

Logic Circuits I

Boolean Algebra

- A set of two elements, $B = \{0,1\}$, together with three operations \vee , \wedge , and $'$, which satisfies the following axioms:
 - Axiom 1 (Closure)
 - For all x and y in B both $x \vee y$ and $x \wedge y$ are in B
 - Axiom 2 (Identity element)
 - There exist distinct elements 0 and 1 in B such that for all x in B , $x \vee 0 = x$ and $x \wedge 1 = x$
 - Axiom 3 (Commutativity)
 - For all x and y in B , $x \vee y = y \vee x$ and $x \wedge y = y \wedge x$
 - Axiom 4 (Distributivity)
 - For all x, y , and z in B , $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ and $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
 - Axiom 5 (Complement)
 - For each x in B , there exists an element in B , denoted x' and called the complement or negation of x , such that $x \vee x' = 1$ and $x \wedge x' = 0$
 - Axiom 6 (Cardinality)
 - There are at least two distinct elements in B .

OR, AND, and NOT Operations

$$\begin{aligned}x \vee y &= \begin{cases} 1 & \text{when either } x \text{ or } y \text{ is 1, or both are 1} \\ 0 & \text{otherwise} \end{cases} \\x \wedge y &= \begin{cases} 1 & \text{when both } x \text{ and } y \text{ are 1} \\ 0 & \text{otherwise} \end{cases} \\x' &= \begin{cases} 1 & \text{when } x = 0 \\ 0 & \text{otherwise} \end{cases}\end{aligned}$$

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

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

NOT	
x	x'
0	1
1	0

Theorems

- Property 1 (Uniqueness of 0 and 1)
 - 0 and 1 are unique
- Property 2 (Idempotency)
 - For all x in B , $x \vee x = x$ and $x \wedge x = x$
- Property 3
 - For all x in B , $x \vee 1 = 1$ and $x \wedge 0 = 0$
- Property 4 (Absorption)
 - For all x and y in B , $(x \vee y) \wedge x = x$ and $(x \wedge y) \vee x = x$

Theorems (contd.)

- Property 5 (Associativity)
 - For all x, y , and z in B , $(x \vee y) \vee z = x \vee (y \vee z)$ and $(x \wedge y) \wedge z = x \wedge (y \wedge z)$
- Property 6 (The uniqueness of complement)
 - For all x in B , x' is unique
- Property 7 (Involution)
 - For all x in B , $(x')' = x$
- Property 8 (De Morgan's law)
 - For all x and y in B , $(x \vee y)' = x' \wedge y'$ and $(x \wedge y)' = x' \vee y'$

Boolean Expressions

- A Boolean expression is a string of symbols involving constants 0 and 1, some variables, and Boolean operations \vee , \wedge , and $'$
- A Boolean expression in n variables x_1, x_2, \dots , and x_n is defined inductively as follows:
 - Each of the symbols 0, 1, x_1, x_2, \dots , and x_n is a Boolean expression
 - If e_1 and e_2 are Boolean expressions, so are e_1' , $(e_1 \vee e_2)$, and $(e_1 \wedge e_2)$

$$((x \vee y)' \vee (x' \wedge y))$$

$$(((x \wedge y) \vee (x \wedge z')) \vee (y' \vee y)')$$

Boolean Functions

- If e is a Boolean expression in n variables x_1, x_2, \dots , and x_n , then e defines a Boolean function mapping B^n into B
- A truth table is the simplest way to specify a Boolean function

$$f(x, y, z) = ((x \vee y) \wedge z')$$

x	y	z	$f(x, y, z)$ $= ((x \vee y) \wedge z')$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Number of Different Boolean Functions

- The number of different Boolean functions with n binary variables is

$$2^{2^n}$$

$f(x, y)$	xy				Comment
	00	01	10	11	
0	0	0	0	0	Constant 0
$x \wedge y$	0	0	0	1	x and y (AND)
$x \wedge y'$	0	0	1	0	x but not y
x	0	0	1	1	x
$x' \wedge y$	0	1	0	0	y but not x
y	0	1	0	1	y
$(x \wedge y') \vee (x' \wedge y)$	0	1	1	0	x or y but not both (XOR)
$x \vee y$	0	1	1	1	x or y (OR)
$(x \vee y)'$	1	0	0	0	NOR
$(x \wedge y) \vee (x' \wedge y')$	1	0	0	1	x equals y (XNOR)
y'	1	0	1	0	NOT y
$x \vee y'$	1	0	1	1	If y then x
x'	1	1	0	0	NOT x
$x' \vee y$	1	1	0	1	If x then y
$(x \wedge y)'$	1	1	1	0	NAND
1	1	1	1	1	Constant 1

Functional Completeness

- A set of Boolean operations is functionally complete if its members can construct all other Boolean functions for any given set of input variables
 - We assume that these operations can be applied as many times as needed
- A well known complete set of Boolean operations is {AND, OR, NOT}

XOR and XNOR

- Exclusive OR and exclusive NOT-OR

$$\begin{aligned}x \oplus y &= (x \wedge y') \vee (x' \wedge y) = \begin{cases} 1 & \text{when } x \neq y \\ 0 & \text{otherwise} \end{cases} \\x \odot y &= (x \wedge y) \vee (x' \wedge y') = \begin{cases} 1 & \text{when } x = y \\ 0 & \text{otherwise} \end{cases}\end{aligned}$$

NOR and NAND

- NOT-OR

$$(x \vee y)'$$

- NOT-AND

$$(x \wedge y)'$$

- {NOR} and {NAND} are also functionally complete

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

XNOR		
x	y	$x \odot y$
0	0	1
0	1	0
1	0	0
1	1	1

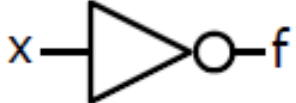






NOR		
x	y	$(x \vee y)'$
0	0	1
0	1	0
1	0	0
1	1	0

NAND		
x	y	$(x \wedge y)'$
0	0	1
0	1	1
1	0	1
1	1	0

Logic Gates

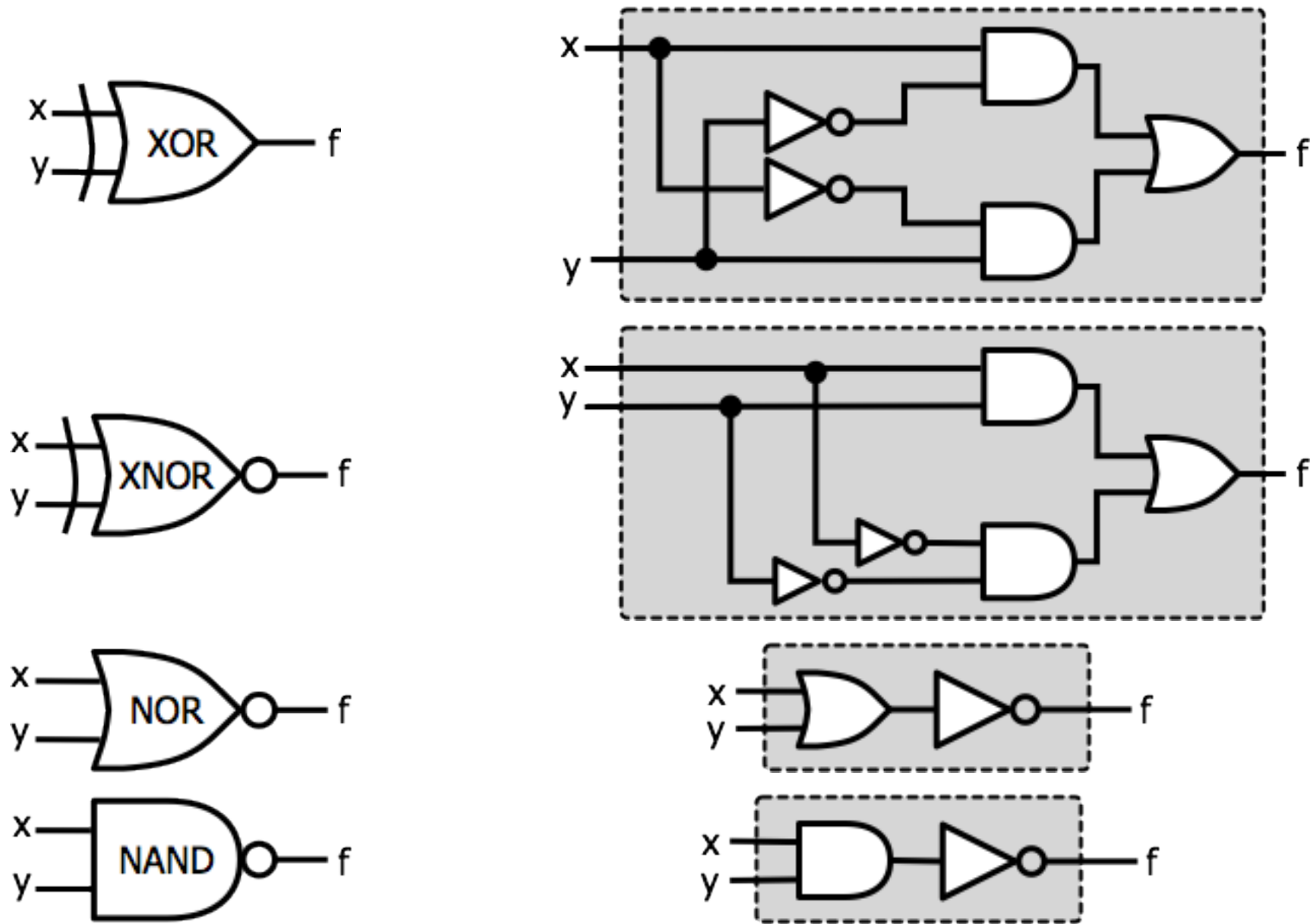
- A logic gate is a conceptual or physical device that performs one or more Boolean operations
- A Boolean function can be implemented with a logic gate
- A logic gate can be viewed as a block box
 - $f: B^n \rightarrow B^m$
 - n input variables and m outputs
 - n input pins and m output pins
- A logic diagram is a graphical representation of a logic circuit that shows connections between logic gates

Some Elementary Logic Gates

Name	Symbol	Function
NOT		$f = x'$
OR		$f = x \vee y$
AND		$f = x \wedge y$
NOR		$f = (x \vee y)'$
NAND		$f = (x \wedge y)'$
XOR		$f = x \oplus y$
XNOR		$f = x \odot y$

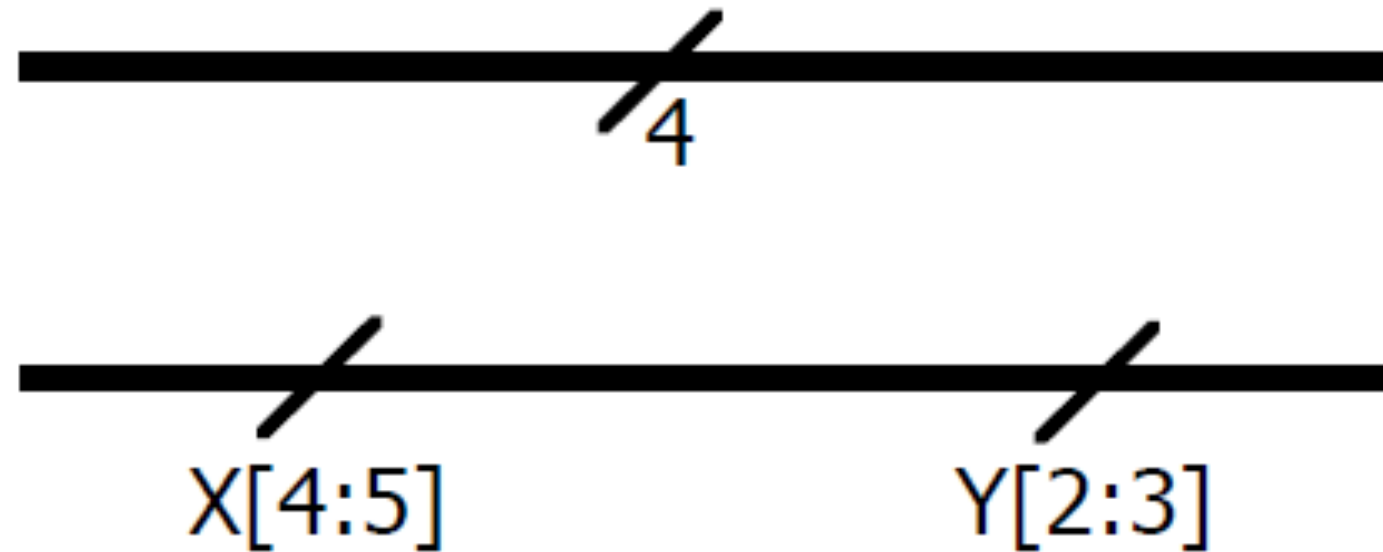
Combining Logic Gates

- A logic gate with more complicated functionality can be implemented by combining and interconnecting some elementary logic gates



Bus Notation

- A bus is a collection of two or more related signal lines

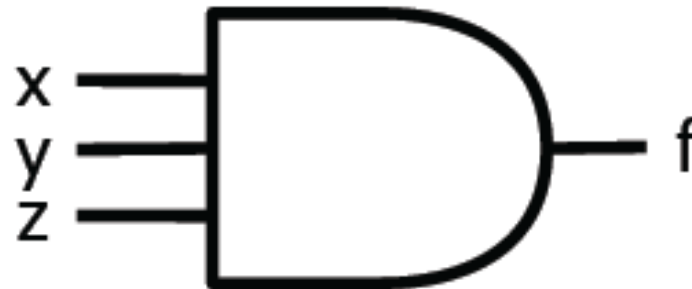
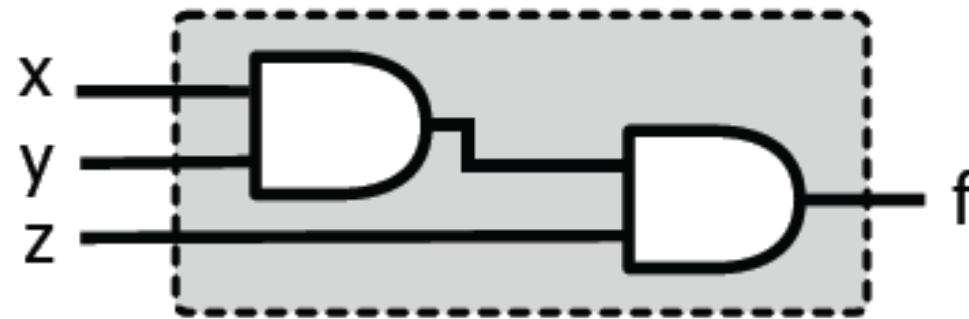


Complete Set of Logic Gates

- Functional completeness can also be applied to logic gates
- A set of logic gates that can implement any Boolean function is called a complete set of logic gates
 - {AND, OR, NOT}
 - {NAND} or {NOR}
- A universal gate is a gate that can implement any Boolean function without need to use any other gate type
 - {NAND} or {NOR}
 - Implementation requires fewer transistors and is faster than that of AND or OR gates
 - Logic designers prefer to use NAND or NOR gate

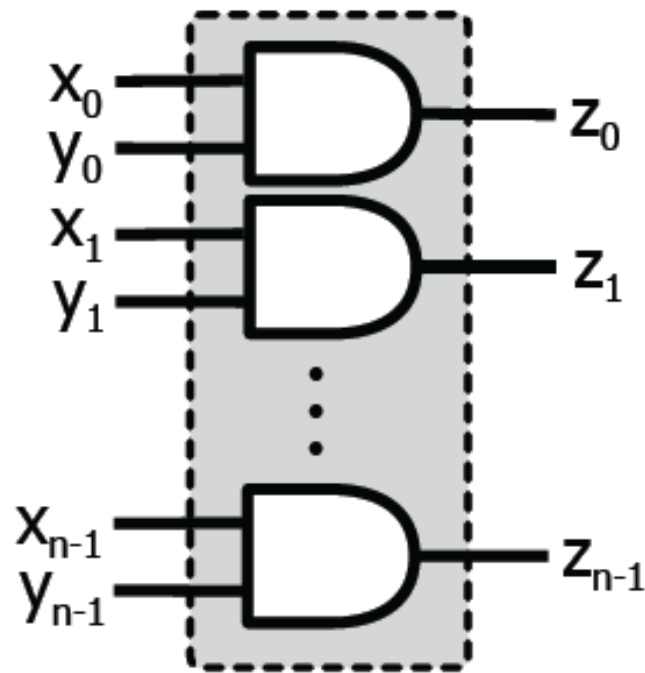
Multi-input Gates

- Multi-input gates can also be made by combining gates of the same type with less inputs



Multi-bit Logic Gates

- A multi-bit (n-bit) logic gate with a bit-wise Boolean operation is implemented by an array of n gates each operating separately on each bit position of the operands



Combinational Logic vs. Sequential Logic

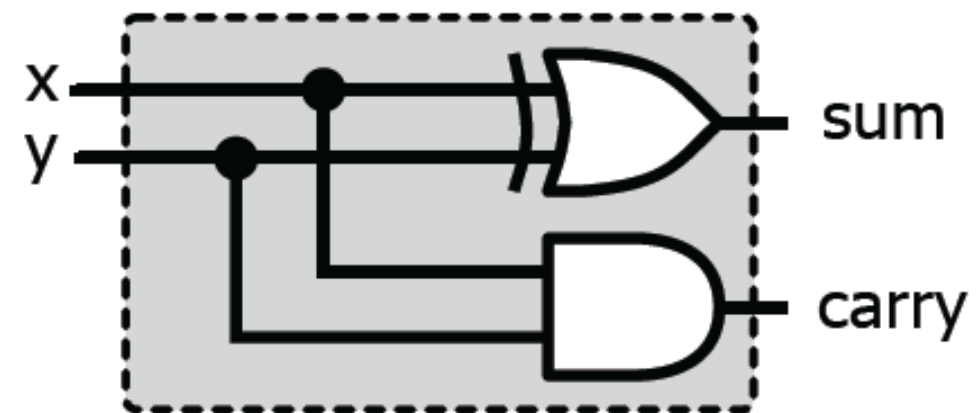
- The outputs of a combinational logic circuit
 - Totally dependent on the current input values and determined by combining the input values using Boolean operations
- The outputs of a sequential logic circuit
 - Depend not only on the current input values but also on the past inputs
 - Logic gates + memory
 - Outputs are a function of the current input values and the data stored in memory
 - States

Half Adder

- Adds two one-bit binary numbers x and y
- Two outputs: sum and carry

$$\begin{aligned}\text{sum} &= x \oplus y \\ \text{carry} &= x \wedge y\end{aligned}$$

x	y	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

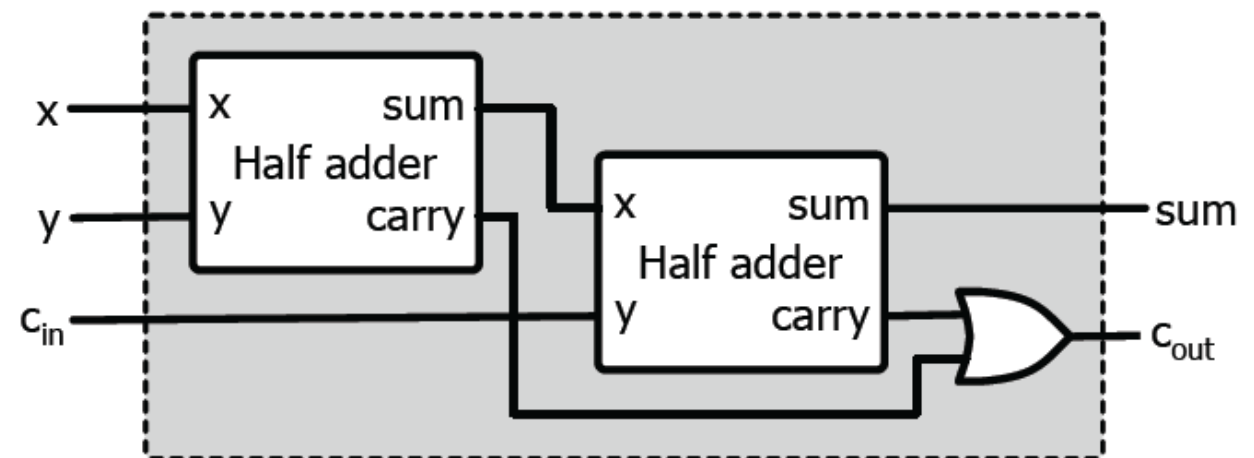
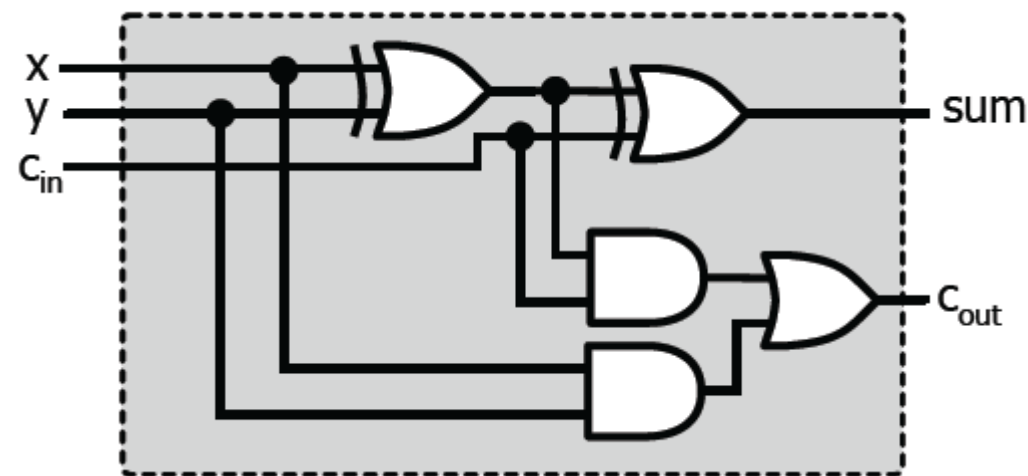


Full Adder

- Adds three one-bit binary numbers x, y, and a carry (c_{in}) coming in
- Two outputs: sum and carry (c_{out})

$$\begin{aligned}\text{sum} &= (x \oplus y) \oplus c_{in} \\ \text{carry} &= (x \wedge y) \vee (c_{in} \wedge (x \oplus y))\end{aligned}$$

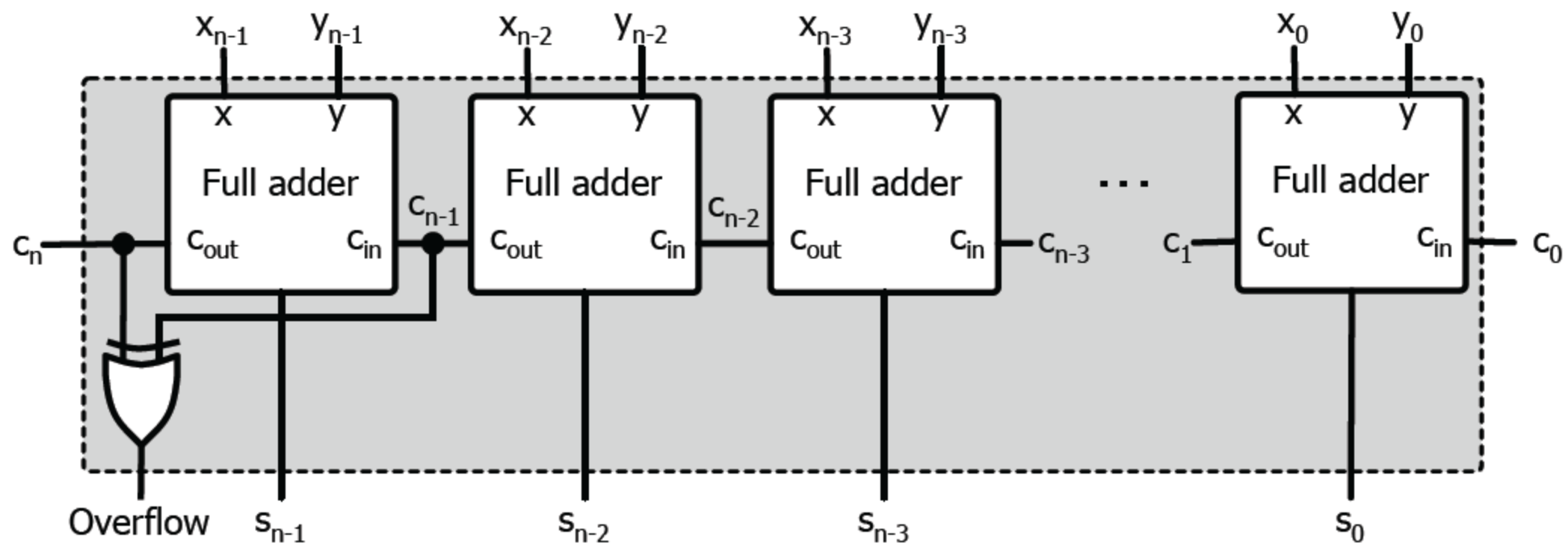
x	y	c_{in}	sum	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Ripple Carry Adder

- We can implement an n-bit binary adder by cascading n full adders
 - c_{out} of the previous full adder is connected to c_{in} of the next full adder
 - outputs are sum and carry (c_n) from the MSB
- For two's complement representation,

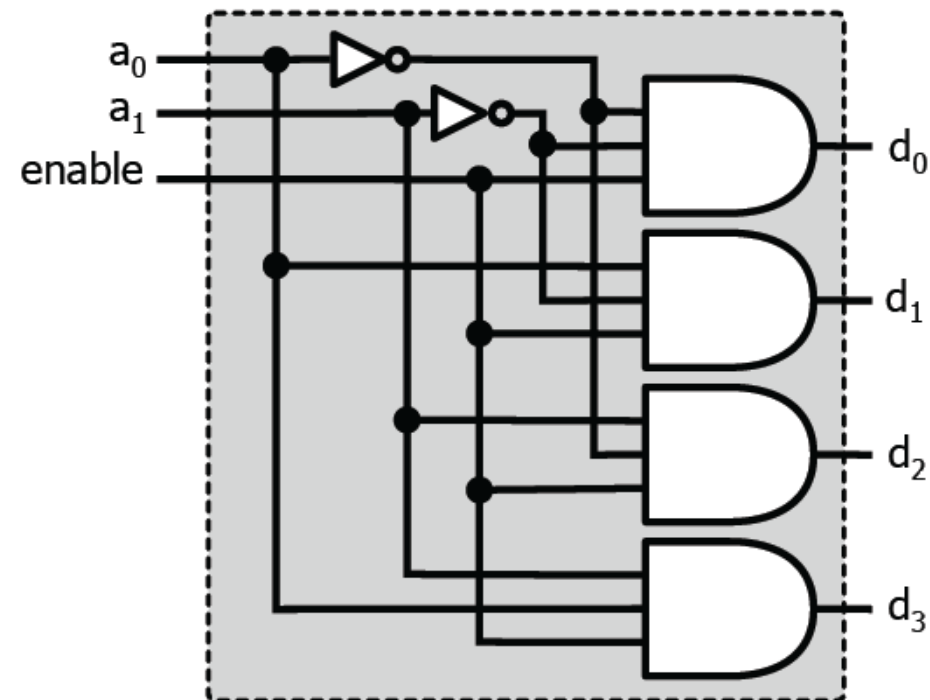
$$\text{Overflow} = c_n \oplus c_{n-1}$$



Decoder

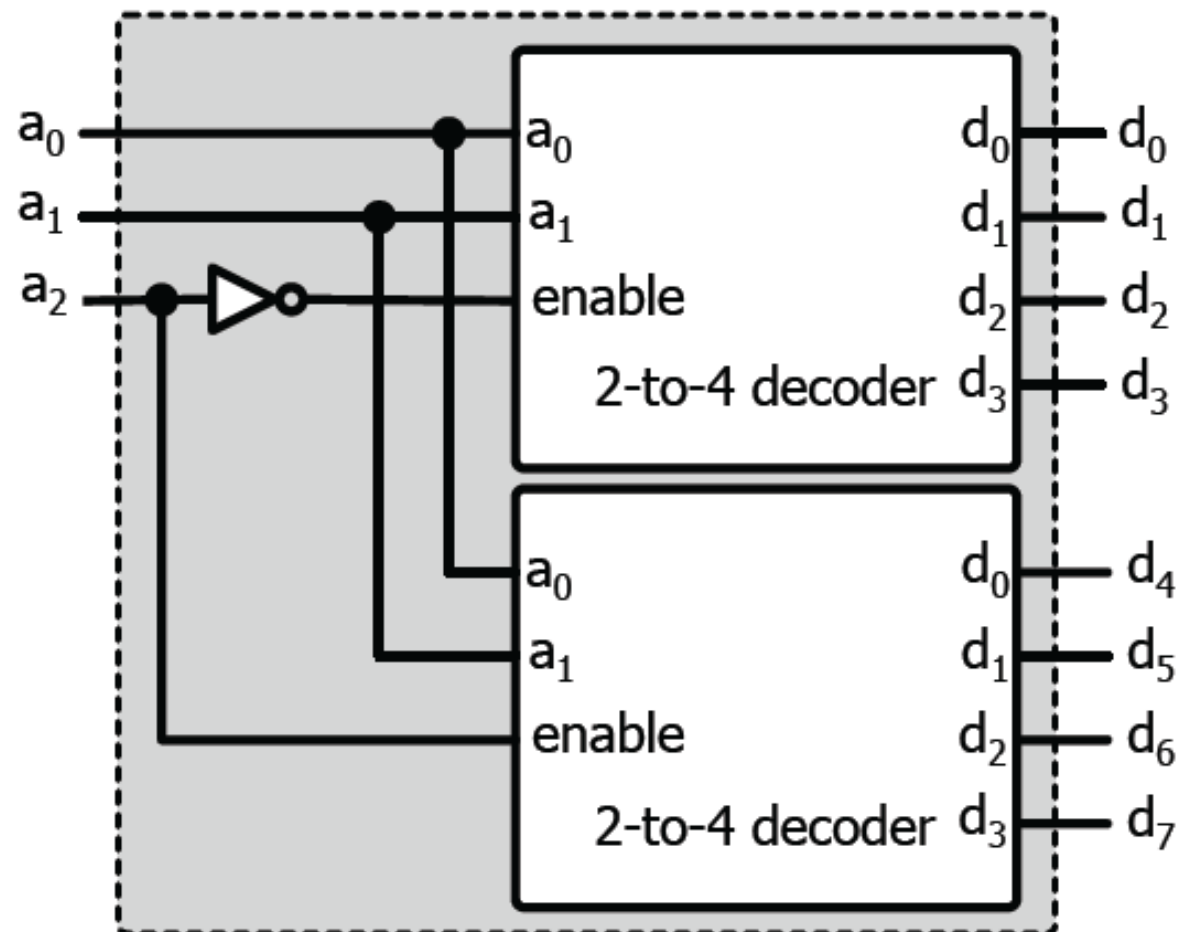
- Also called demultiplexer
- Converts binary information from the n coded inputs to a maximum of 2^n unique outputs
- 2-to-4 decoder, 3-to-8 decoder, 4-to-16 decoder, etc.
- Often has an enable input
 - When the enable input is 1, the outputs of the decoder are enabled
 - Otherwise, all the outputs are 0

enable	a_1	a_0	d_3	d_2	d_1	d_0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0
0	X	X	0	0	0	0



Combining Decoders

- We can build a 3-to-8 decoder by combining two 2-to-4 decoders each with an enable input



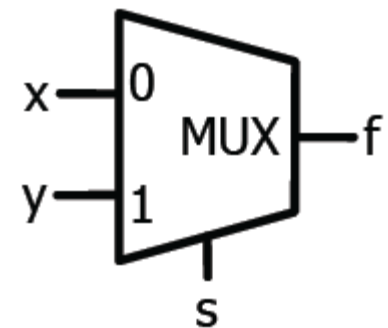
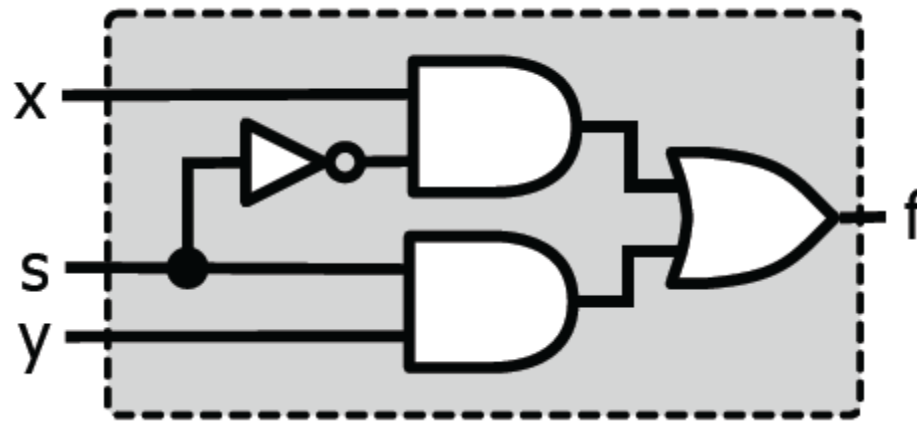
Multiplexer

- Also known as a selector
- A digital switch that connects data from one of n sources to its output
- MUX is a shorthand for multiplexer

x	y	s	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

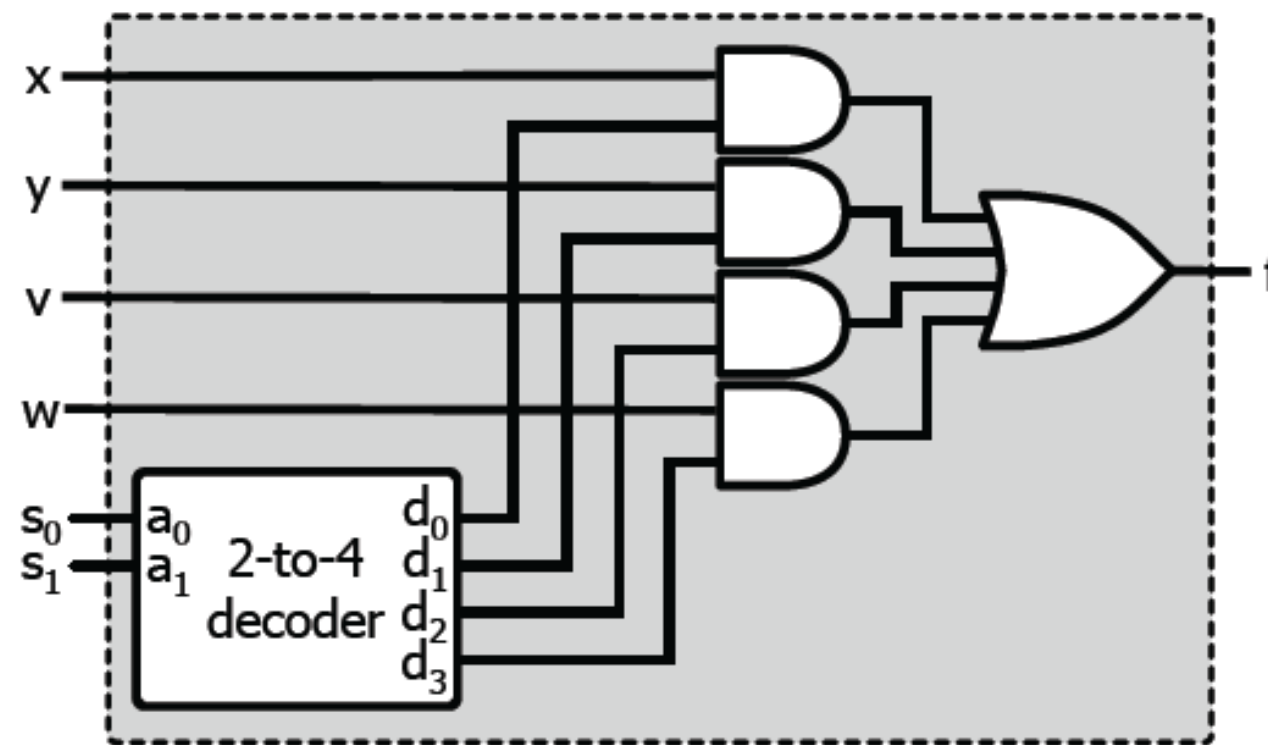
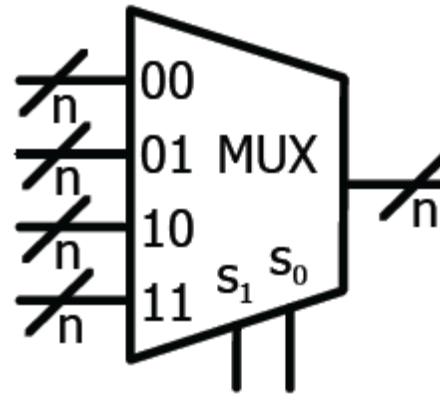
s	f
0	x
1	y

$$f(x, y, s) = (x \wedge s') \vee (y \wedge s)$$



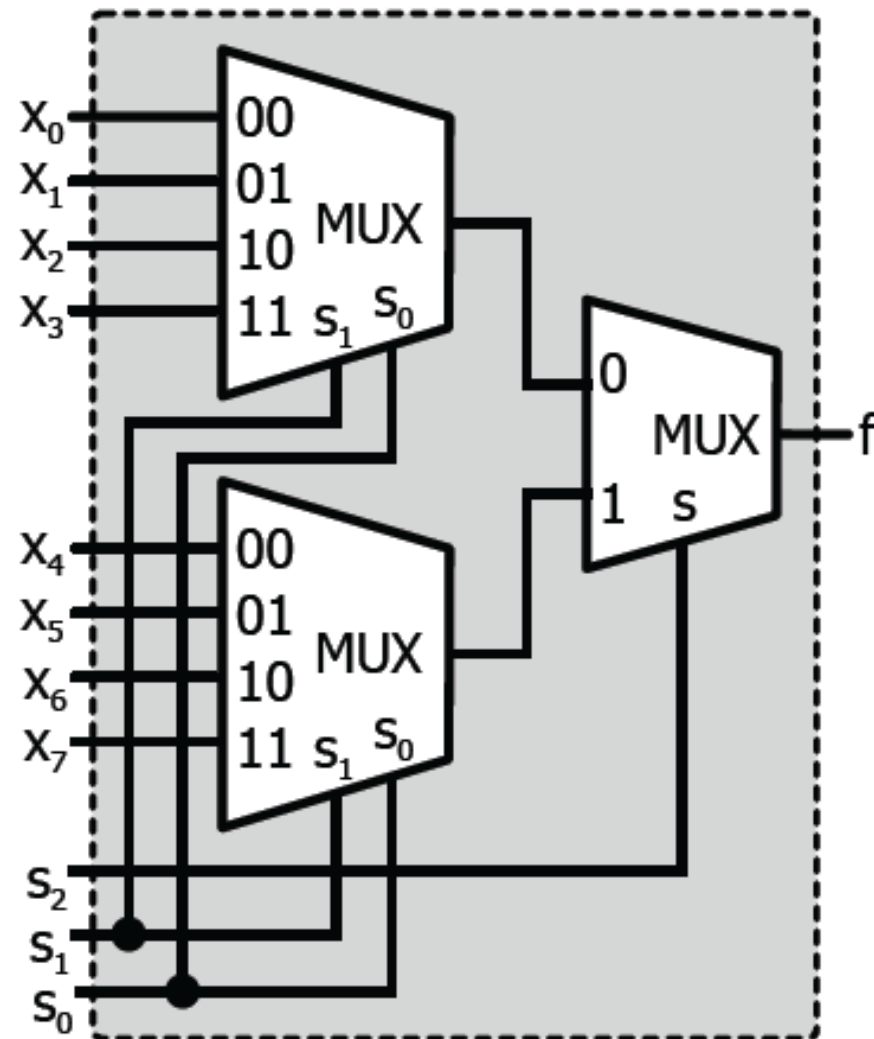
4-to-1 MUX

s_1	s_0	f
0	0	x
0	1	y
1	0	v
1	1	w



Combining MUXes

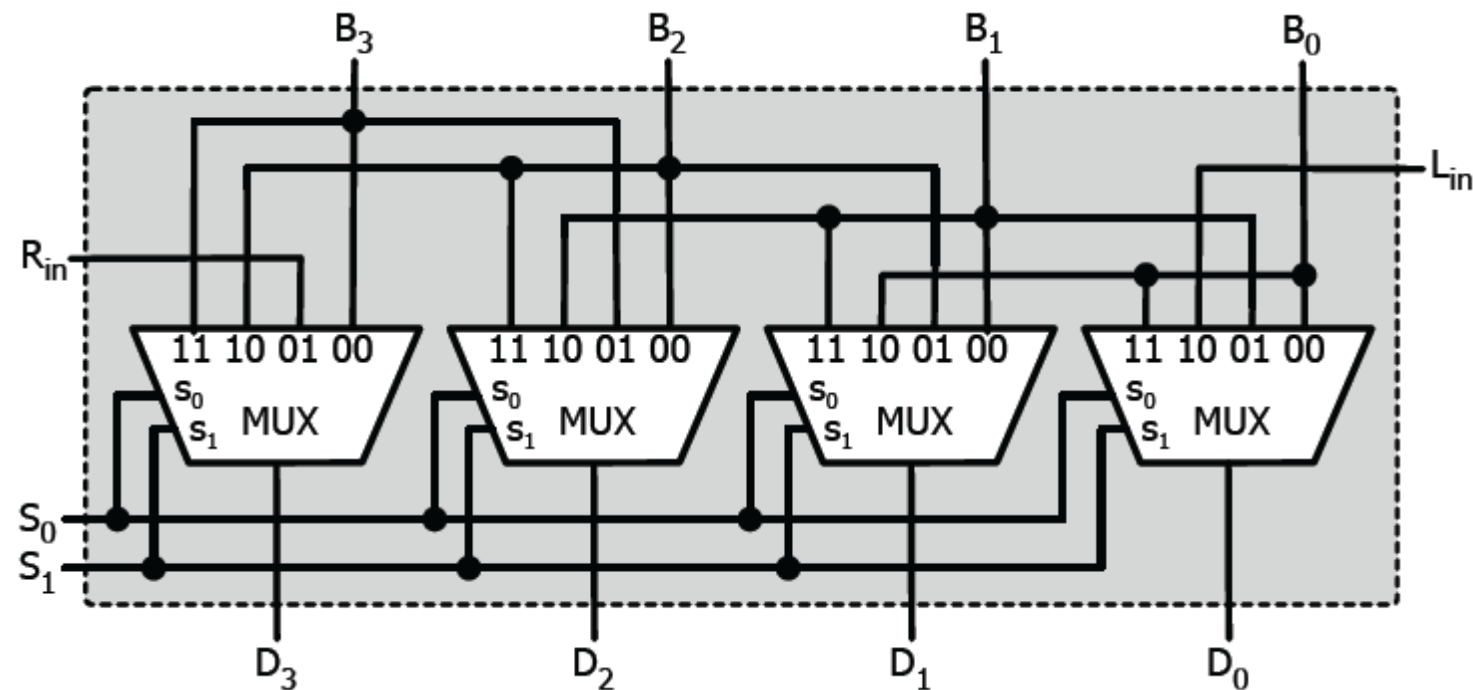
- A larger MUX can be constructed by combining smaller MUXes together



Shifter

- $S_1 = 0$ and $S_0 = 1$
 - One-bit right shift
 - Arithmetic right shift if B_3 is connected to R_{in}
 - If R_{in} is set to 0, one-bit logical right shift
- $S_1 = 1$, $S_0 = 0$, and $L_{in} = 0$
 - One-bit left shift

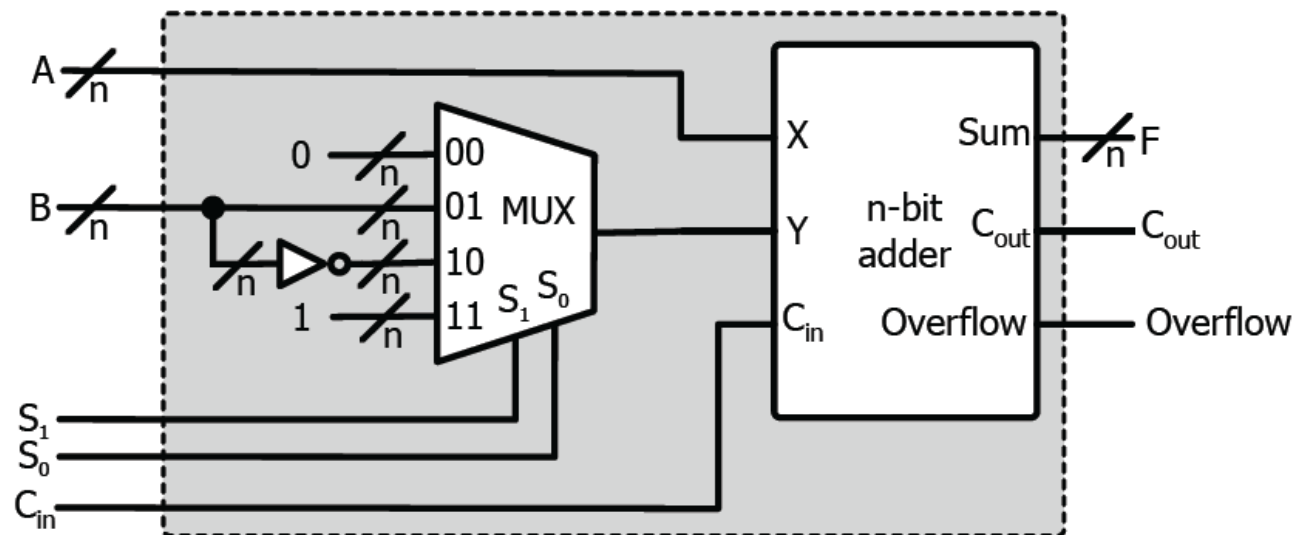
S_1	S_0	D_3	D_2	D_1	D_0
0	0	B_3	B_2	B_1	B_0
0	1	R_{in}	B_3	B_2	B_1
1	0	B_2	B_1	B_0	L_{in}
1	1	B_3	B_2	B_1	B_0



Arithmetic Unit

- Integer addition and subtraction
- Two's complement representation

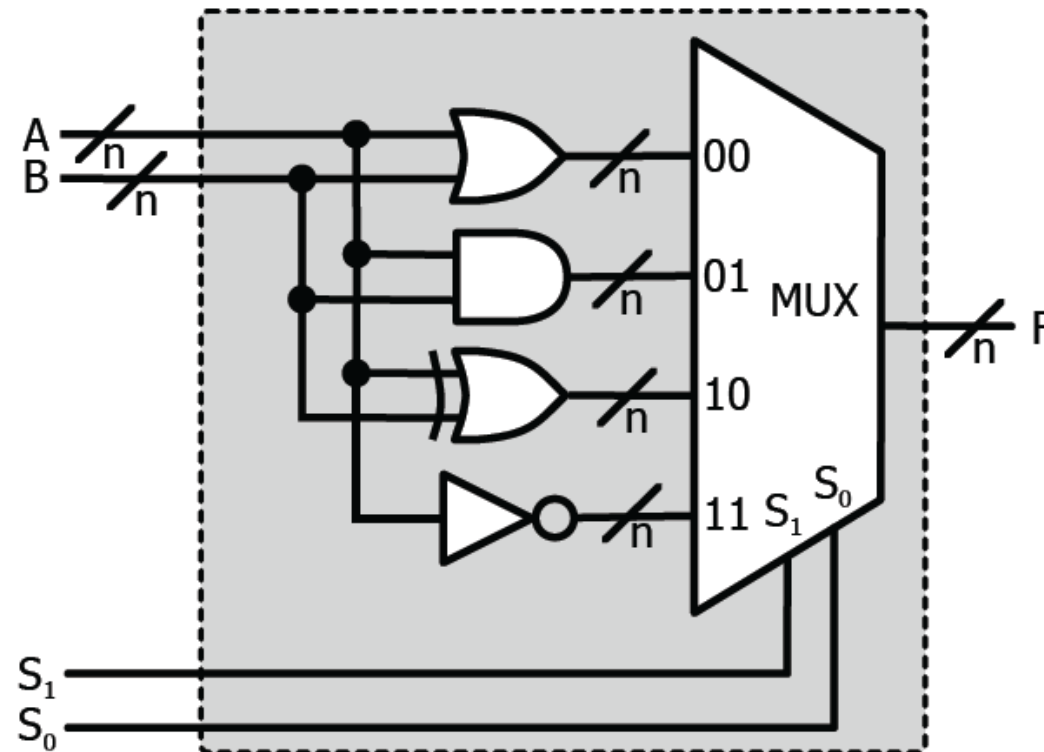
S_1	S_0	Y	$C_{in} = 0$	$C_{in} = 1$
0	0	$00\cdots 0$	$F = A$	$F = A + 1$
0	1	B	$F = A + B$	$F = A + B + 1$
1	0	B'	$F = A + B'$	$F = A + B' + 1$
1	1	$11\cdots 1$	$F = A - 1$	$F = A$



Logic Unit

- Bitwise OR, AND, XOR, and NOT

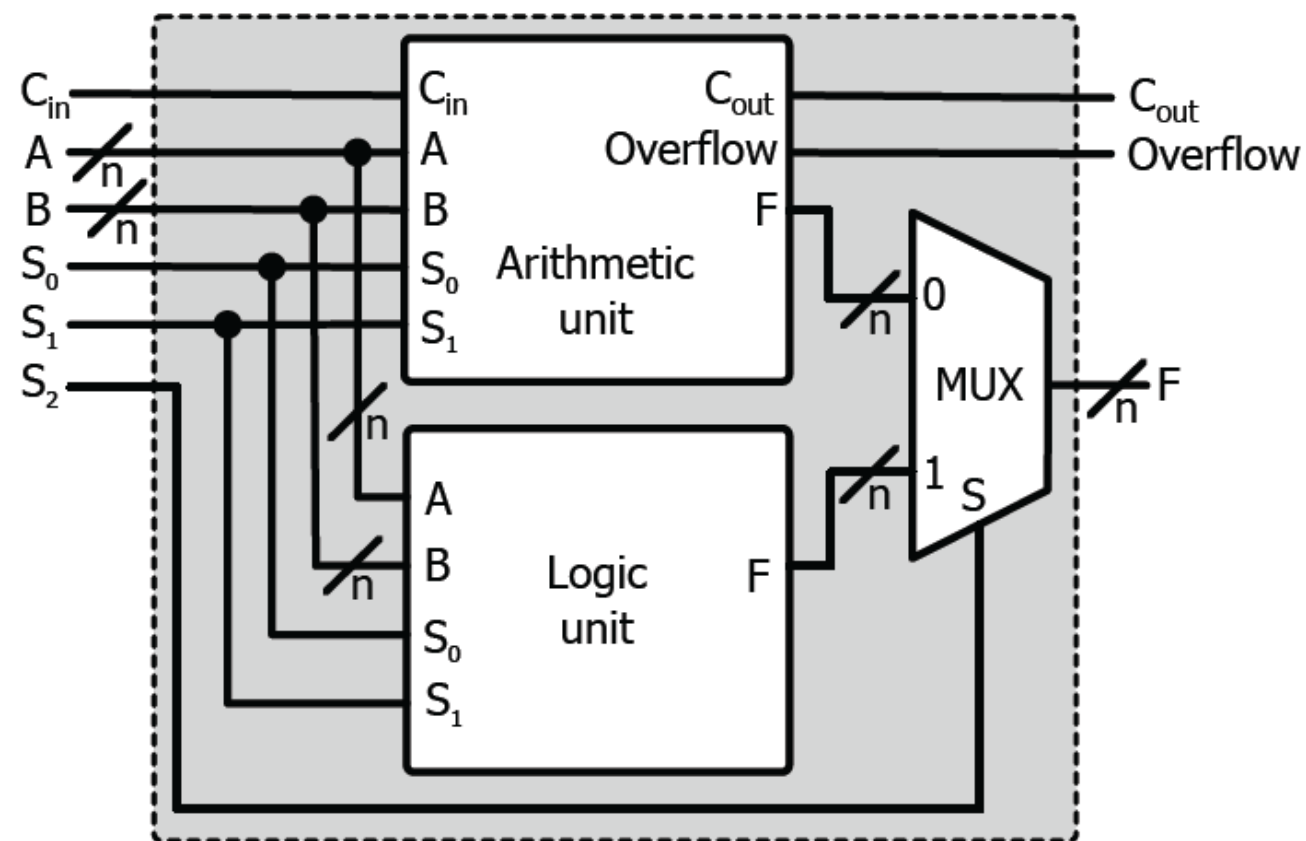
S_1	S_0	output	operation
0	0	$F = A \vee B$	OR
0	1	$F = A \wedge B$	AND
1	0	$F = A \oplus B$	XOR
1	1	$F = A'$	NOT



ALU

- Arithmetic unit + logic unit

S ₂	S ₁	S ₀	C _{in}	Operation
0	0	0	0	$F = A$
0	0	0	1	$F = A + 1$
0	0	1	0	$F = A + B$
0	0	1	1	$F = A + B + 1$
0	1	0	0	$F = A + B'$
0	1	0	1	$F = A + B' + 1$
0	1	1	0	$F = A - 1$
0	1	1	1	$F = A$
1	0	0	X	$F = A \vee B$
1	0	1	X	$F = A \wedge B$
1	1	0	X	$F = A \oplus B$
1	1	1	X	$F = A'$



ALU in a Broader Sense

- ALU + shifter

FS ₄	FS ₃	FS ₂	FS ₁	FS ₀	Operation
0	0	0	0	0	$F = A$
0	0	0	0	1	$F = A + 1$
0	0	0	1	0	$F = A + B$
0	0	0	1	1	$F = A + B + 1$
0	0	1	0	0	$F = A + B'$
0	0	1	0	1	$F = A + B' + 1$
0	0	1	1	0	$F = A - 1$
0	0	1	1	1	$F = A$
0	1	0	0	X	$F = A \vee B$
0	1	0	1	X	$F = A \wedge B$
0	1	1	0	X	$F = A \oplus B$
0	1	1	1	X	$F = A'$
1	0	0	X	X	$F = B$
1	0	1	0	X	$F = \text{logical shift right } B$
1	0	1	1	X	$F = \text{arithmetic shift right } B$
1	1	0	X	X	$F = \text{shift left } B$
1	1	1	X	X	$F = B$

Arithmetic

Logic unit

Shifter

