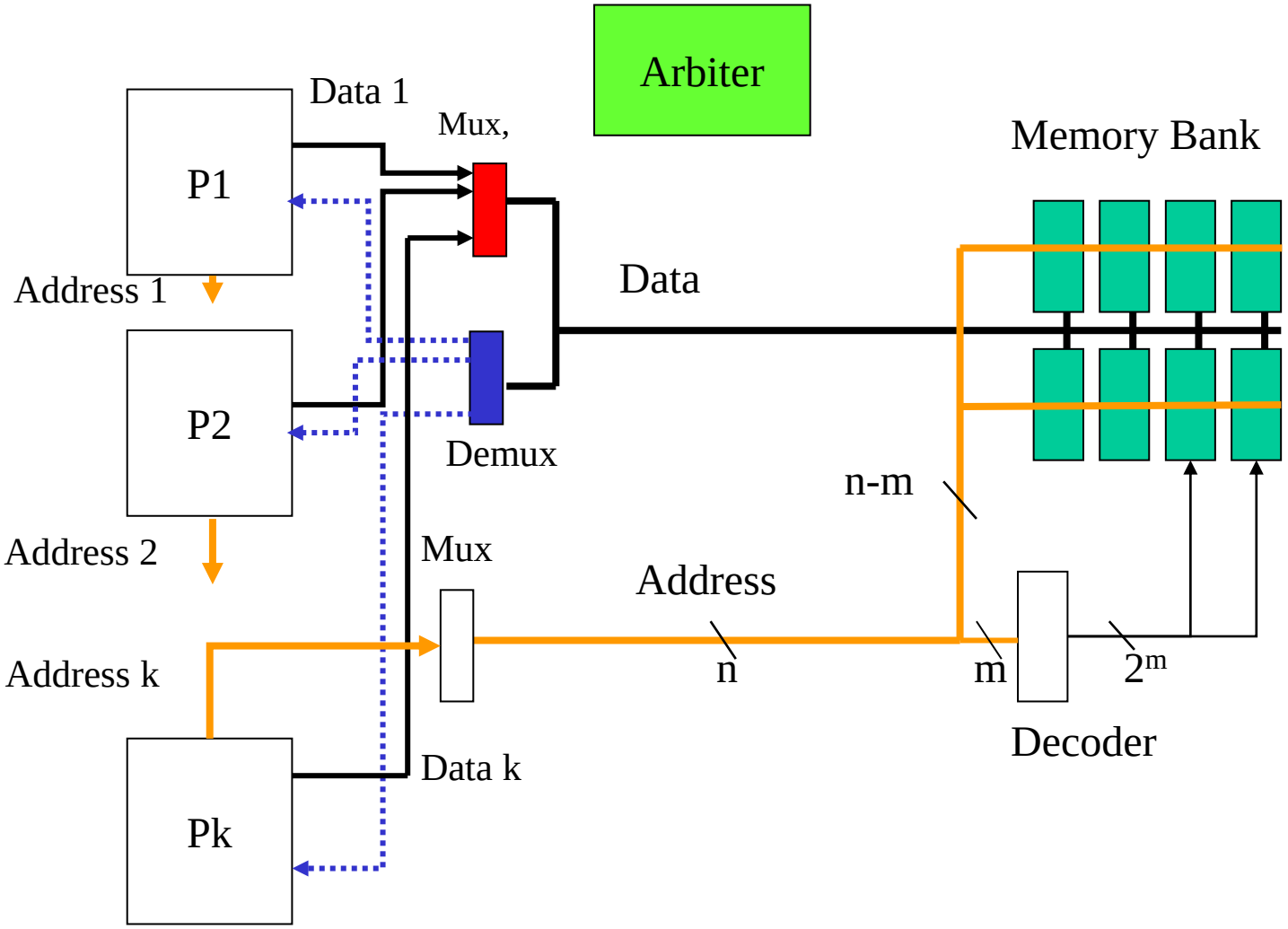


Barrel Shifter





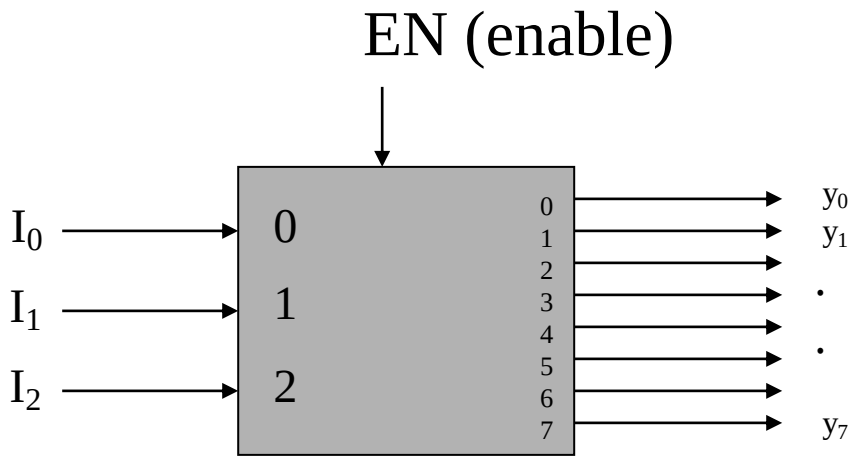
Interconnect: Decoder, Encoder, Mux, DeMux



1. Decoder

- Definition
- Logic Diagram
- Application (Universal Set)
- Tree of Decoders

1. Decoder: Definition



n inputs
n= 3

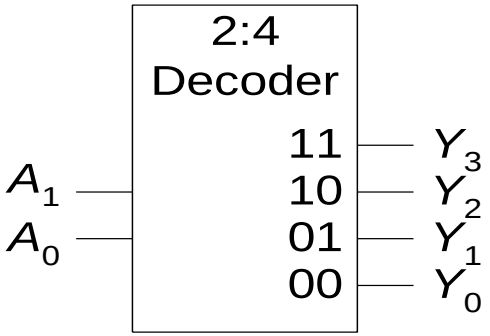
2^n outputs
2^3= 8

n to 2^n decoder function:

y_i = 1 if En= 1 & (I_2, I_1, I_0) = i
y_i= 0 otherwise

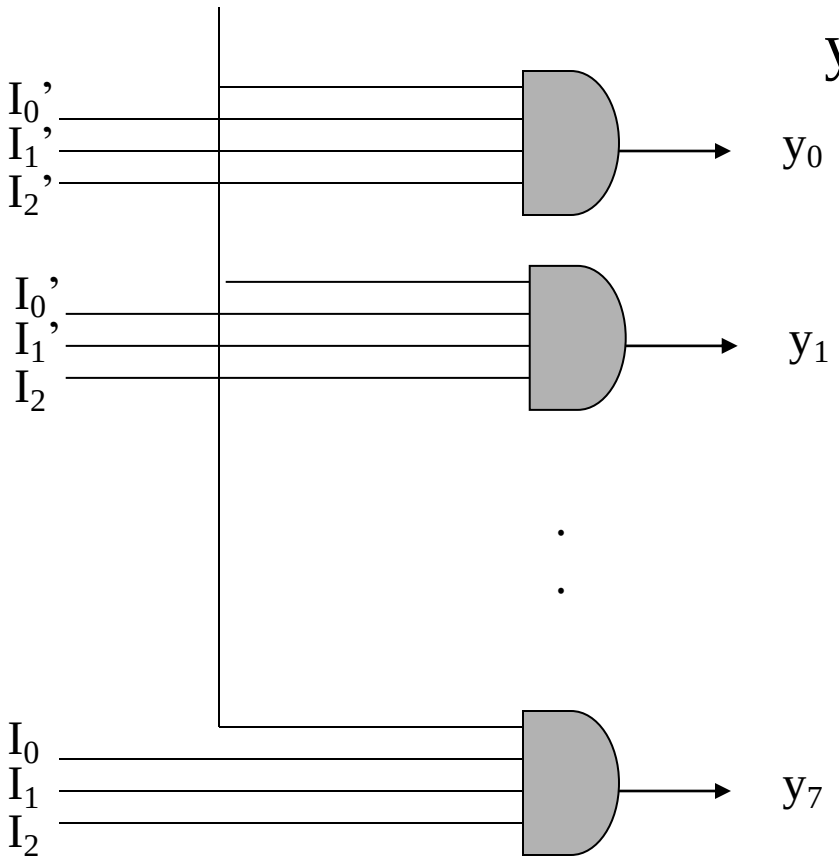
1. Decoder: Definition

- N inputs, 2^N outputs
- One-hot outputs: only one output HIGH at once



A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Decoder: Logic Diagram



$y_i = m_i \, En$

$y_0 = 1$ if $(I_2, I_1, I_0) = (0,0,0)$ & $En = 1$

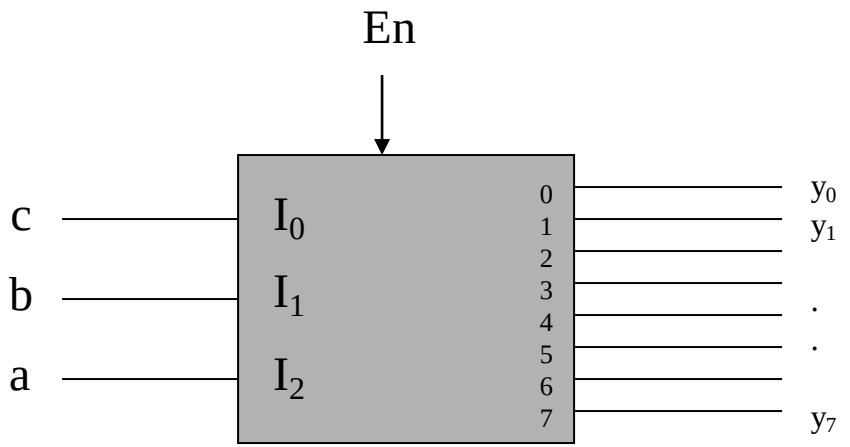
$y_7 = I_2 I_1 I_0 En$

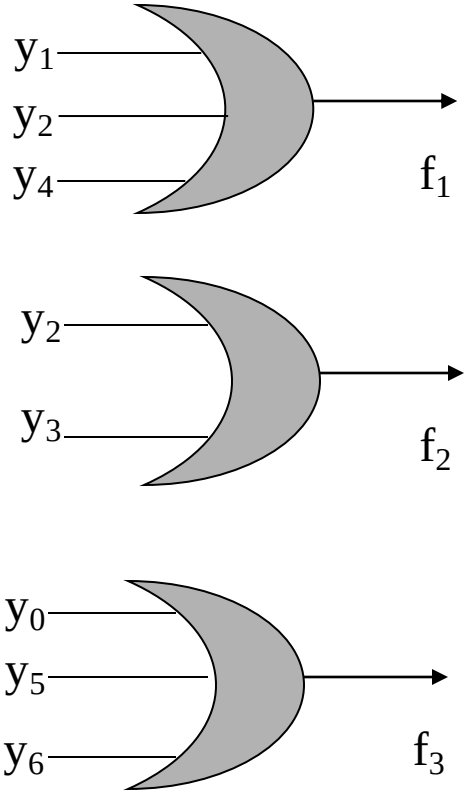
Decoder Application: universal set {Decoder, OR}

Example: Implement functions $f_1(a,b,c) = \Sigma m(1,2,4)$

$f_2(a,b,c) = \Sigma m(2,3)$, and $f_3(a,b,c) = \Sigma m(0,5,6)$

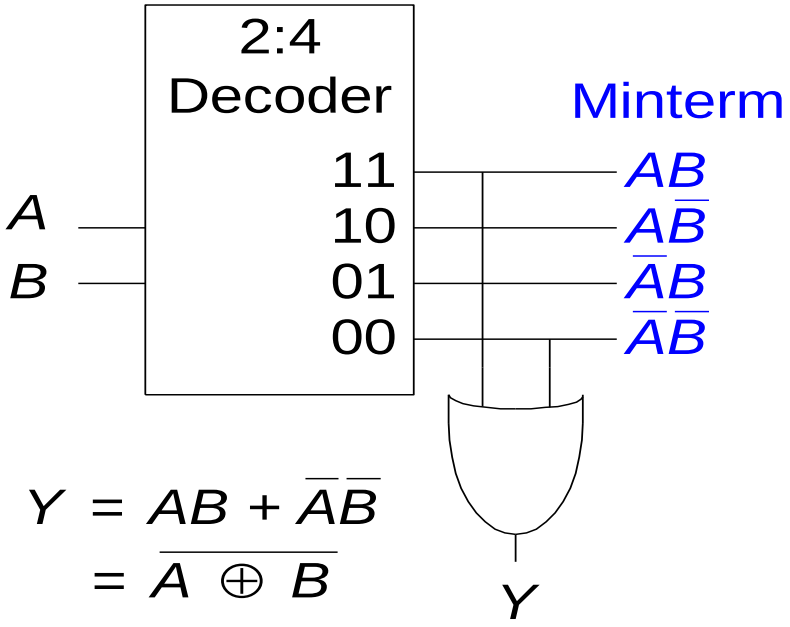
with a 3-input decoder and OR gates.





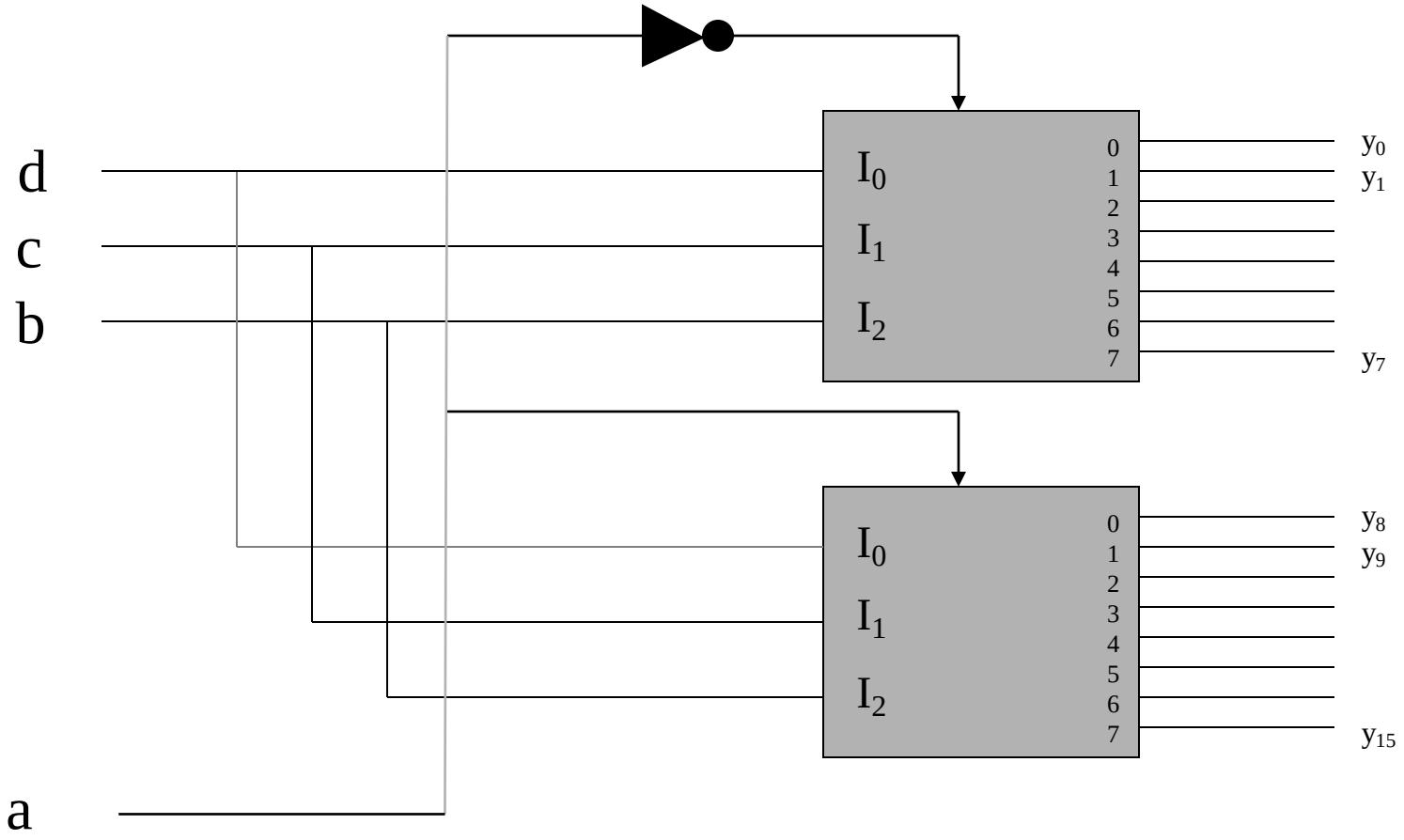
Decoders

- OR minterms



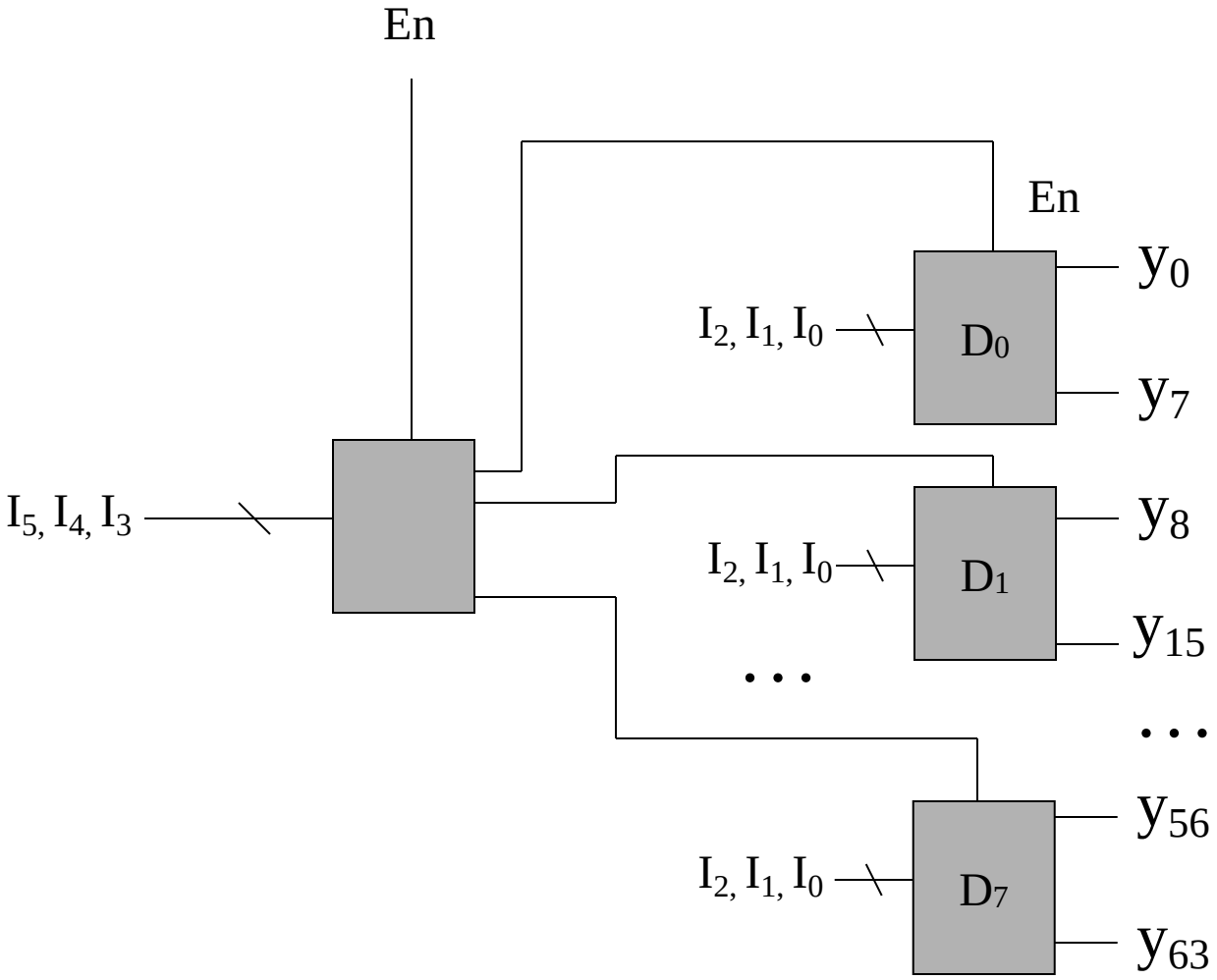
Tree of Decoders

Implement a 4-2^4 decoder with 3-2^3 decoders.



Tree of Decoders

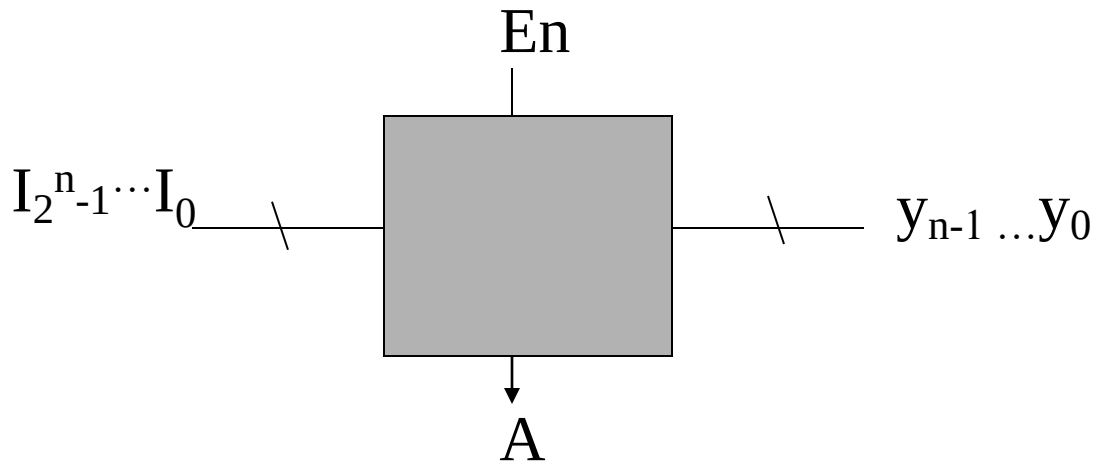
Implement a 6-2^6 decoder with 3-2^3 decoders.

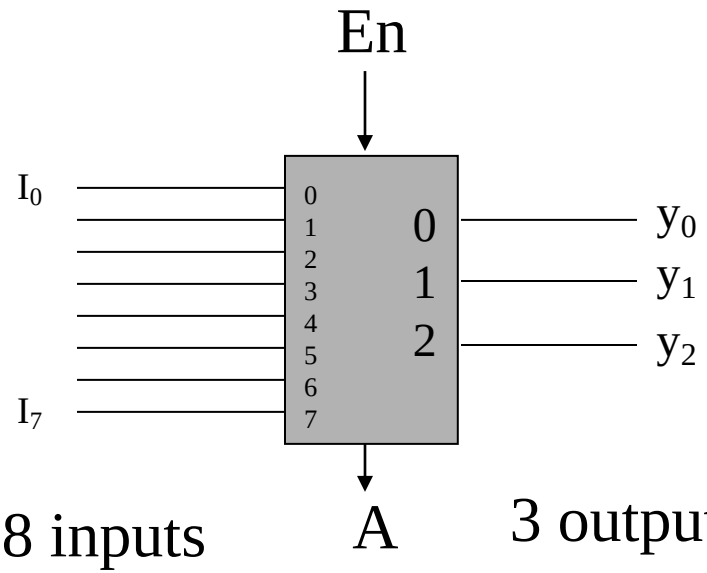


2. Encoder

- Definition
- Logic Diagram
- Priority Encoder

2. Encoder: Definition





Encoder Description:

At most one $I_i = 1$.

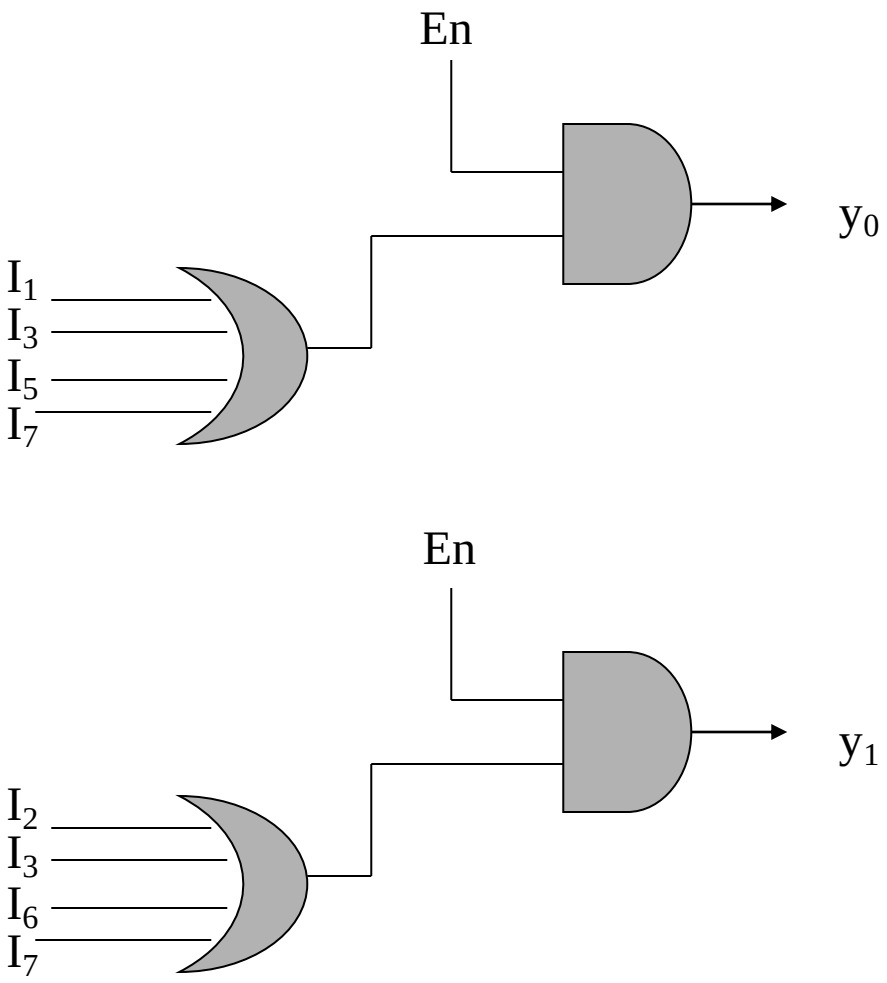
$(y_{n-1},.., y_0) = i$ if $I_i = 1$ & En = 1

$(y_{n-1},.., y_0) = 0$ otherwise.

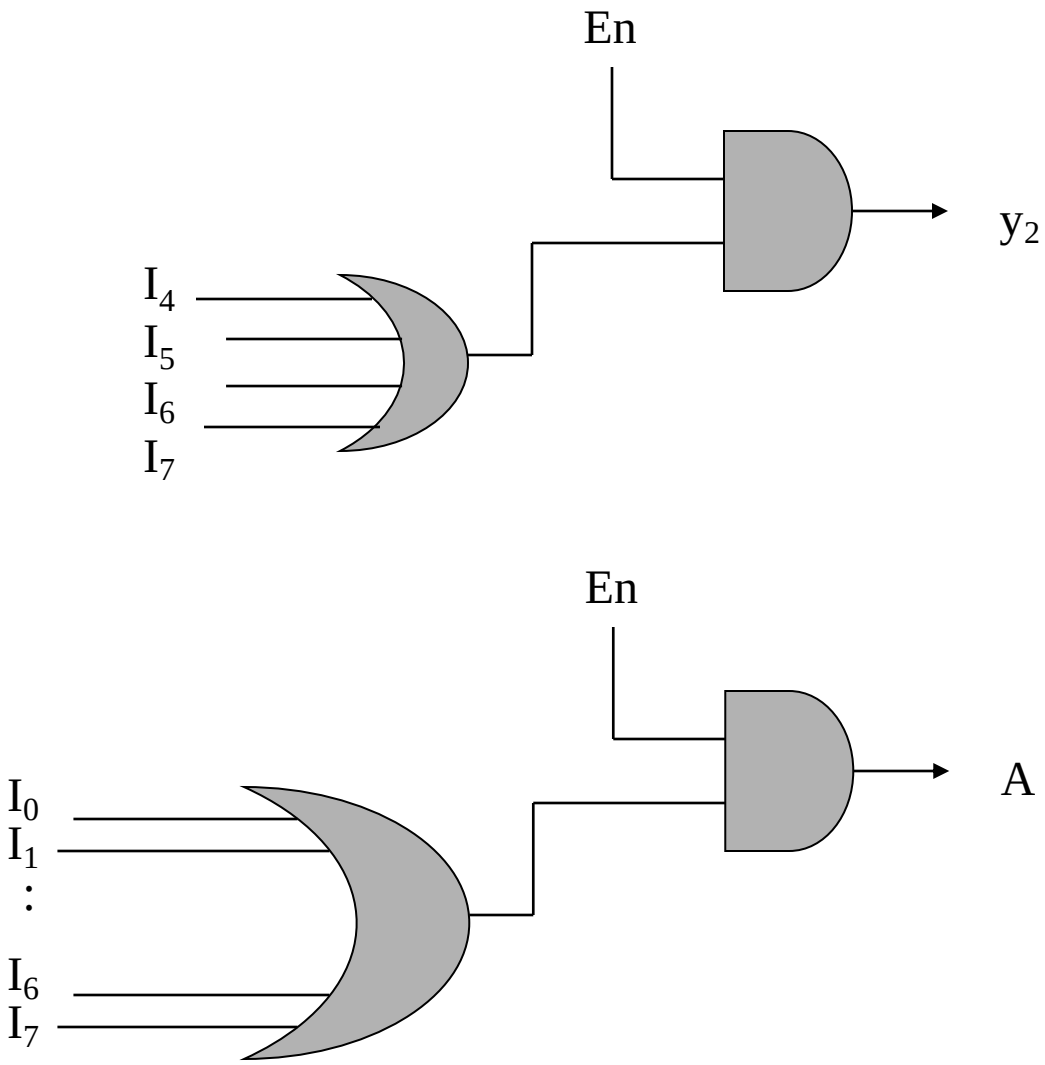
A = 1 if En = 1 and one i s.t. $I_i = 1$

A = 0 otherwise.

Encoder: Logic Diagram



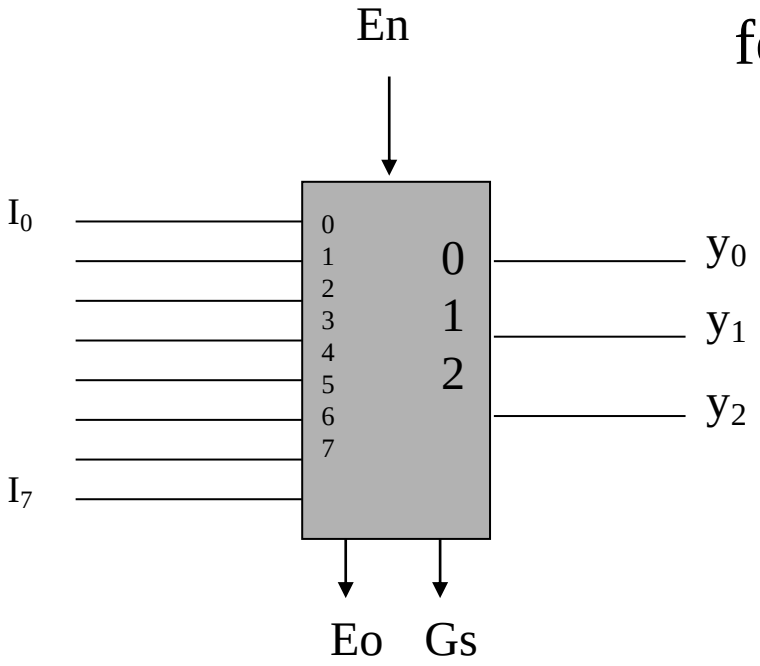
Encoder: Logic Diagram



Priority Encoder: Definition

Description: Input (I_{2^n-1},…, I_0), Output (y_{n-1} ,…,y_0)

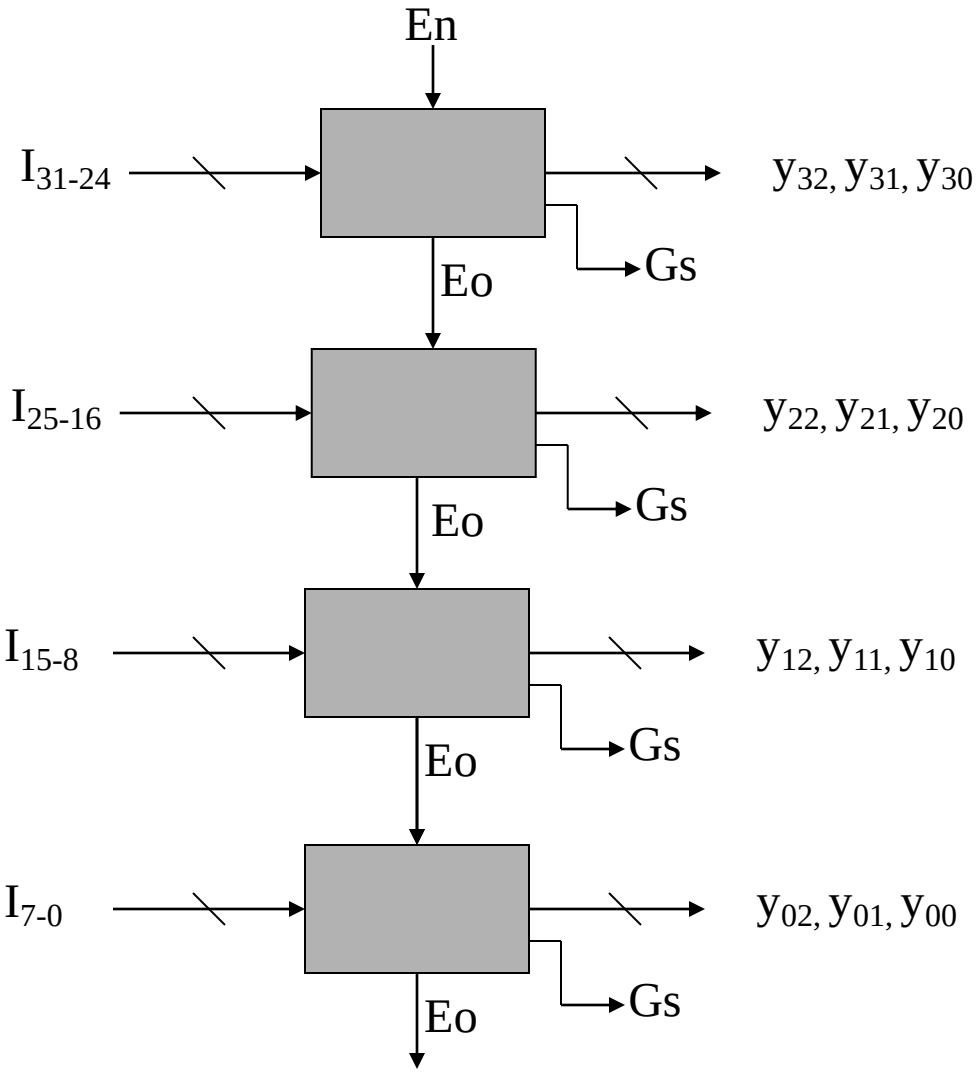
(y_{n-1} ,…,y_0) = i if $I_i = 1$ & En = 1 & $I_k = 0$
for all k > i (high bit priority) or
for all k< i (low bit priority).



$E_o = 1$ if En = 1 & $I_i = 0$ for all i,
$G_s = 1$ if En = 1 & ∃ i s.t. $I_i = 1$.

(G_s is like A, and E_o tells us if enable is true or not).

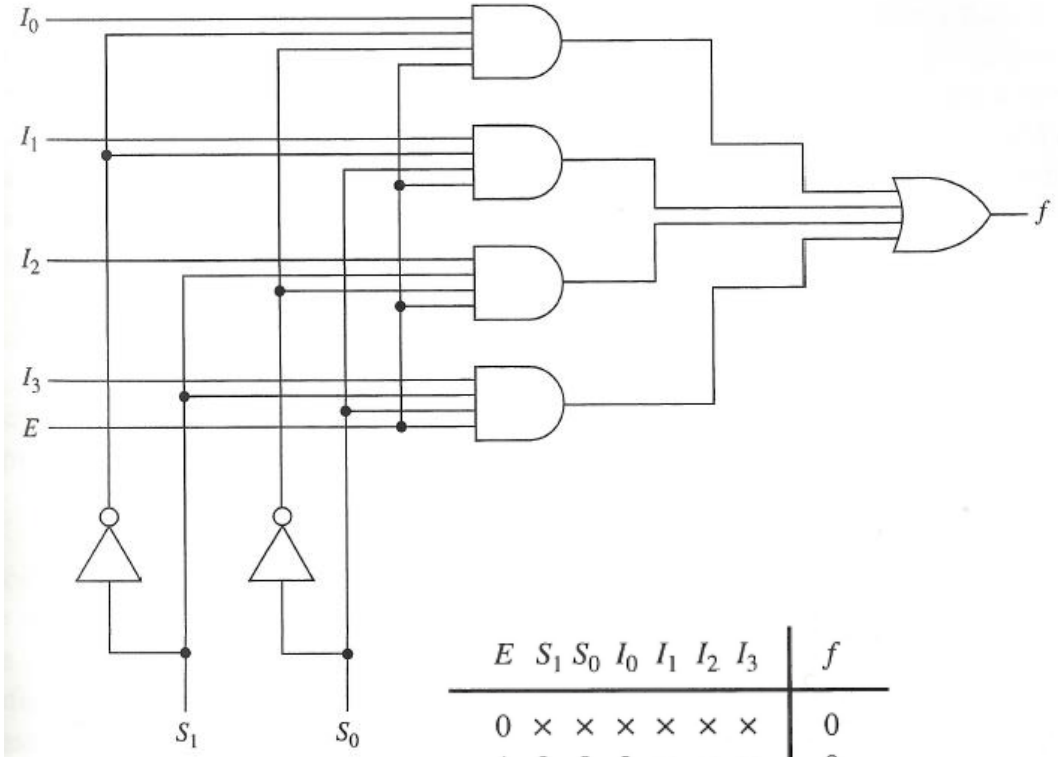
Priority Encoder: Implement a 32-input priority encoder w/ 8 input priority encoders (high bit priority).



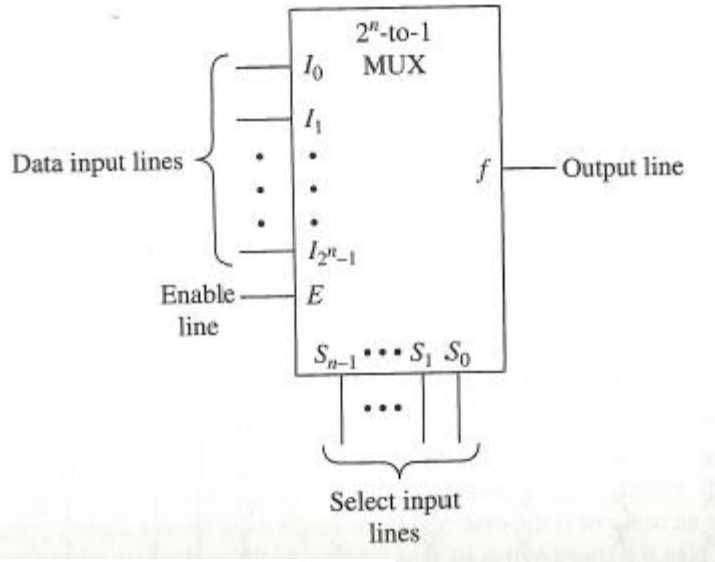
Multiplexer

- Also called data selectors.
- Basic function: select one of its 2^n data input lines and place the corresponding information onto a single output line.
- n input bits needed to specify which input line is to be selected.
  - Place binary code for a desired data input line onto its n select input lines.

Realization of 4-to-1 line

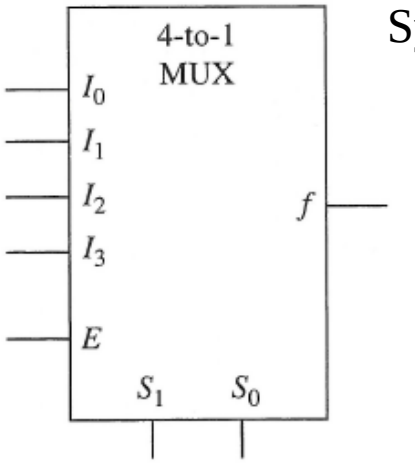


Logic Diagram



E	S_1	S_0	I_0	I_1	I_2	I_3	f
0	×	×	×	×	×	×	0
1	0	0	0	×	×	×	0
1	0	0	1	×	×	×	1
1	0	1	×	0	×	×	0
1	0	1	×	1	×	×	1
1	1	0	×	×	0	×	0
1	1	0	×	×	1	×	1
1	1	1	×	×	×	0	0
1	1	1	×	×	×	1	1

Truth Table



Symbol

Realization of 4-to-1 line multiplexer

- Alternate description:

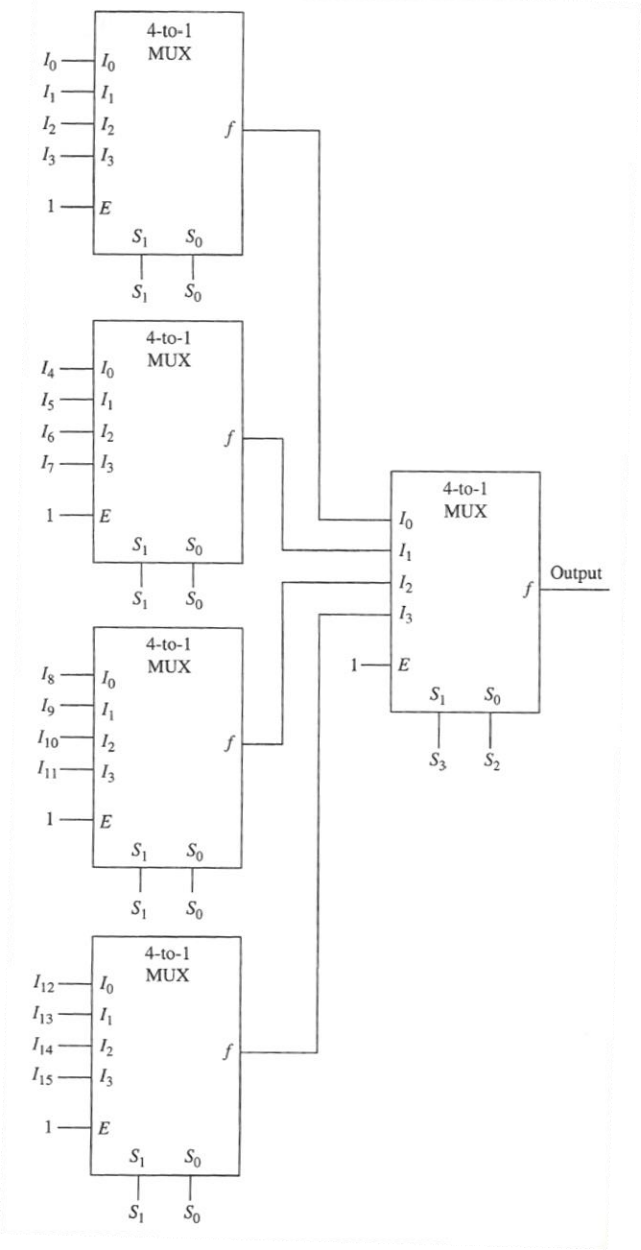
Table 5.6 Function table for a 4-to-1-line multiplexer

E	S_1	S_0	f
0	×	×	0
1	0	0	I_0
1	0	1	I_1
1	1	0	I_2
1	1	1	I_3

- Algebraic description of multiplexer:

$$f = \left(I_0\overline{S}_1\overline{S}_0 + I_1\overline{S}_1S_0 + I_2S_1\overline{S}_0 + I_3S_1S_2\right)E$$

Building a Large Multiplexer



Multiplexers

- One of the primary applications of multiplexers is to provide for the transmission of information from several sources over a single path.
- This process is known as multiplexing.
- Demultiplexer = decoder with an enable input.

Multiplexer/De-multiplexer for information transmission

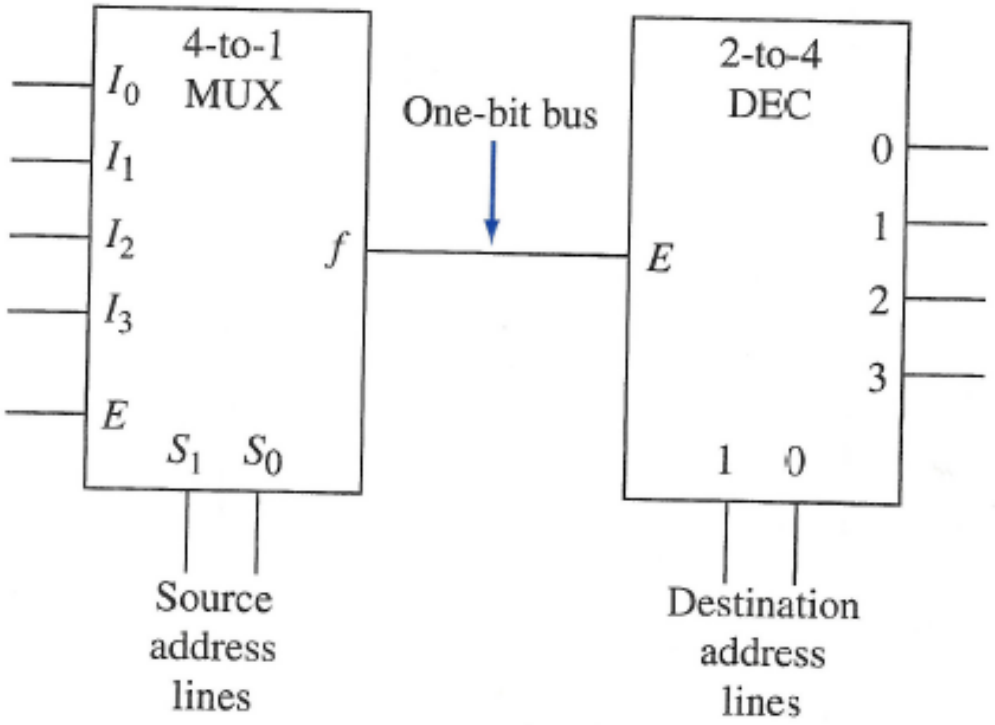


Figure 5.35 A multiplexer/demultiplexer arrangement for information transmission.