

CPS 104
Computer Organization and Programming
Lecture 10: Boolean Algebra & gates.

Robert Wagner

Overview of Today's Lecture:

- **Truth tables, Boolean functions, Gates and Circuits**
- **Karnaugh maps for simplifying Boolean equations**
- **Examples: 2-1 MUX, Full Adder**

Read Appendix B

How Computers Really Work

- **Computers manipulate sequences of 0's and 1's in ways that simulate ordinary arithmetic, and comparisons**
- **Computer hardware uses “Boolean operations”, also called “Logic functions” to do this**
- **These mathematical functions operate on 2 values, interpreted as True (T) and False (F), or as 0 and 1**
- **Basic Boolean functions can be implemented by circuits whose inputs and outputs are voltages representing 0 and 1.**
- **The voltages involved can be observed -- 1 can light a lamp, while 0 leaves the lamp unlit. Or, the Boolean value on a wire can set a displayed pixel “on” or “off”.**

Boolean Functions and Gates

- Boolean functions have arguments that each take one of two values (called either $\{T,F\}$ or $\{0,1\}$) . Each returns one ($\{T,F\}$ or $\{0,1\}$) value, or a vector of such values.
- Boolean functions can always be represented by a table called: “Truth Table” which gives the answer, for every input combination.
- Example: $F: \{0,1\}^3 \rightarrow \{0,1\}^2$

a	b	c	f_1	f_2
0	0	0	0	1
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Boolean Functions and Gates (Cont.)

- **Examples: Boolean functions: NOT, AND, OR, XOR, ...**

a	NOT(a)
0	1
1	0

a	b	AND(a,b)
0	0	0
0	1	0
1	0	0
1	1	1

a	b	OR(a,b)
0	0	0
0	1	1
1	0	1
1	1	1

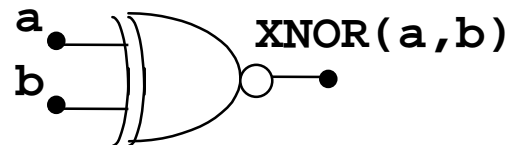
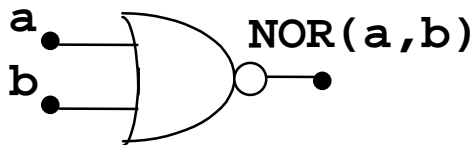
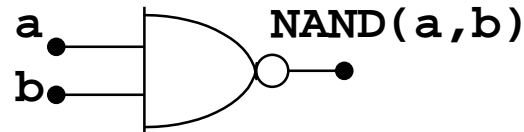
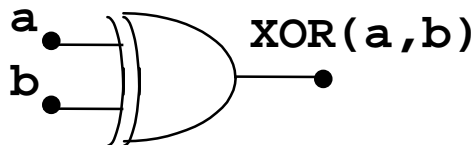
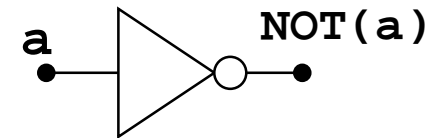
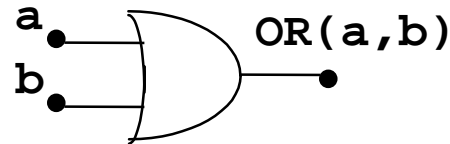
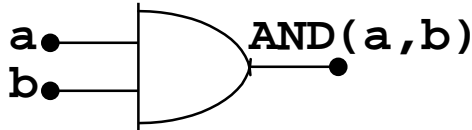
a	b	XOR(a,b)
0	0	0
0	1	1
1	0	1
1	1	0

a	b	XNOR(a,b)
0	0	1
0	1	0
1	0	0
1	1	1

a	b	NOR(a,b)
0	0	1
0	1	0
1	0	0
1	1	0

Boolean Functions and Gates (Cont.)

- **Gates are electronic devices that implement simple Boolean functions:**
- **Examples (gate shape indicates the function the gate computes):**



Boolean Expressions

- **Boolean algebra notation: Use:**
* for AND, + for OR, ~ for NOT.
- **Using the above notation one could write Boolean expressions for functions:**

Example:

$$F(A, B, C) = (A * B) + (\sim A * C)$$

Boolean Functions and Expressions

- One can evaluate the Boolean expression with all possible argument values to construct a truth table.

Example:

$$F(A, B, C) = (A * B) + (\sim A * C)$$

A	B	C	F
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

Boolean functions simplification

- Boolean expressions could be simplified by using the following rules:

$$* A * A = A \quad A + A = A \quad \sim \sim A = A$$

$$* A * 0 = 0 \quad A + 1 = 1$$

$$* A * 1 = A \quad A + 0 = A$$

$$* A * \sim A = 0 \quad A + \sim A = 1$$

$$* A * B = B * A \quad A + B = B + A$$

$$* A * (B + C) = A * B + A * C$$

$$A + (B * C) = (A + B) * (A + C)$$

$$* \sim (A + B) = \sim A * \sim B$$

$$\sim (A * B) = \sim A + \sim B$$

Boolean Functions simplification

- Example:

a	b	c	f ₁	f ₂
0	0	0	0	1
0	0	1	1	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	0
1	1	0	0	1
1	1	1	1	1

$$f_1 = \sim a^* \sim b^* c + \sim a^* b^* c + a^* \sim b^* c + a^* b^* c$$

$$f_2 = \sim a^* \sim b^* \sim c + \sim a^* \sim b^* c + a^* b^* \sim c + a^* b^* c$$

Boolean Function Simplification

$$\begin{aligned}f_1 &= \sim a^* \sim b^* c + \sim a^* b^* c + a^* \sim b^* c + a^* b^* c \\&= \sim a^* (\sim b^* c + b^* c) + a^* (\sim b^* c + b^* c) \\&= \sim a^* c^* (\sim b + b) + a^* c^* (\sim b + b) \\&= \sim a^* c + a^* c \\&= c^* (\sim a + a) \\&= c\end{aligned}$$

$$\begin{aligned}f_2 &= \sim a^* \sim b^* \sim c + \sim a^* \sim b^* c + a^* b^* \sim c + a^* b^* c \\&= \sim a^* (\sim b^* \sim c + \sim b^* c) + a^* (b^* \sim c + b^* c) \\&= \sim a^* \sim b^* (c + \sim c) + a^* b^* (\sim c + c) \\&= \sim a^* \sim b + a^* b\end{aligned}$$

Karnaugh Maps

- **Graphical device for simplifying Boolean equations**
- **For 4-variable equation:**
 - * **Enter desired truth table in 4 by 4 matrix, whose rows are labeled with all combinations of values of 2 variables, and whose cols are labeled with other variable values**
 - Include “Don’t Care” entries as “x”
 - * **Row / Column order: One variable value changes between adjacent rows, and between adjacent columns**
 - * **Each rectangle containing only “T” or “x” entries, whose side-lengths are 1, 2, or 4 represents a simplified “term” -- identified by ANDing variables or their complements which are constant over cells in the rectangle.**
 - * **OR enough terms together to cover all “T” entries in table.**

Karnaugh Map Example

CD: AB	FF	FT	TT	TF	
FF	T	x		T	$\sim A^* \sim D$
FT	x		T	x	$B^* C$
TT			x	x	
TF					

$$\sim A^* \sim D + B^* C$$

Boolean Functions and Expressions

The Fundamental Theorem of Boolean Algebra:
Every Boolean function can be written in disjunctive normal form as an OR of ANDs (Sum-of-products) of its arguments or their complements.

“Proof:” Write the truth table, construct sum-of-product from the table.

Example:

a	b	XNOR(a,b)
0	0	1
0	1	0
1	0	0
1	1	1

$$\text{XNOR} = (\sim a * \sim b) + (a * b)$$

Boolean Functions and Boolean Expressions

- Example-2:

a	b	c	f ₁	f ₂
0	0	0	0	1
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	1	0	0	1
1	1	1	1	1

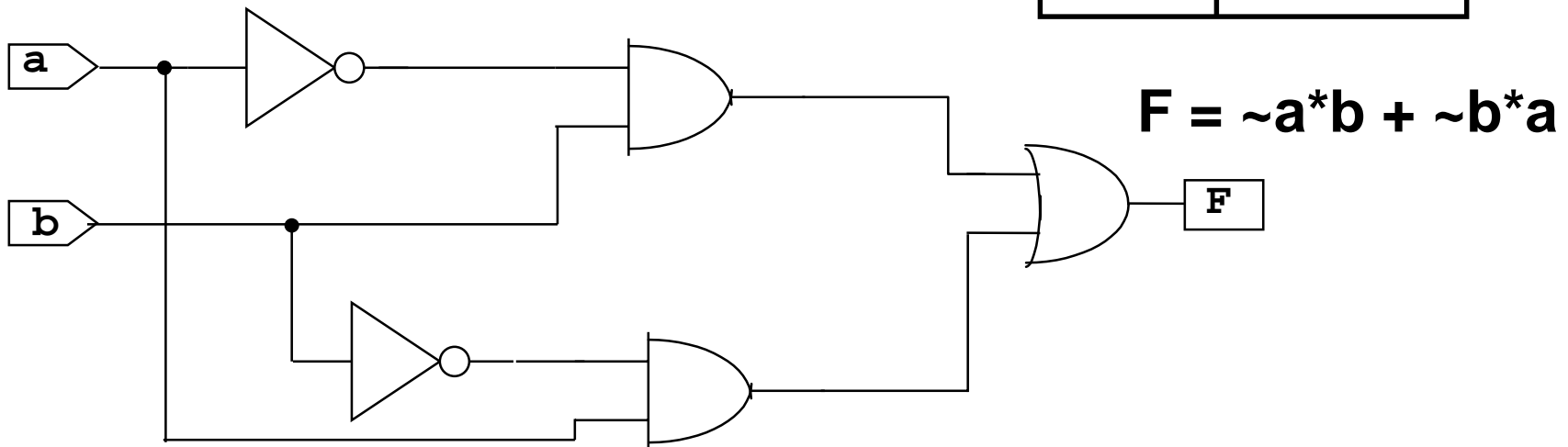
$$f_1 = \sim a^* \sim b^* c + \sim a^* b^* \sim c + a^* \sim b^* \sim c + a^* b^* c$$

$$f_2 = \sim a^* \sim b^* \sim c + \sim a^* \sim b^* c + a^* b^* \sim c + a^* b^* c$$

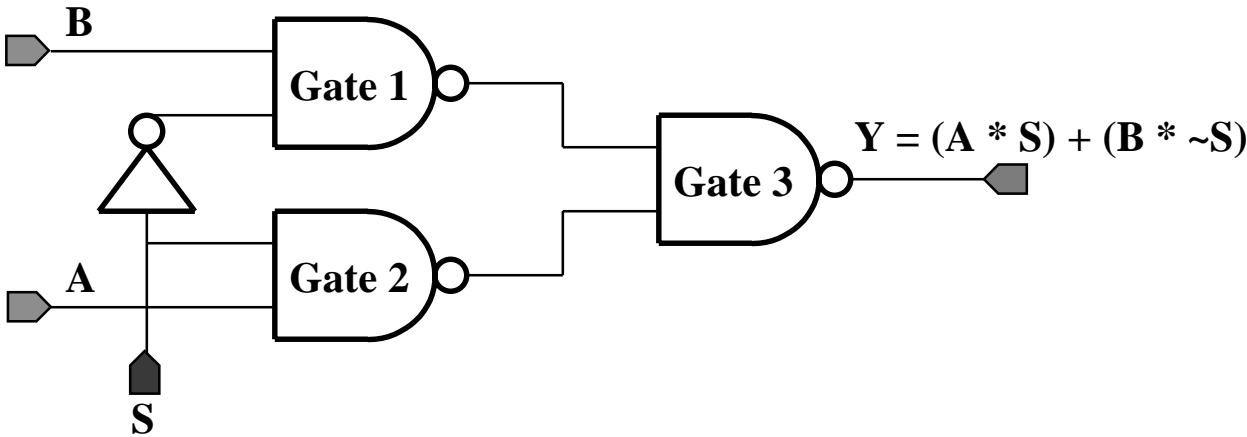
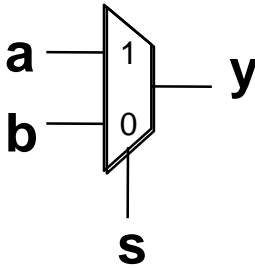
Boolean Functions, Gates and Circuits

- Circuits are made from a network of gates. (function compositions).
- Example:

a	b	XOR(a,b)
0	0	0
0	1	1
1	0	1
1	1	0

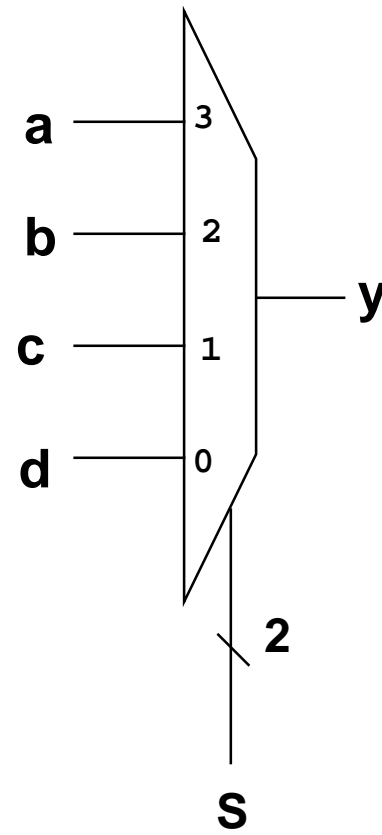
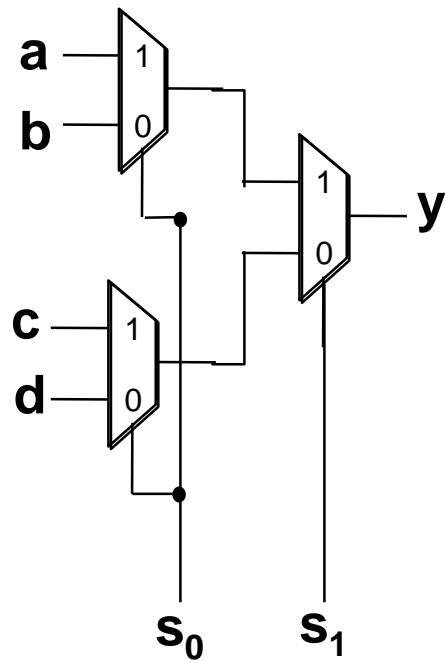


Circuit Example: 2x1 MUX

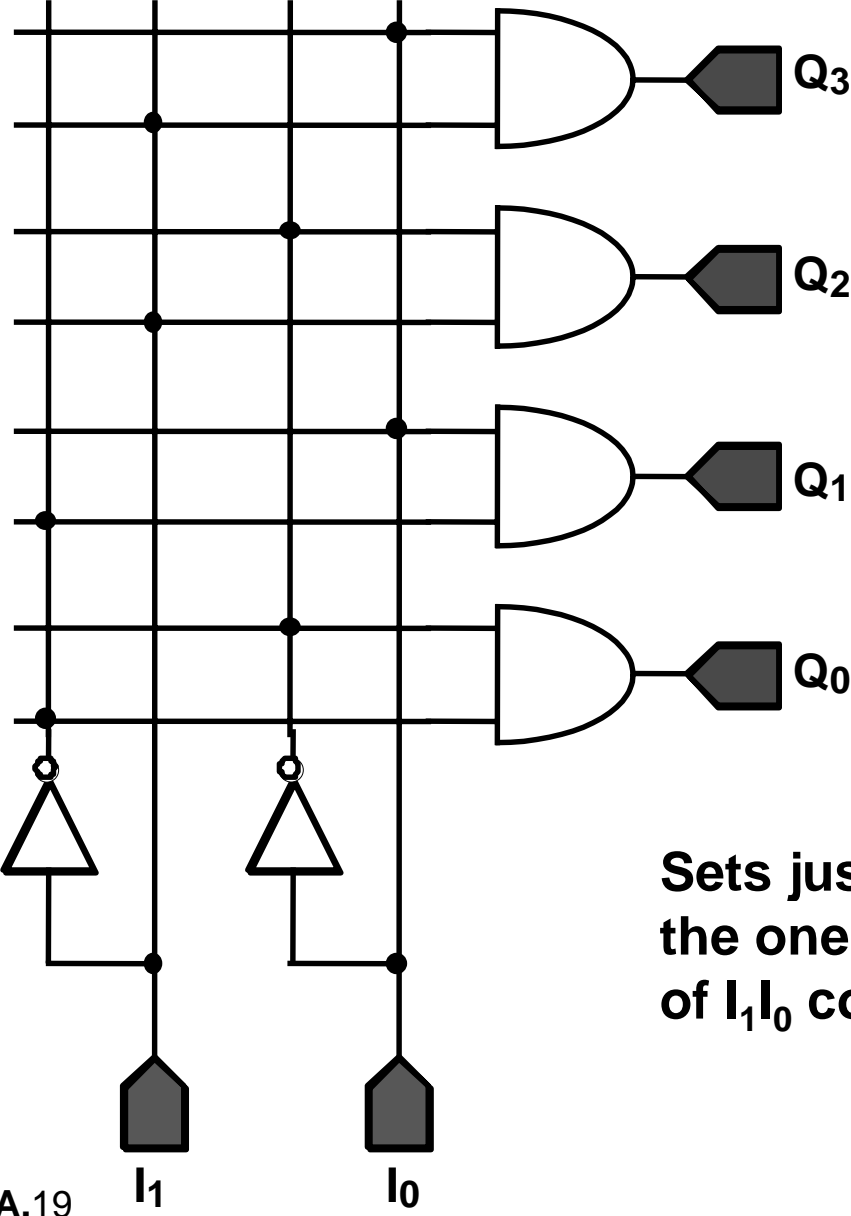


$$\sim(\sim A * \sim B) = \sim\sim A + \sim\sim B = A + B$$

Example 4x1 MUX



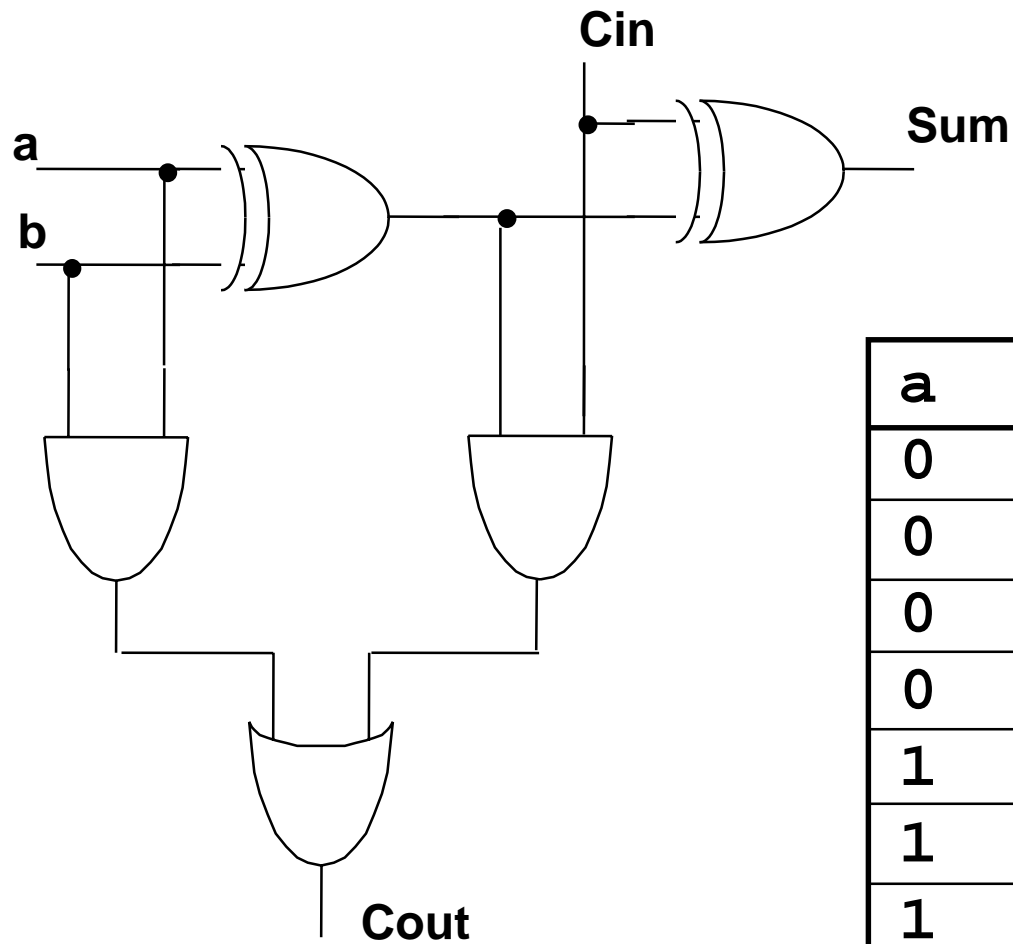
Circuit Example: Selector



I_1	I_0	Q_0	Q_1	Q_2	Q_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

Sets just ONE output True -- the one indexed by the value of I_1I_0 converted to decimal

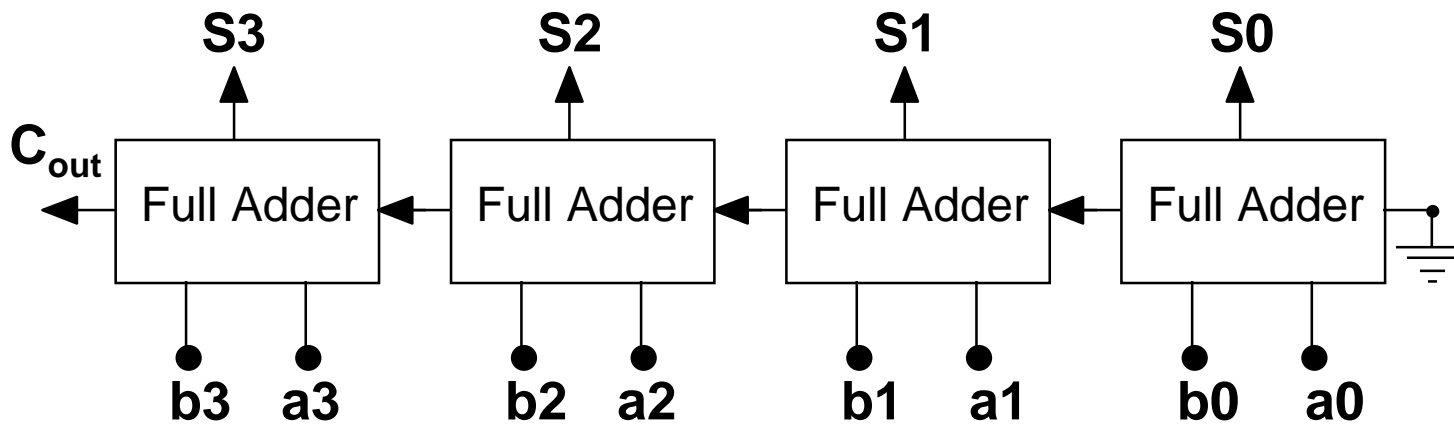
Full Adder



$$\begin{array}{r}
 01101100 \\
 01101101 \\
 +00101100 \\
 \hline
 10011001
 \end{array}$$

a	b	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

Example: 4-bit adder



Overflow Detection: examples

Example1:

$$\begin{array}{r} 01000000 \\ \swarrow \swarrow \swarrow \swarrow \swarrow \swarrow \swarrow \swarrow \\ 0110101_2 \quad (= 53_{10}) \\ +0101010_2 \quad (= 42_{10}) \\ \hline 1011111_2 \quad (= -33_{10}) \end{array}$$

Example2:

$$\begin{array}{r} 10000000 \\ \swarrow \swarrow \swarrow \swarrow \swarrow \swarrow \swarrow \swarrow \\ 1010101_2 \quad (= -43_{10}) \\ +1001010_2 \quad (= -54_{10}) \\ \hline 0011111_2 \quad (= 31_{10}) \end{array}$$

Example3:

$$\begin{array}{r} 11000000 \\ \swarrow \swarrow \swarrow \swarrow \swarrow \swarrow \swarrow \swarrow \\ 0110101_2 \quad (= 53_{10}) \\ +1101010_2 \quad (= -22_{10}) \\ \hline 0011111_2 \quad (= 31_{10}) \end{array}$$

Example4:

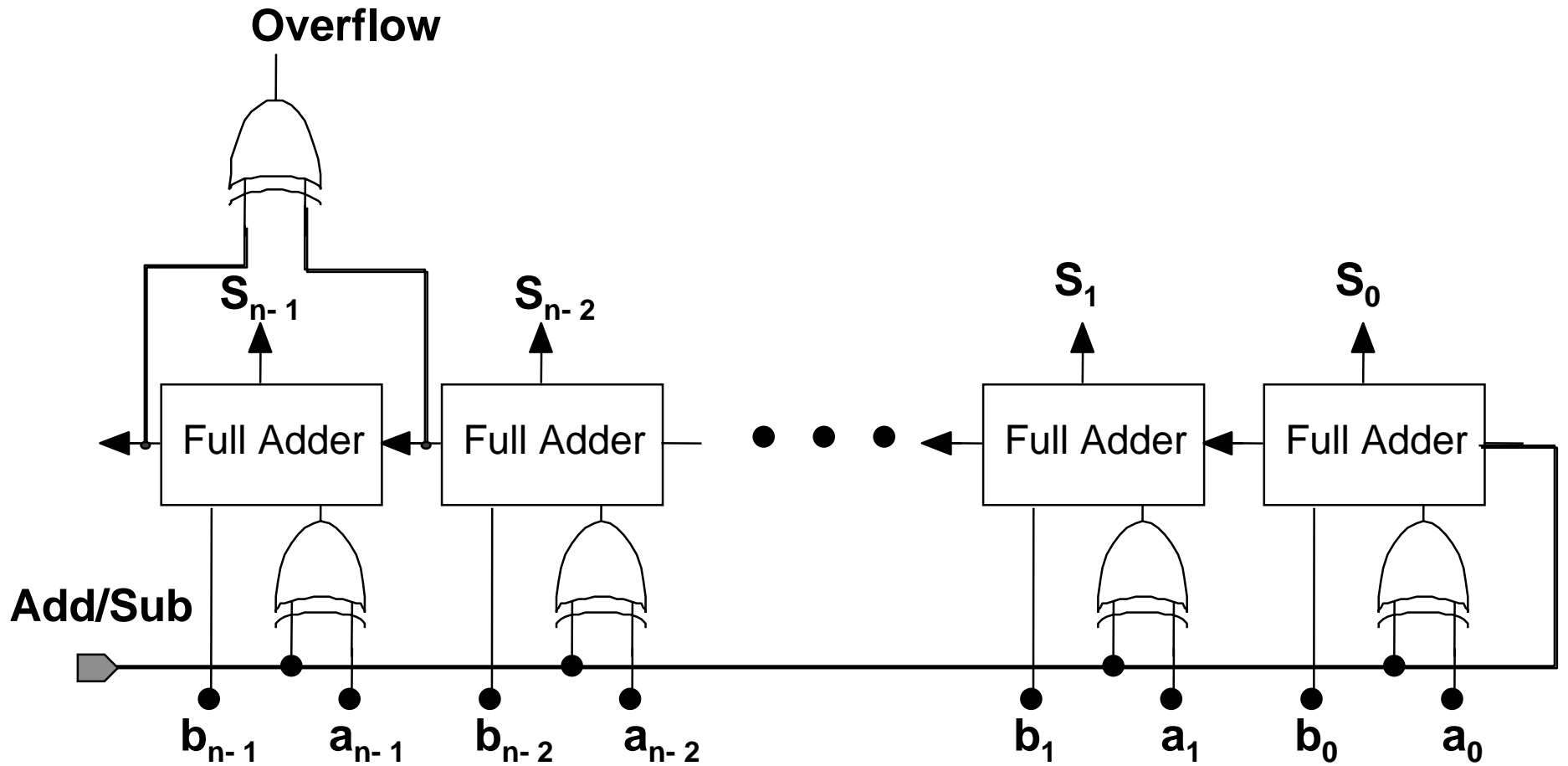
$$\begin{array}{r} 00000000 \\ \swarrow \swarrow \swarrow \swarrow \swarrow \swarrow \swarrow \swarrow \\ 0010101_2 \quad (= 21_{10}) \\ +0101010_2 \quad (= 42_{10}) \\ \hline 0111111_2 \quad (= 63_{10}) \end{array}$$

Overflow Detection and Carries

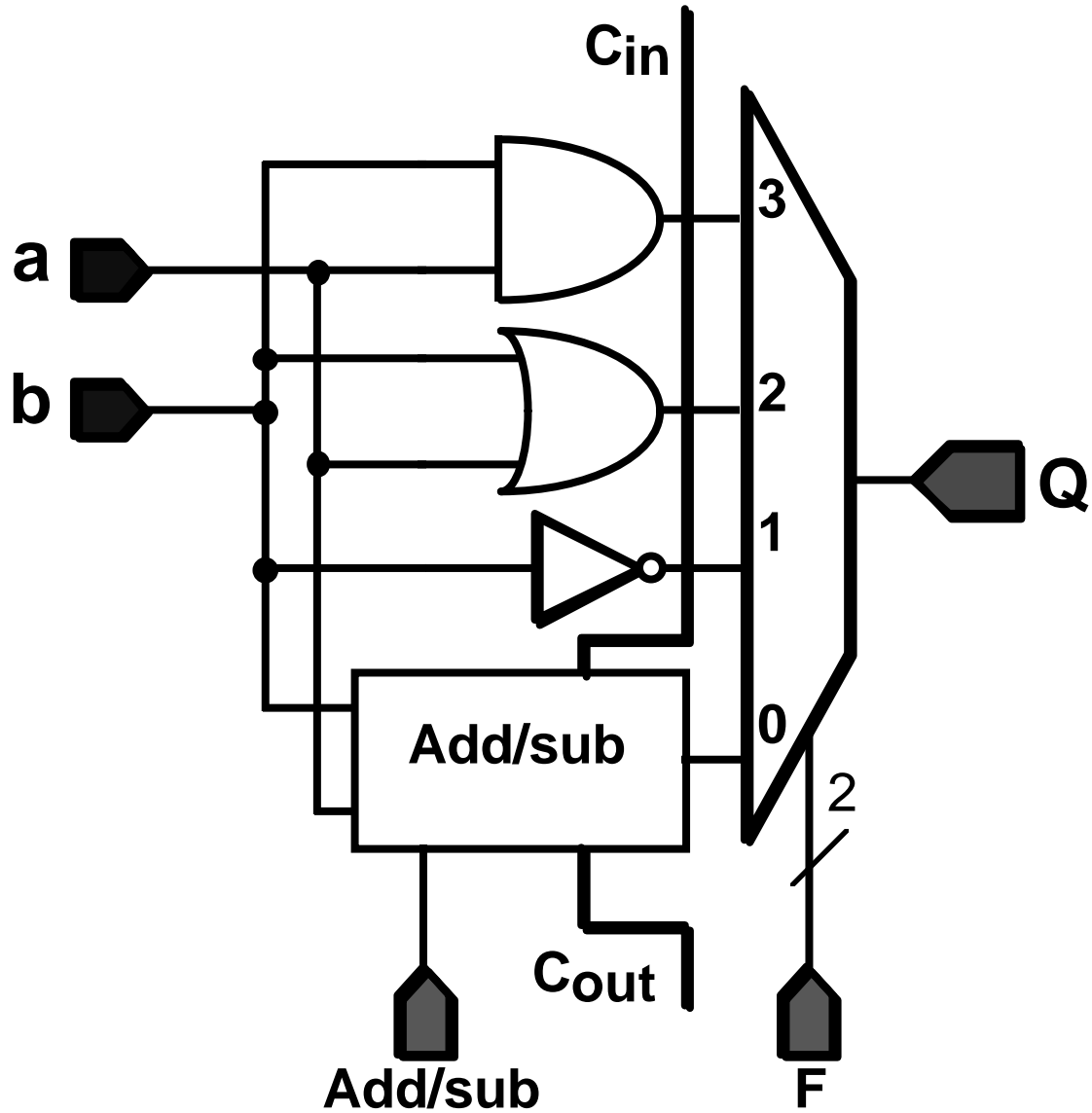
- In Addition and Subtraction operations, Overflow occurs when two number of same sign are added, and result has different sign.
- Must be a function of what happens in high-order full-adder

A_{31}	B_{31}	Cin_{31}	S_{31}	$Cout_{31}$	OVF	Cin= Cout
0	0	0	0	0	F	Y
0	0	1	1	0	T	N
0	1	0	1	0	F	Y
0	1	1	0	1	F	Y
1	0	0	1	0	F	Y
1	0	1	0	1	F	Y
1	1	0	0	1	T	N
1	1	1	1	1	F	Y

Add/Subtract With Overflow detection



MIPS ALU Slice



A	F	Q
0	0	$a + b$
1	0	$a - b$
-	1	NOT <i>b</i>
-	2	$a \text{ OR } b$
-	3	$a \text{ AND } b$

The MIPS ALU

