Combinational Circuit

COE 233

Digital Logic and Computer Organization

Dr. Muhamed Mudawar

King Fahd University of Petroleum and Minerals

Presentation Outline

- Half-Adder, Full-Adder, and Ripple-Carry Adder
- Decoders, Implementing Functions with Decoders
- Multiplexers
- Design Examples

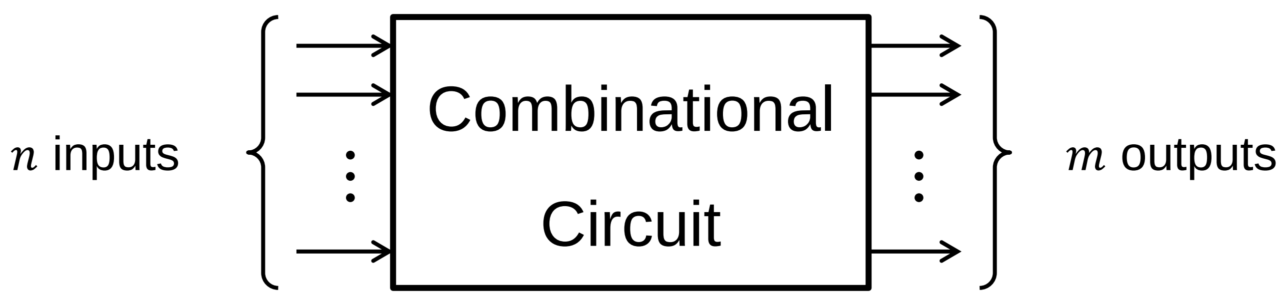
Combinational Circuit

- A combinational circuit is a block of logic gates having:

  n inputs: $x_1, x_2, \ldots, x_n$

  m outputs: $f_1, f_2, \ldots, f_m$

- Each output is a function of the input variables
- Each output is determined from present combination of inputs
- Combination circuit performs operation specified by logic gates



Half Adder

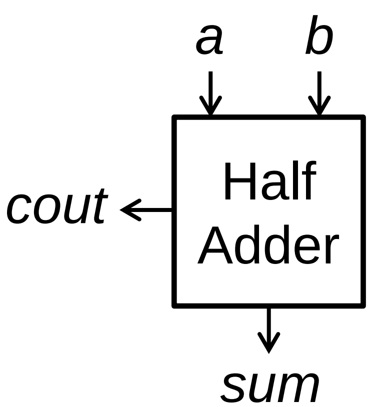
A half adder adds two bits: **a** and **b**

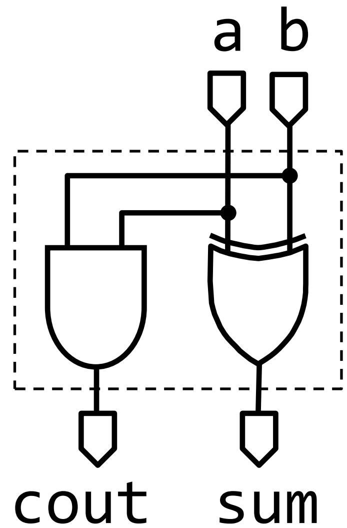
Two output bits:

1. Carry bit: $\mathbf{cout = a \cdot b}$
2. Sum bit: $\mathbf{sum = a \oplus b}$

Truth Table

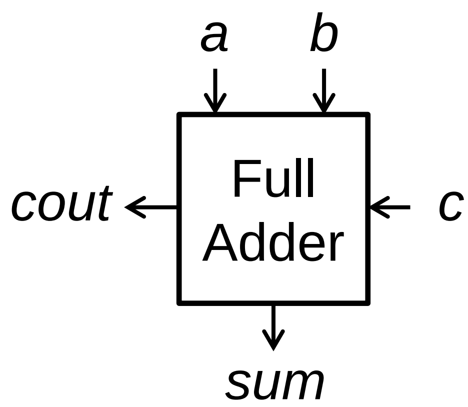
a	b	cout	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0





Full Adder

- A full adder adds 3 bits: **a**, **b**, and **c**
- Two output bits:
  1. Carry bit: **cout**
  2. Sum bit: **sum**



- Sum bit is 1 if **odd input** of 1's

  sum = (a ⊕ b) ⊕ c (odd function)

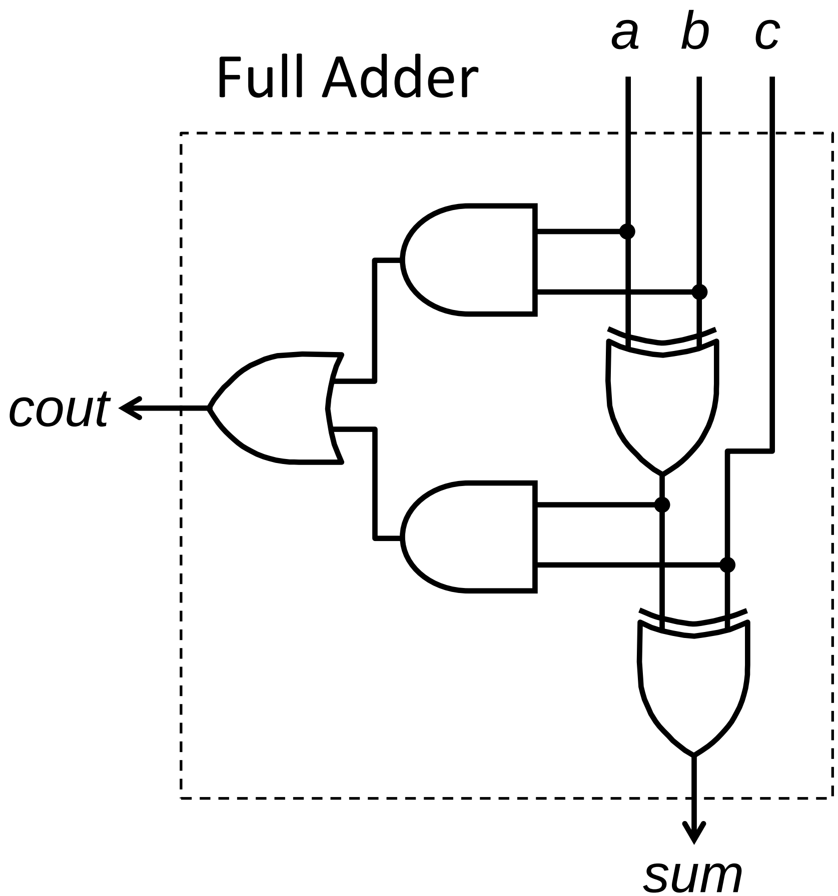
- Carry bit is 1 if the input of 1's is 2 or 3

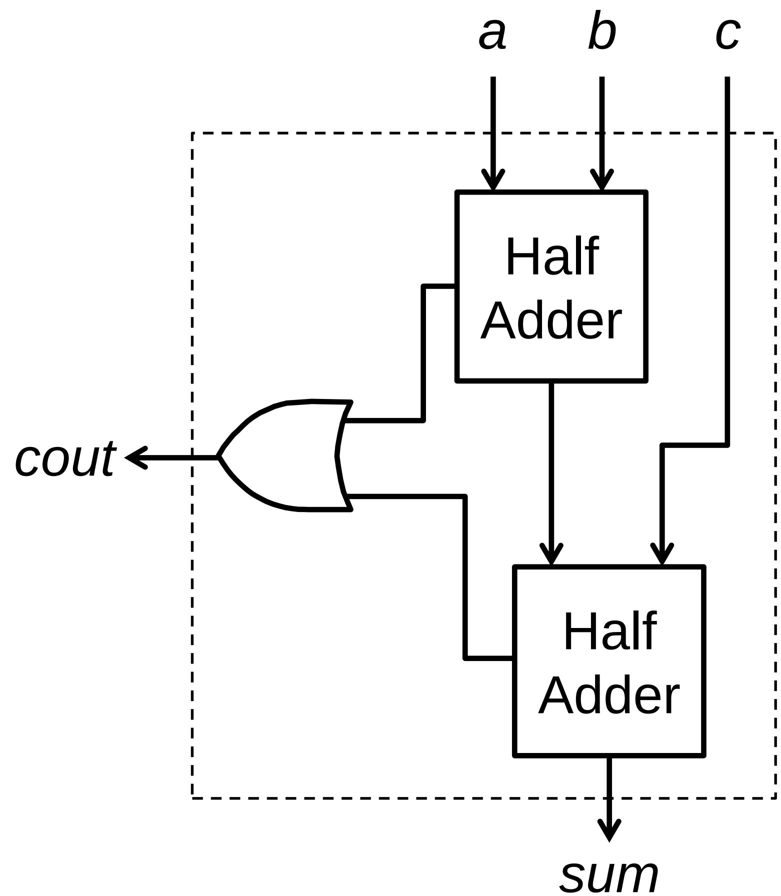
  cout = a·b + (a ⊕ b)·c

Truth Table

a b c	cout	sum
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

Full Adder Implementation





A full adder can be implemented as two half-adders and an OR gate

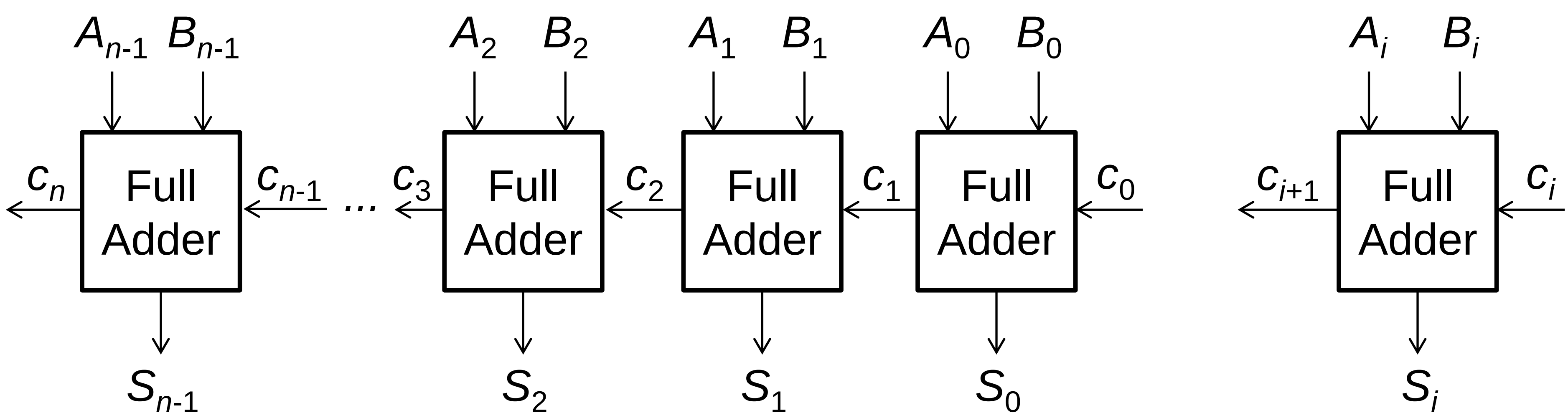
Binary Addition

- Start with the least significant bit (rightmost bit)
- Add each pair of bits
- Include the carry in the addition

carry		1	1	1	1				
	0	0	1	1	0	1	1	0	(54)
+	0	0	0	1	1	1	0	1	(29)
	0	1	0	1	0	0	1	1	(83)
bit position:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	

Ripple Carry Adder

- Adds two ***n*-bit** numbers: A = {$A_{n\text{-}1}$, …, A_0} and B = {$B_{n\text{-}1}$, …, B_0}
- Uses **identical copies** of a full adder to build a large *n*-bit adder
- Iterative design: the **cell** is a **full adder**
- Carry propagation: carry-out of cell *i* becomes carry-in to cell *i* +1

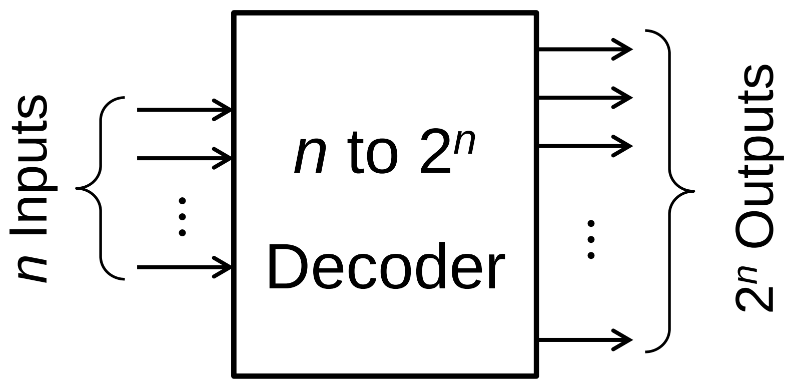


Next . . .

- ❖ Half-Adder, Full-Adder, and Ripple-Carry Adder
- ❖ Decoders, Implementing Functions with Decoders
- ❖ Multiplexers
- ❖ Design Examples

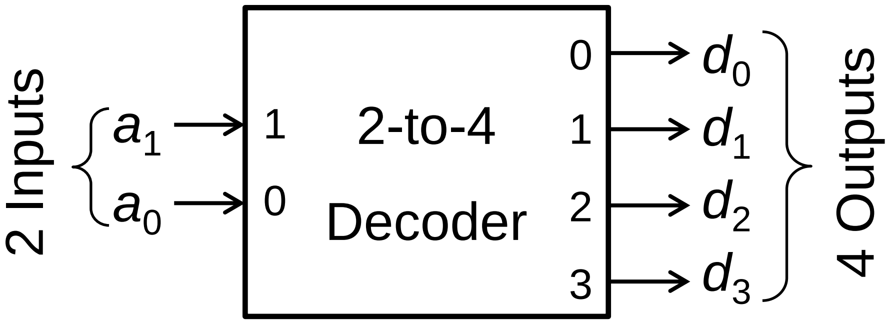
Binary Decoders

- ❖ Given a n-bit binary code, there are 2^n possible code values
- ❖ The decoder has an output for each possible code value
- ❖ The n-to-2^n decoder has n inputs and 2^n outputs
- ❖ Depending on the input code, **only one output** is set to **logic 1**
- ❖ The conversion of input to output is called **decoding**



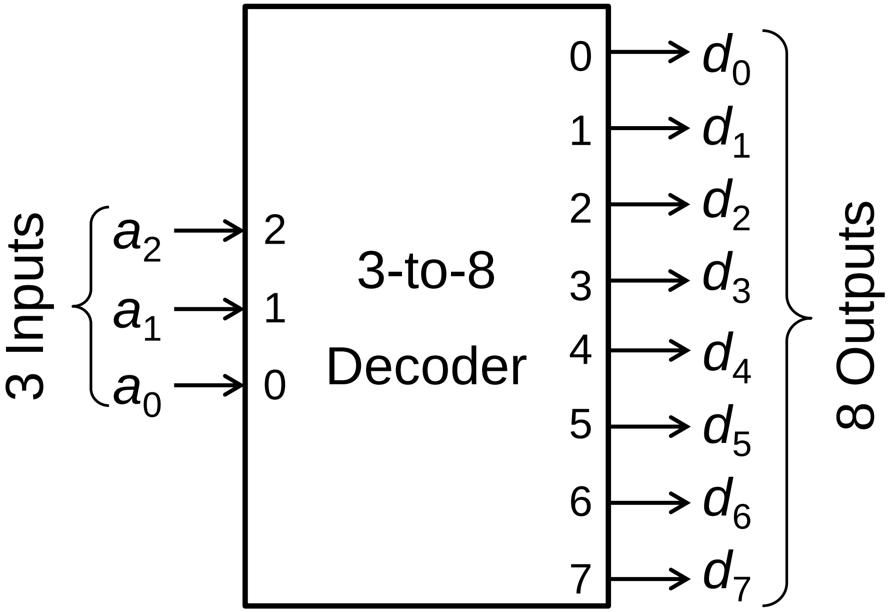
A decoder can have less than 2^n outputs if some input codes are unused

Examples of Binary Decoders



Inputs	Outputs			
a_1 a_0	d_0	d_1	d_2	d_3
0 0	1	0	0	0
0 1	0	1	0	0
1 0	0	0	1	0
1 1	0	0	0	1

Truth Tables

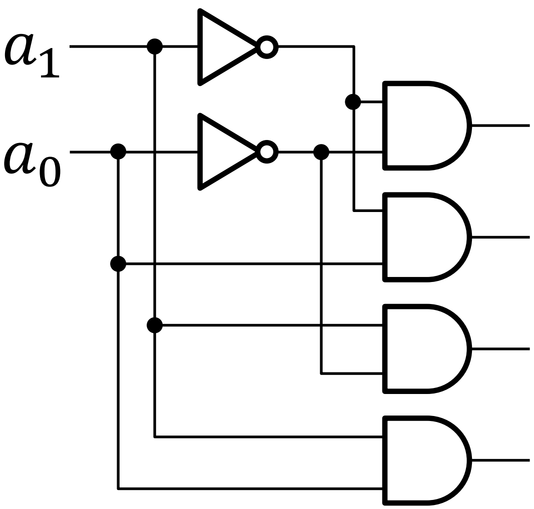


Inputs	Outputs							
a_2 a_1 a_0	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7
0 0 0	1	0	0	0	0	0	0	0
0 0 1	0	1	0	0	0	0	0	0
0 1 0	0	0	1	0	0	0	0	0
0 1 1	0	0	0	1	0	0	0	0
1 0 0	0	0	0	0	1	0	0	0
1 0 1	0	0	0	0	0	1	0	0
1 1 0	0	0	0	0	0	0	1	0
1 1 1	0	0	0	0	0	0	0	1

Decoder Implementation

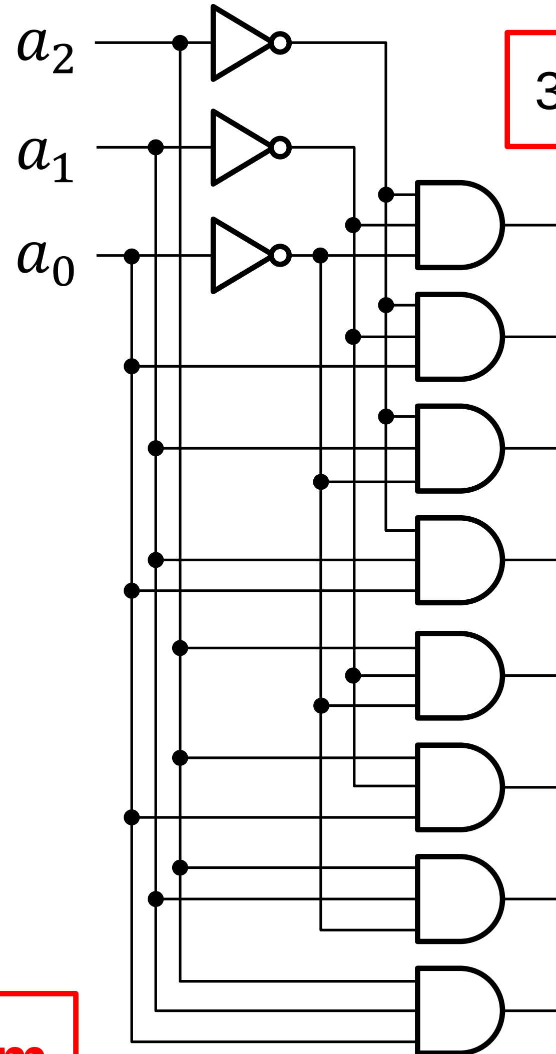
Inputs	Outputs			
a_1 a_0	d_0	d_1	d_2	d_3
0 0	1	0	0	0
0 1	0	1	0	0
1 0	0	0	1	0
1 1	0	0	0	1

2-to-4 Decoder



$$d_0 = a_1'a_0'$$
$$d_1 = a_1'a_0$$
$$d_2 = a_1a_0'$$
$$d_3 = a_1a_0$$

3-to-8 Decoder



$$d_0 = a_2'a_1'a_0'$$
$$d_1 = a_2'a_1'a_0$$
$$d_2 = a_2'a_1a_0'$$
$$d_3 = a_2'a_1a_0$$
$$d_4 = a_2a_1'a_0'$$
$$d_5 = a_2a_1'a_0$$
$$d_6 = a_2a_1a_0'$$
$$d_7 = a_2a_1a_0$$

Each decoder output is a **minterm**

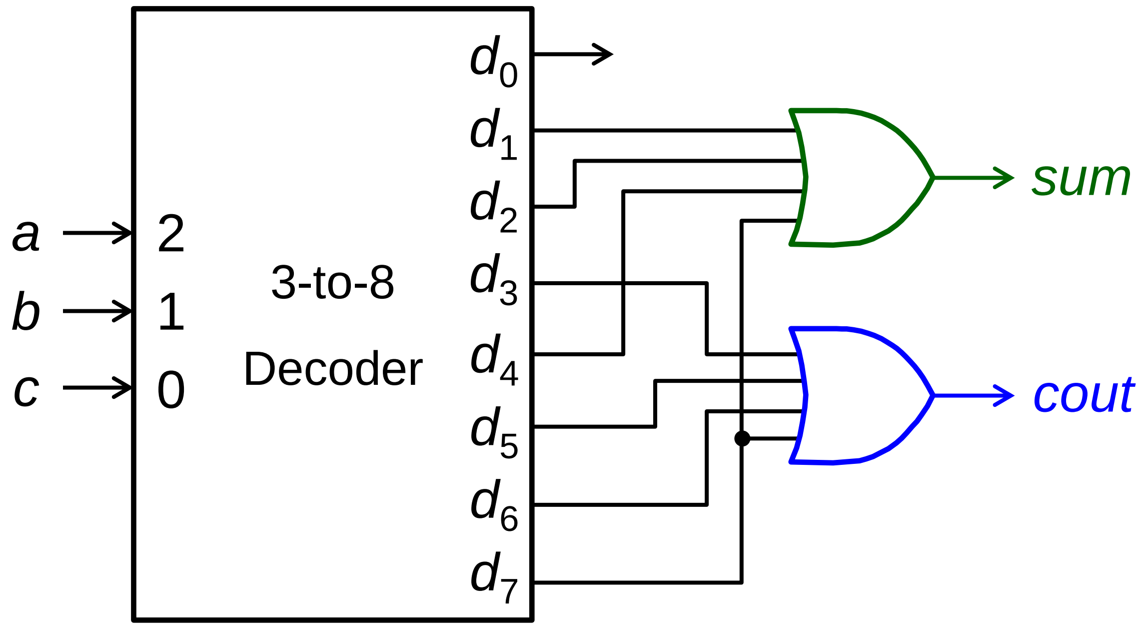
Using Decoders to Implement Functions

- A decoder generates all the minterms
- A Boolean function can be expressed as a sum of minterms
- Any function can be implemented using a decoder + OR gate

  Note: the function **must not be minimized**

- **Example:** Full Adder sum = ∑(1, 2, 4, 7), cout = ∑(3, 5, 6, 7)

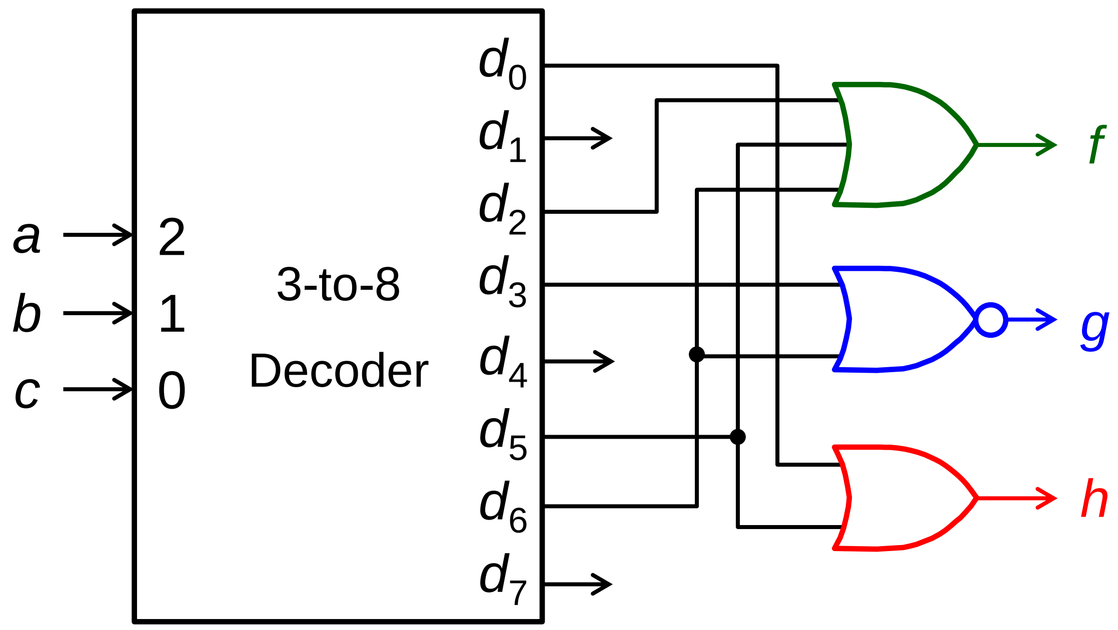
Inputs			Outputs	
a	b	c	cout	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Using Decoders to Implement Functions

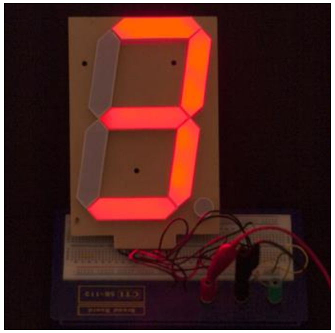
- Good if many output functions of the same input variables
- If number of minterms is large ➔ Wider OR gate is needed
- Use NOR gate if number of maxterms is less than minterms
- **Example:** $f = \sum(2, 5, 6)$, $g = \prod(3, 6)$ ➔ $g' = \sum(3, 6)$, $h = \sum(0, 5)$

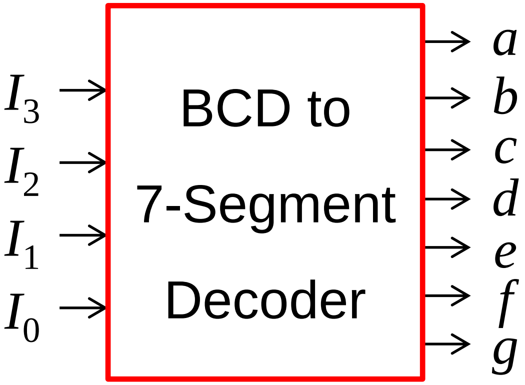
Inputs			Outputs		
a	b	c	f	g	h
0	0	0	0	1	1
0	0	1	0	1	0
0	1	0	1	1	0
0	1	1	0	0	0
1	0	0	0	1	0
1	0	1	1	1	1
1	1	0	1	0	0
1	1	1	0	1	0



BCD to 7-Segment Decoder

- Seven-Segment Display:
  - Made of Seven segments: light-emitting diodes (LED)
  - Found in electronic devices: such as clocks, calculators, etc.





- BCD to 7-Segment Decoder
  - Called also a decoder, but not a binary decoder
  - Accepts as input a BCD decimal digit (0 to 9)
  - Generates output to the seven LED segments to display the BCD digit
  - Each segment can be turned on or off separately

BCD to 7-Segment Decoder

Specification:

- ✧ Input: 4-bit BCD (I_3, I_2, I_1, I_0)
- ✧ Output: 7-bit (a, b, c, d, e, f, g)
- ✧ Display should be OFF for Non-BCD input codes.

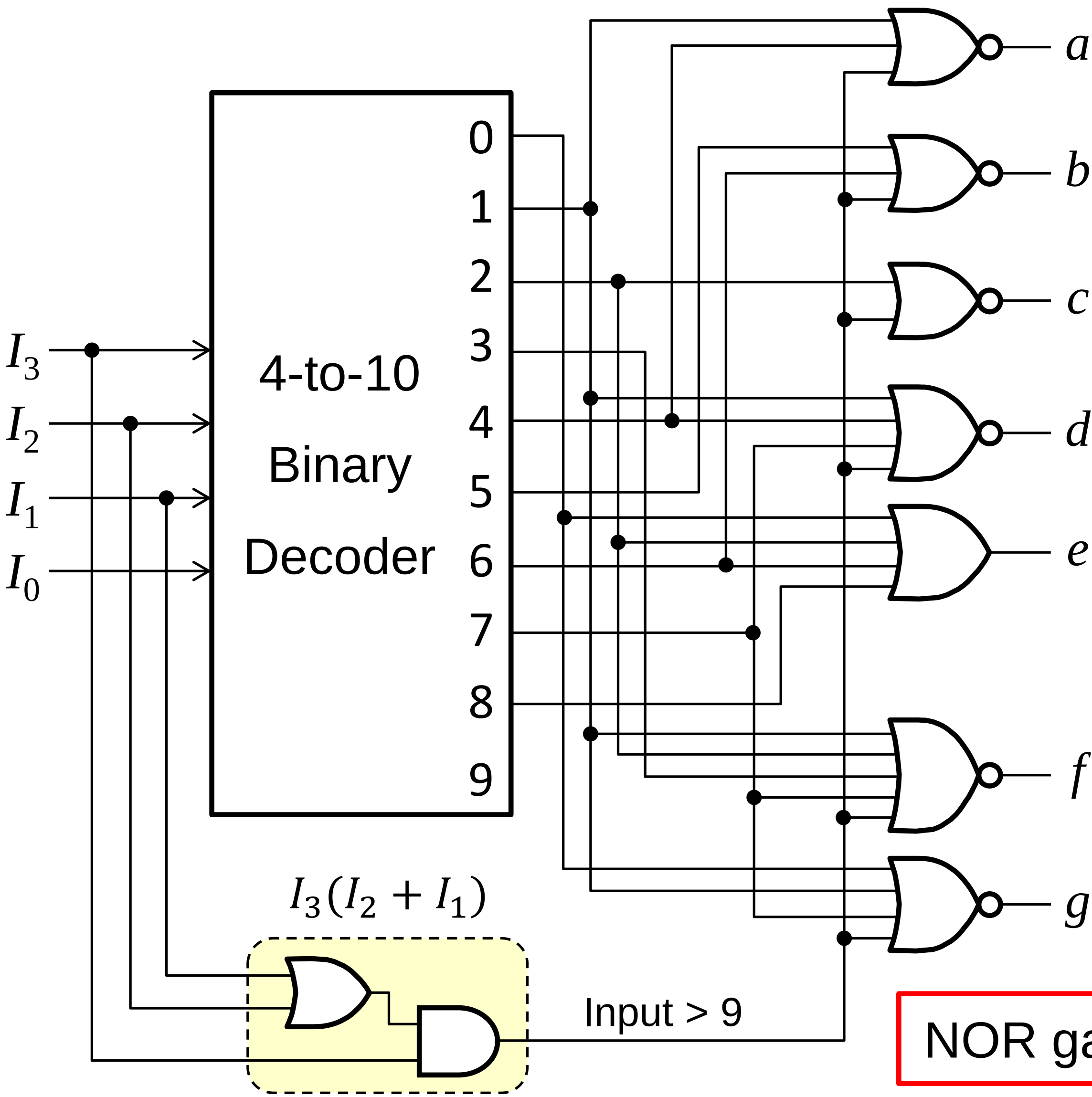
Implementation can use:

- ✧ A binary decoder
- ✧ Additional gates

Truth Table

BCD input				7-Segment Output						
I_3	I_2	I_1	I_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1010 to 1111				0	0	0	0	0	0	0

Implementing a BCD to 7-Segment Decoder



Truth Table

I_3	I_2	I_1	I_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1010 – 1111				0	0	0	0	0	0	0

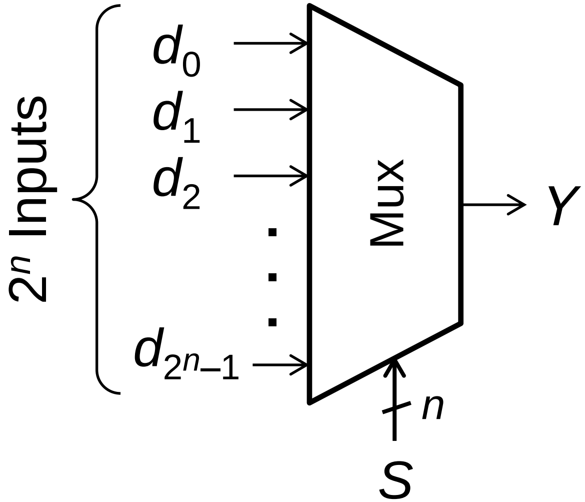
NOR gate is used for **0**'s

Next . . .

- Half-Adder, Full-Adder, and Ripple-Carry Adder
- Decoders, Implementing Functions with Decoders
- Multiplexers
- Design Examples

Multiplexers

- Selecting data is an essential function in digital systems
- Functional blocks that perform selecting are called **multiplexers**
- A Multiplexer (or Mux) is a combinational circuit that has:
  - Multiple data inputs (typically 2^n) to select from
  - An n-bit select input S used for control
  - One output Y



- The n-bit select input directs one of the data inputs to the output

Examples of Multiplexers

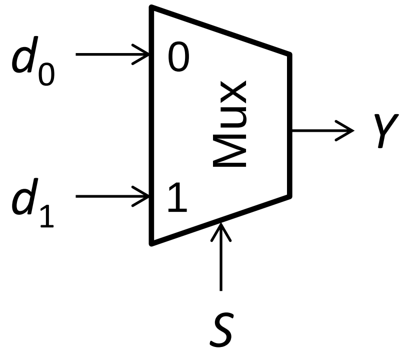
❖ 2-to-1 Multiplexer

if (S == 0) $Y = d_0$;

else $Y = d_1$;

Logic expression:

$$Y = d_0\ S' + d_1\ S$$



Inputs			Output
S	d_0	d_1	Y
0	0	X	0 = d_0
0	1	X	1 = d_0
1	X	0	0 = d_1
1	X	1	1 = d_1

❖ 4-to-1 Multiplexer

if (S_1S_0 == 00) $Y = d_0$;

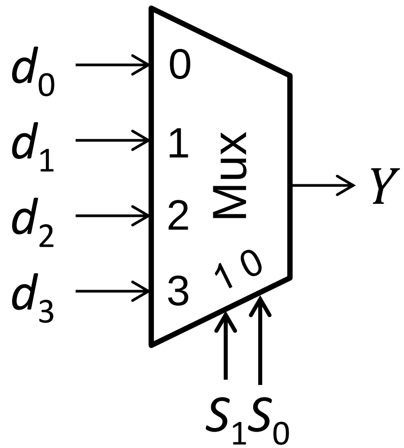
else if (S_1S_0 == 01) $Y = d_1$;

else if (S_1S_0 == 10) $Y = d_2$;

else $Y = d_3$;

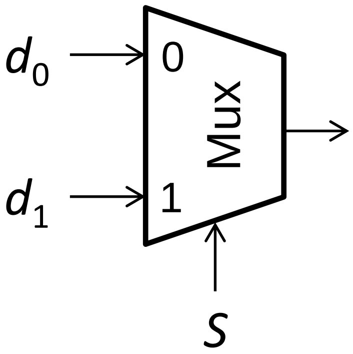
Logic expression:

$$Y = d_0\ S_1'S_0' + d_1\ S_1'S_0 + d_2\ S_1S_0' + d_3\ S_1S_2$$

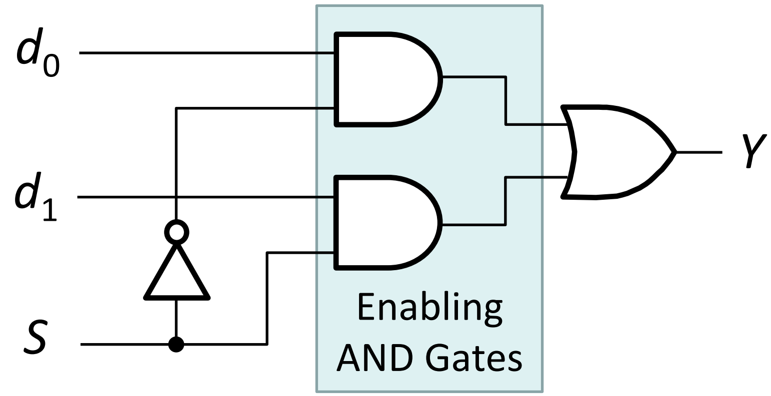


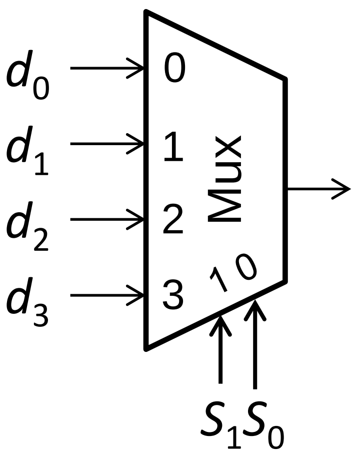
Inputs						Output
S_1	S_0	d_0	d_1	d_2	d_3	Y
0	0	0	X	X	X	0 = d_0
0	0	1	X	X	X	1 = d_0
0	1	X	0	X	X	0 = d_1
0	1	X	1	X	X	1 = d_1
1	0	X	X	0	X	0 = d_2
1	0	X	X	1	X	1 = d_2
1	1	X	X	X	0	0 = d_3
1	1	X	X	X	1	1 = d_3

Implementing Multiplexers

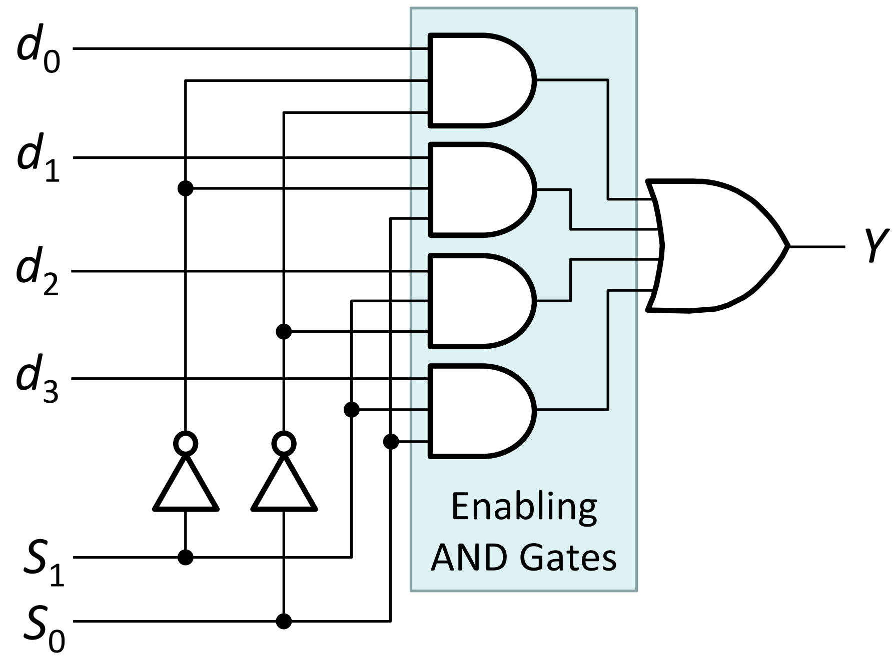


$$Y = d_0\, S' + d_1\, S$$



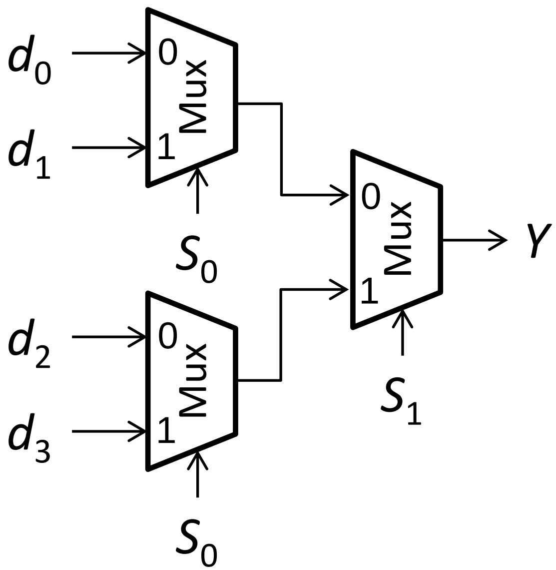


$$Y = d_0\, S_1' S_0' + d_1\, S_1' S_0 + d_2\, S_1 S_0' + d_3\, S_1 S_0$$

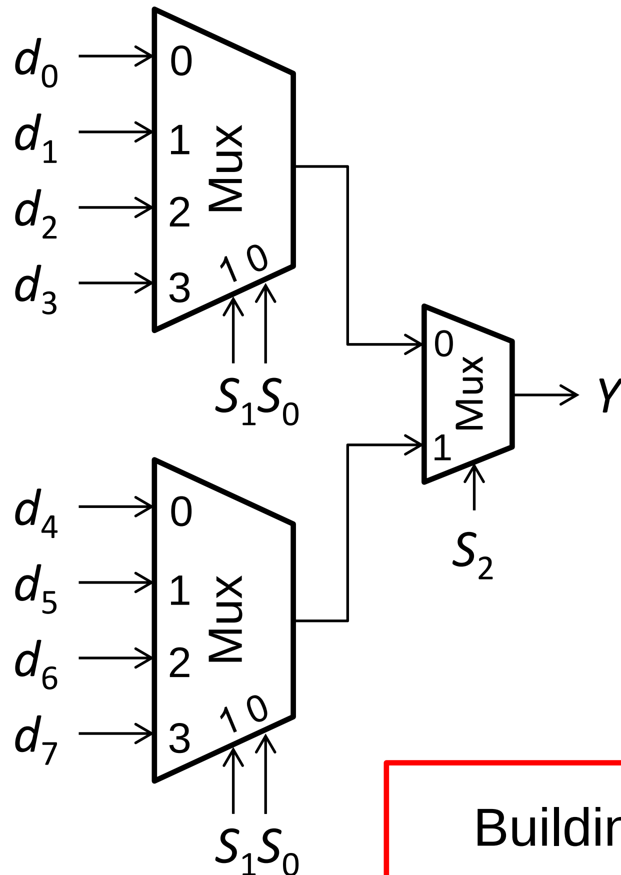


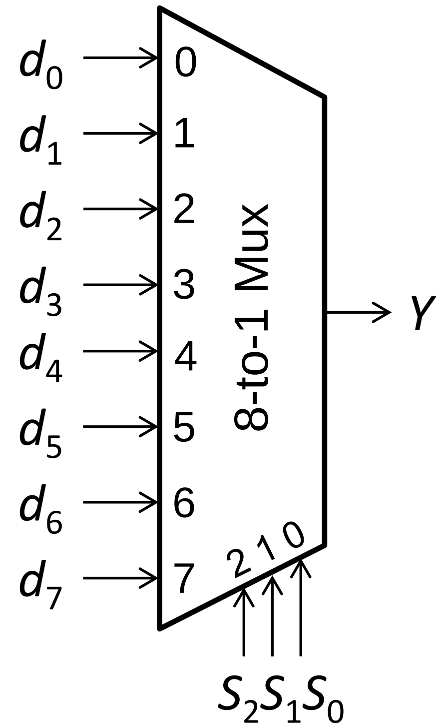
Building Larger Multiplexers

Larger multiplexers can be built hierarchically using smaller ones



Building 4-to-1 Mux using three 2-to-1 Muxes

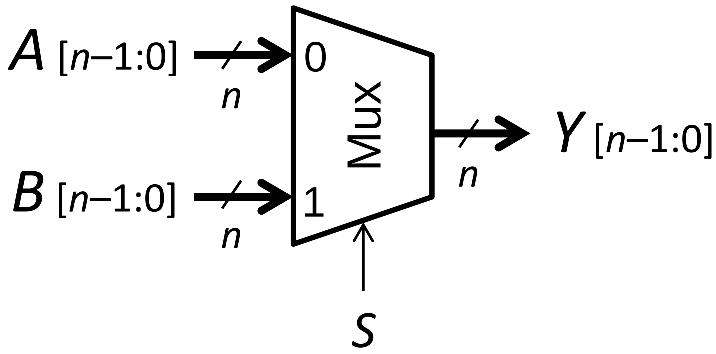


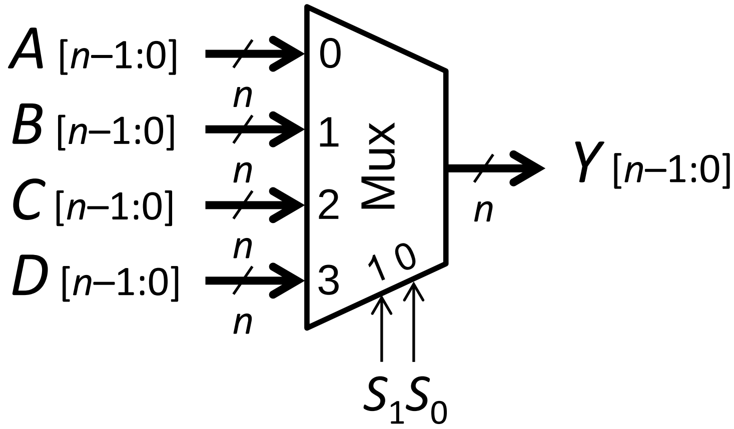


Building 8-to-1 Mux using two 4-to-1 Muxes and a 2-to-1 Mux

Multiplexers with Vector Input and Output

The inputs and output of a multiplexer can be *n*-bit vectors





2-to-1 Multiplexer with *n* bits

Inputs and output are *n*-bit vectors

Using *n* copies of a 2-to-1 Mux

4-to-1 Multiplexer with *n* bits

Inputs and output are *n*-bit vectors

Using *n* copies of a 4-to-1 Mux

Next . . .

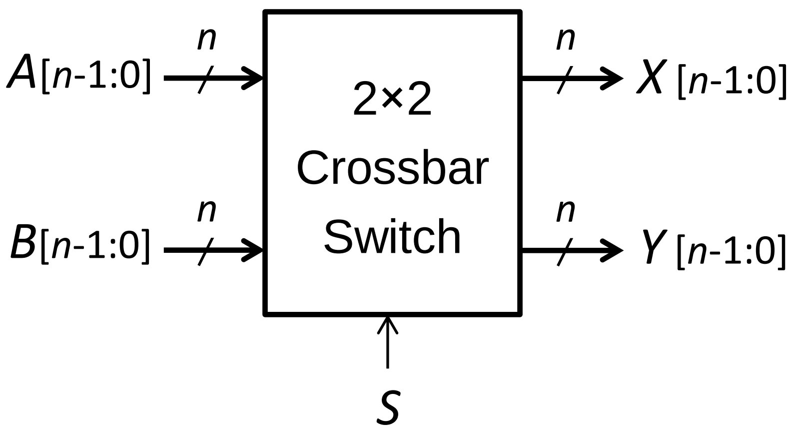
- Half-Adder, Full-Adder, and Ripple-Carry Adder
- Decoders, Implementing Functions with Decoders
- Multiplexers
- Design Examples

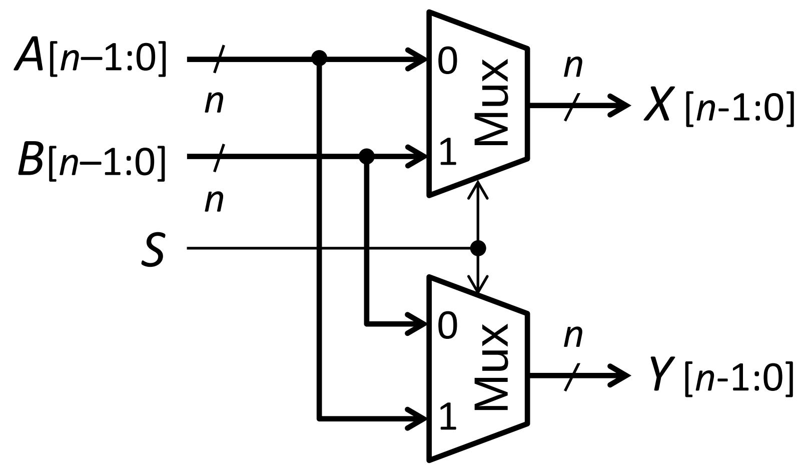
2-by-2 Crossbar Switch

- A 2×2 crossbar switch is a combinational circuit that has:
  - Two n-bit Inputs: A and B
  - Two n-bit outputs: X and Y
  - 1-bit select input S

if (S == 0) { $X = A$; $Y = B$; }
else { $X = B$; $Y = A$; }

- Implement the 2×2 crossbar switch using multiplexers
- **Solution:** Two n-bit multiplexers are used

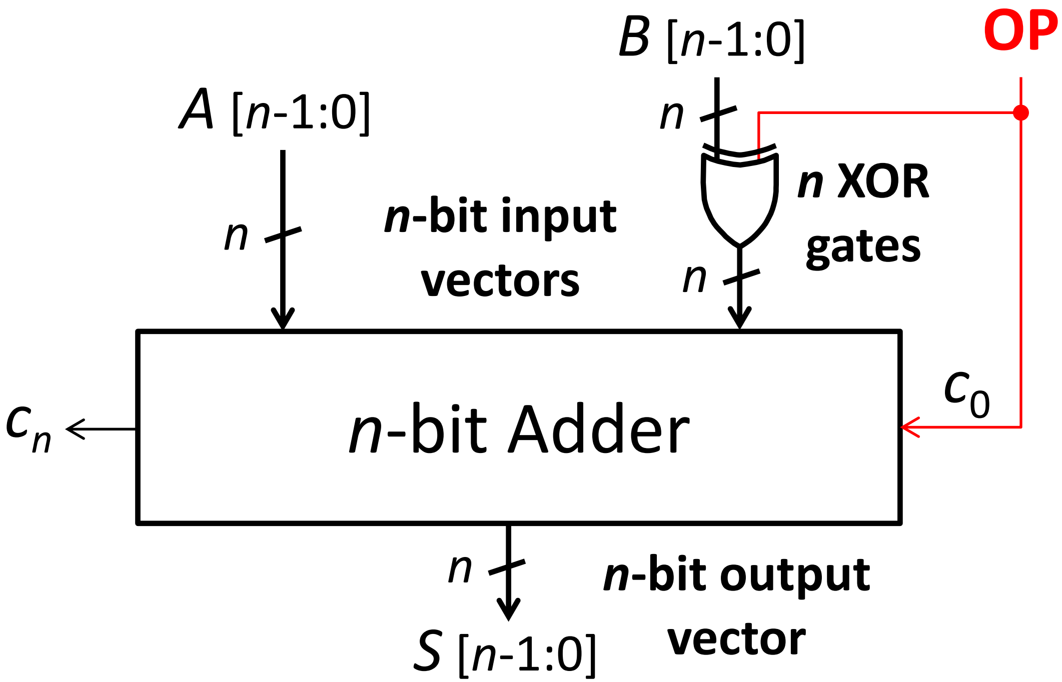




Addition and Subtraction with Same Adder

- Same adder can be used to compute: (A + B) and (A – B)
- Two operations: **OP = 0 (ADD)**, **OP = 1 (SUBTRACT)**
- Subtraction (A – B) is computed as: A + (2's complement of B)

  2's complement of B = (1's complement of B) + 1



OP = 0 (ADD)

B XOR 0 = B

S = A + B + 0 = A + B

OP = 1 (SUBTRACT)

B XOR 1 = 1's complement of B

S = A + (1's complement of B) + 1

S = A + (2's complement of B)

S = A – B

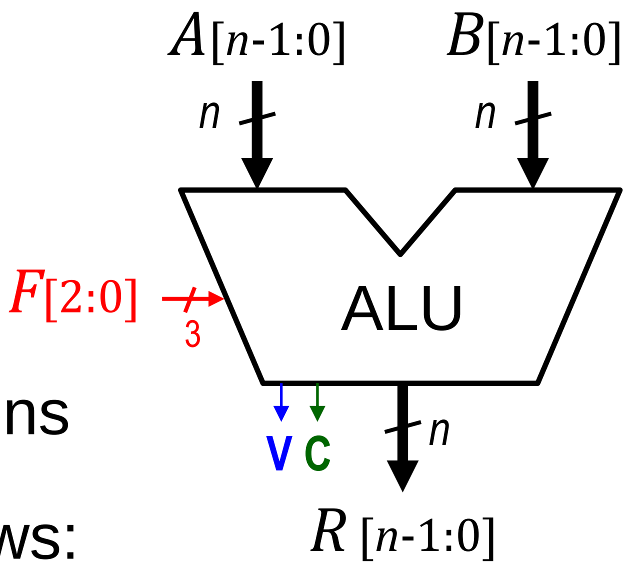
Arithmetic and Logic Unit (ALU)

- Can perform many functions
- Most common ALU functions

  Arithmetic functions: ADD, SUB (Subtract)

  Logic functions: AND, OR, XOR, etc.
- We will design a simple ALU with 8 functions
- The function *F* is coded with 3 bits as follows:

ALU Symbol



Function	ALU Result	Function	ALU Result
F = 000 (ADD)	R = A + B	F = 100 (AND)	R = A & B
F = 001 (ADD + 1)	R = A + B + 1	F = 101 (OR)	R = A | B
F = 010 (SUB – 1)	R = A – B – 1	F = 110 (NOR)	R = ~(A | B)
F = 011 (SUB)	R = A – B	F = 111 (XOR)	R = (A ^ B)

Designing a Simple ALU