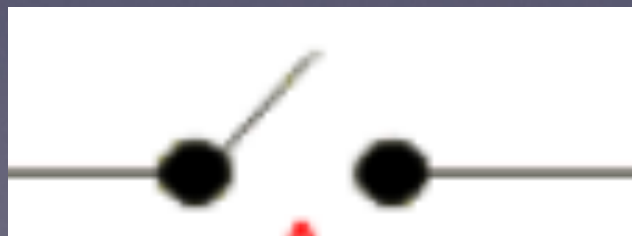
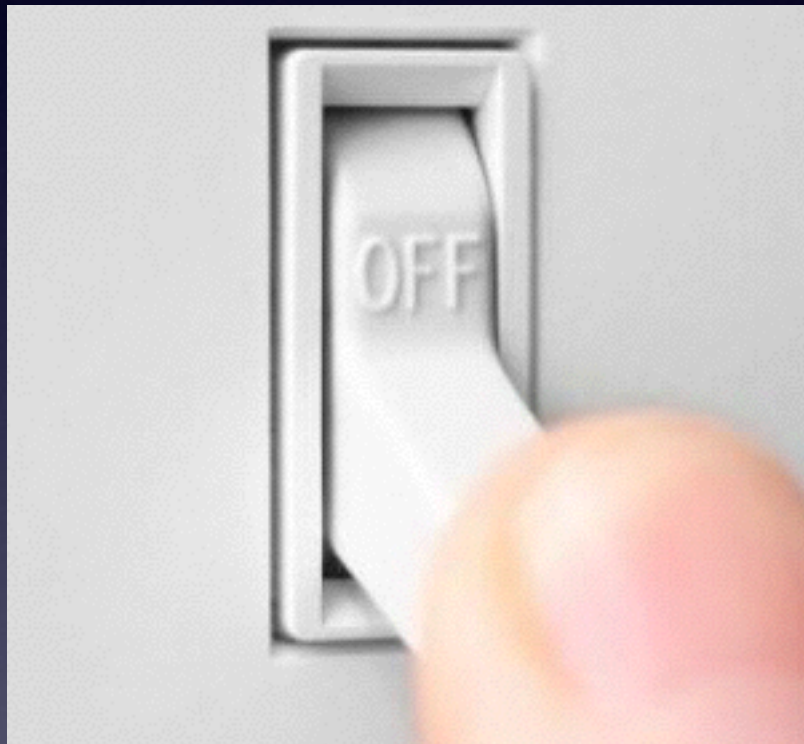


Combinational Logic

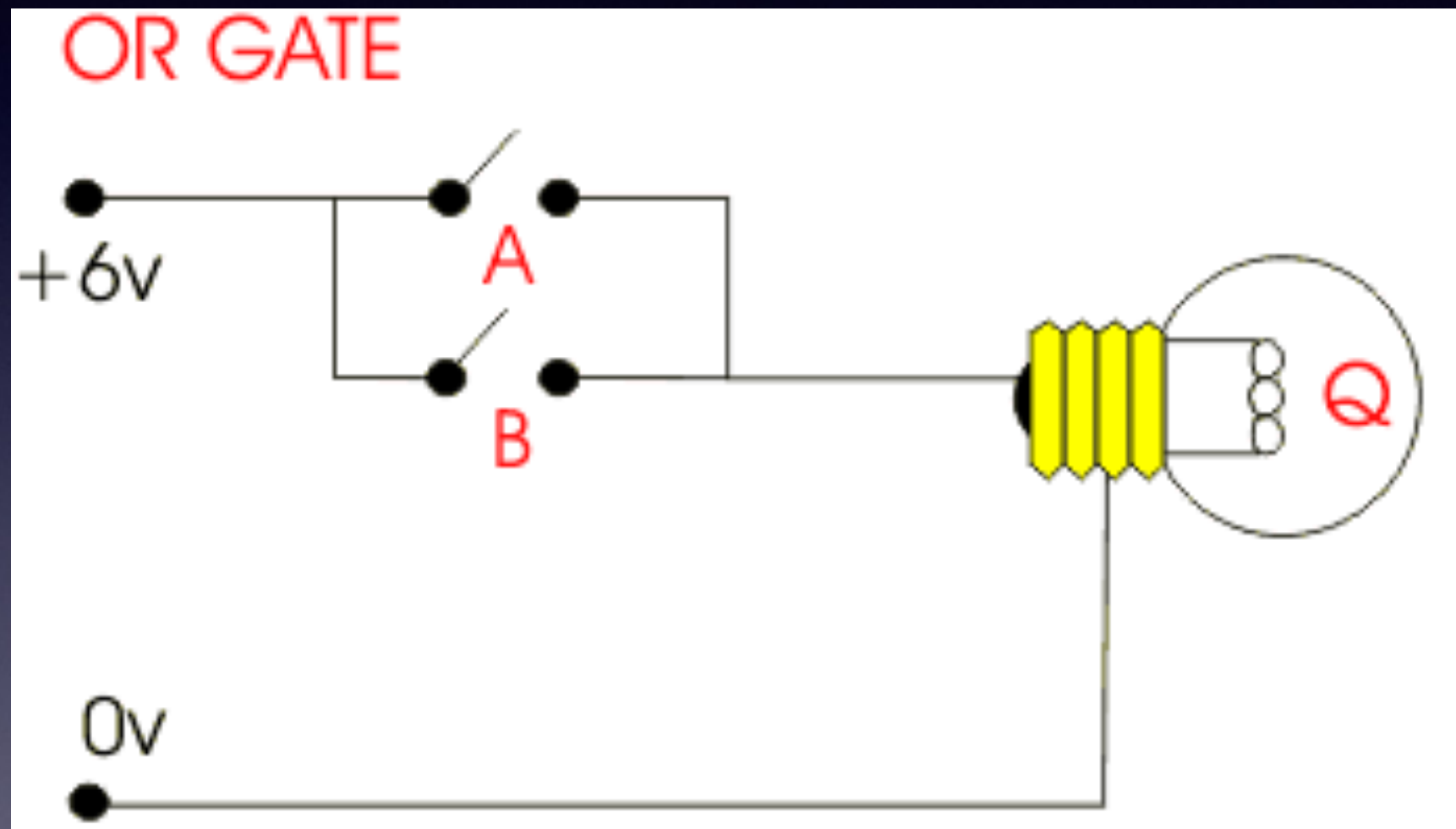
CS270

Graciously provided by Max Luttrell

switch



OR (either)

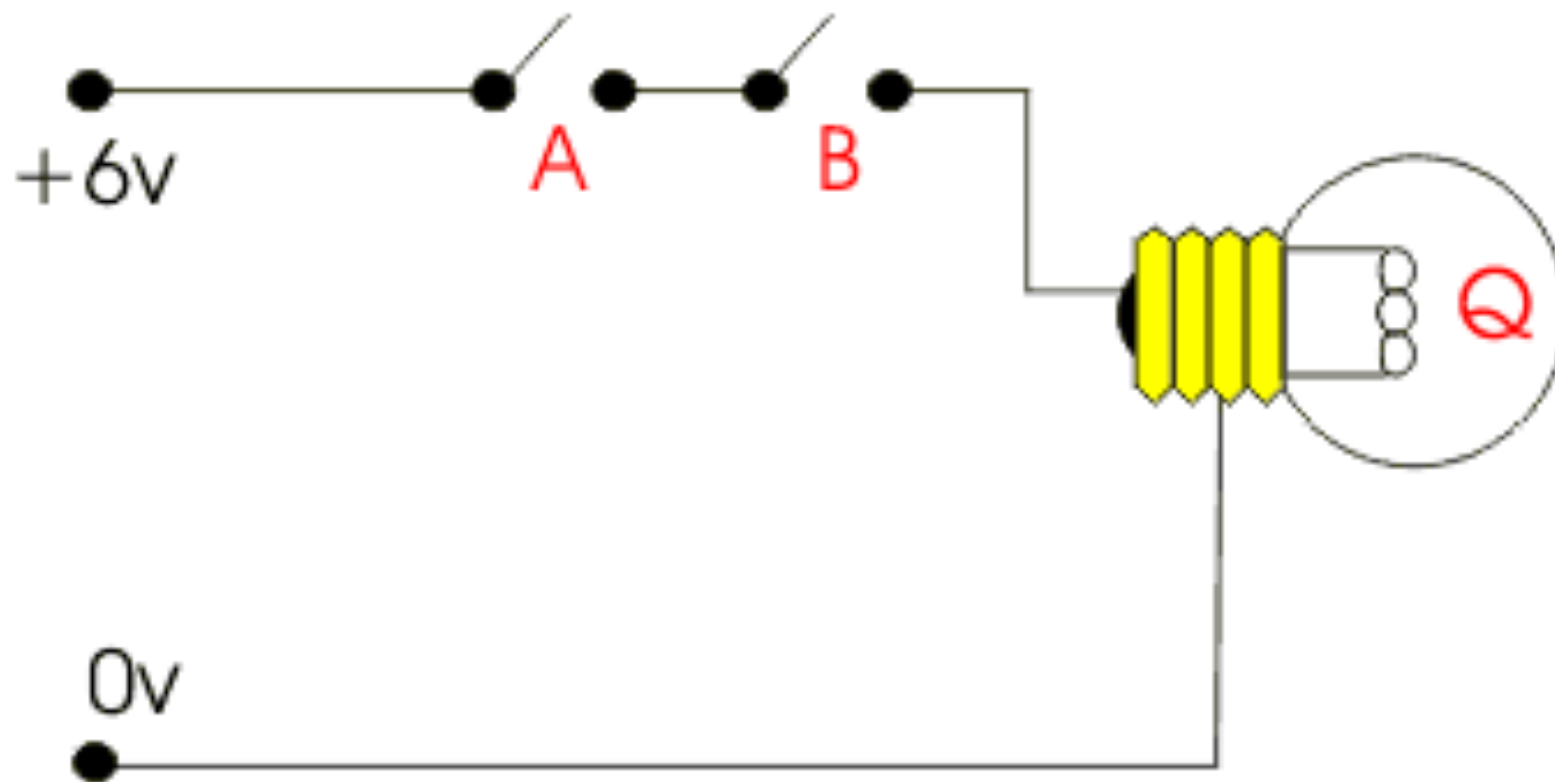


A	B	Light
OFF	OFF	
OFF	ON	
ON	OFF	
ON	ON	

For light to be on, we need a path from 6v to 0v

AND (both)

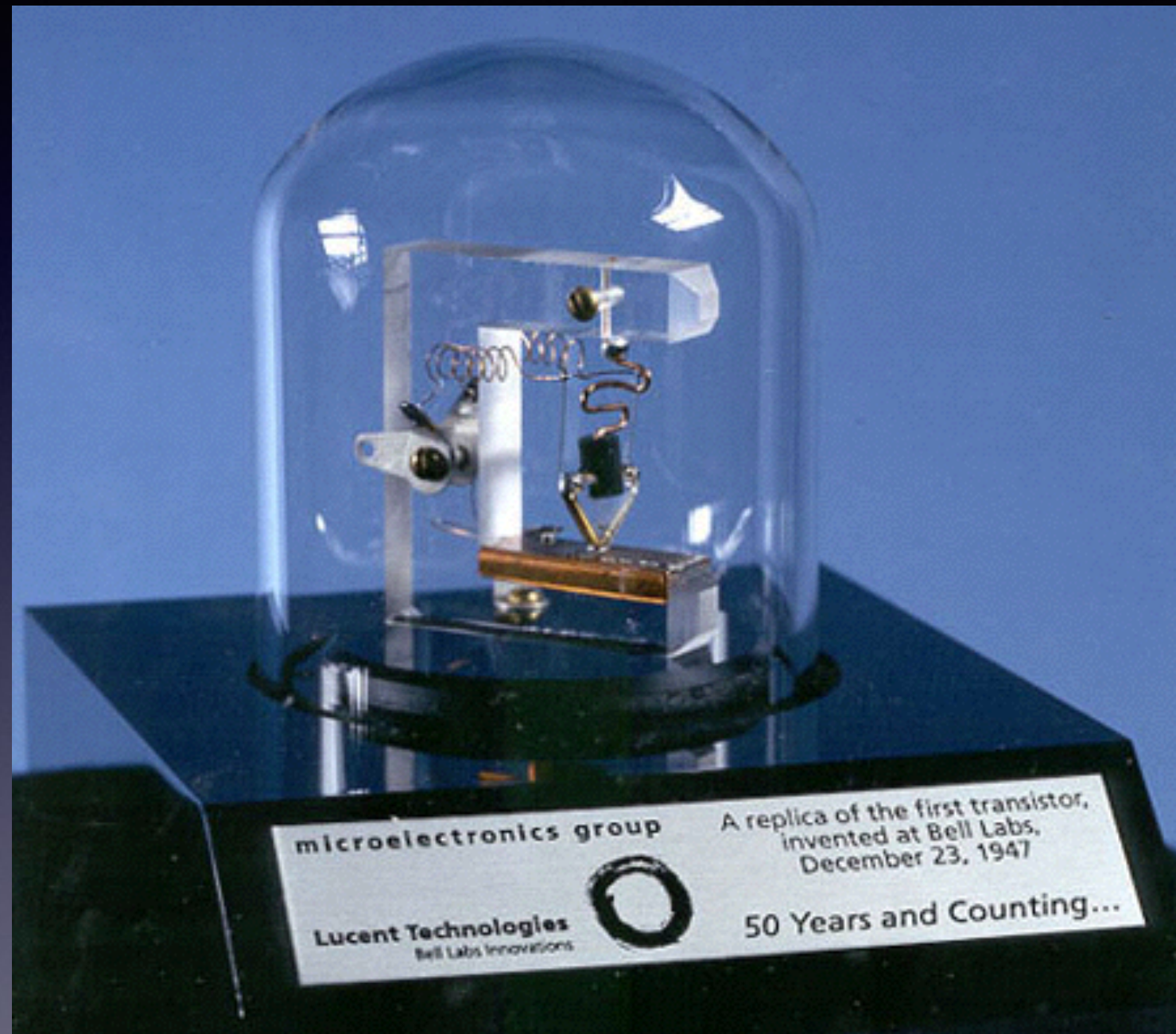
AND GATE



A	B	Light
OFF	OFF	
OFF	ON	
ON	OFF	
ON	ON	

For light to be on, we need a path from 6v to 0v




transistors






Replica of 1947 Bell Labs transistor

logic gates

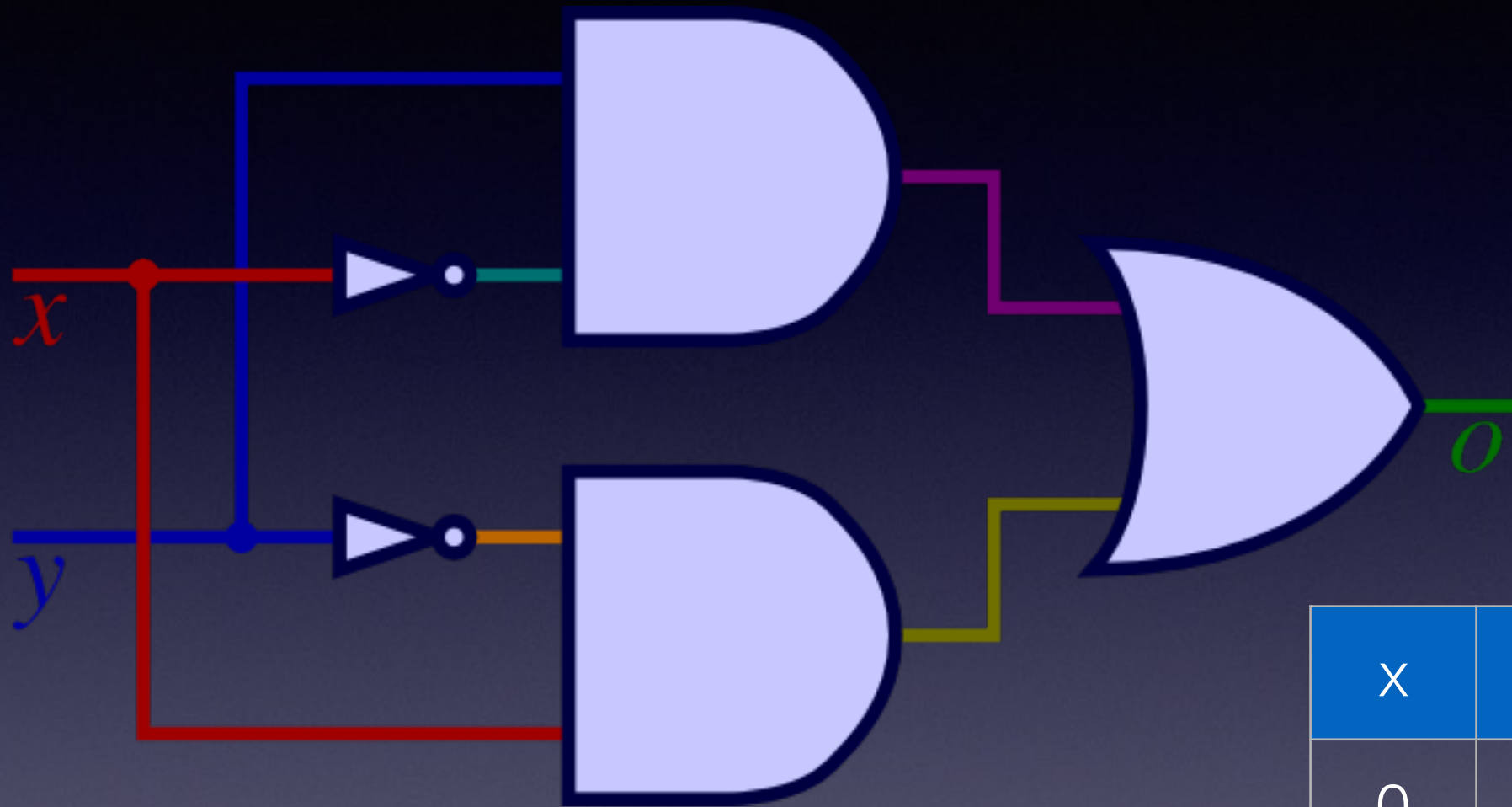
- logic gate: digital circuit that either allows input signal(s) to pass through or not
- we can combine gates together to make a logic circuit

<u>AND</u>		$A \cdot B$	<table><tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr><tr><th>A</th><th>B</th><th>A AND B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	INPUT		OUTPUT	A	B	A AND B	0	0	0	0	1	0	1	0	0	1	1	1
INPUT		OUTPUT																			
A	B	A AND B																			
0	0	0																			
0	1	0																			
1	0	0																			
1	1	1																			
<u>OR</u>		$A + B$	<table><tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr><tr><th>A</th><th>B</th><th>A OR B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	INPUT		OUTPUT	A	B	A OR B	0	0	0	0	1	1	1	0	1	1	1	1
INPUT		OUTPUT																			
A	B	A OR B																			
0	0	0																			
0	1	1																			
1	0	1																			
1	1	1																			
<u>NOT</u>		\overline{A}	<table><tr><th>INPUT</th><th>OUTPUT</th></tr><tr><th>A</th><th>NOT A</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	INPUT	OUTPUT	A	NOT A	0	1	1	0										
INPUT	OUTPUT																				
A	NOT A																				
0	1																				
1	0																				

NAND/NOR/XOR gates

<u>NAND</u>		$\overline{A \cdot B}$	<table><tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr><tr><th>A</th><th>B</th><th>A NAND B</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	INPUT		OUTPUT	A	B	A NAND B	0	0	1	0	1	1	1	0	1	1	1	0
INPUT		OUTPUT																			
A	B	A NAND B																			
0	0	1																			
0	1	1																			
1	0	1																			
1	1	0																			
<u>NOR</u>		$\overline{A + B}$	<table><tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr><tr><th>A</th><th>B</th><th>A NOR B</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	INPUT		OUTPUT	A	B	A NOR B	0	0	1	0	1	0	1	0	0	1	1	0
INPUT		OUTPUT																			
A	B	A NOR B																			
0	0	1																			
0	1	0																			
1	0	0																			
1	1	0																			
<u>XOR</u>		$A \oplus B$	<table><tr><th colspan="2">INPUT</th><th>OUTPUT</th></tr><tr><th>A</th><th>B</th><th>A XOR B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	INPUT		OUTPUT	A	B	A XOR B	0	0	0	0	1	1	1	0	1	1	1	0
INPUT		OUTPUT																			
A	B	A XOR B																			
0	0	0																			
0	1	1																			
1	0	1																			
1	1	0																			

logic circuit example



x	y	o
0	0	
0	1	
1	0	
1	1	

truth table

- consider a logic function with three inputs: A, B, and C, and 3 outputs: D, E, and F. D is true if at least one input is true. E is true if exactly two inputs are true. F is true only if all three inputs are true.

inputs			outputs		
A	B	C	D	E	F
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

truth table

- consider a logic function with three inputs: A, B, and C, and 3 outputs: D, E, and F. D is true if at least one input is true. E is true if exactly two inputs are true. F is true only if all three inputs are true.

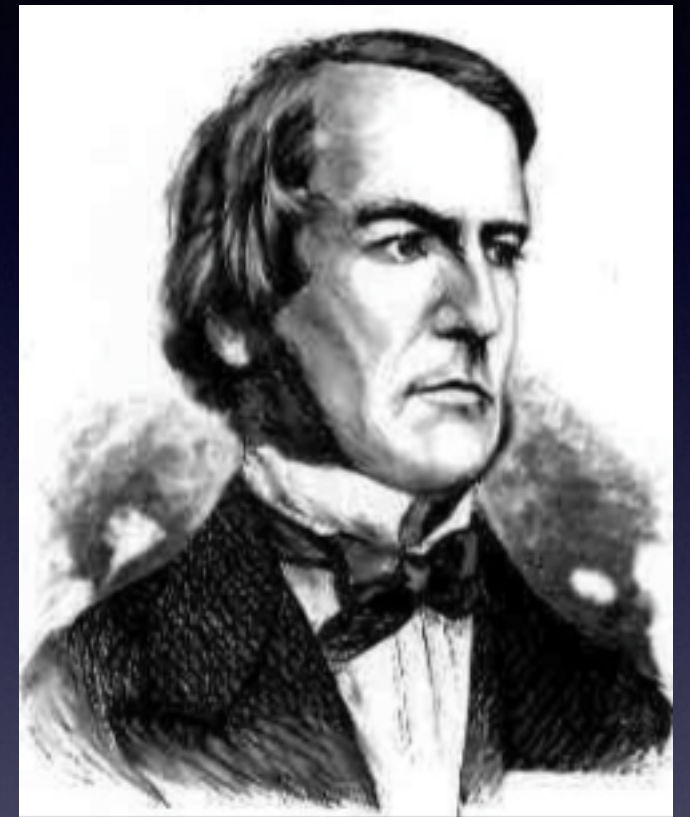
inputs			outputs		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

combinational logic

- combinational logic: output is a function of present input only
- sequential logic: output is a function of present input and also the history of input. in other words, sequential logic has memory

Boolean algebra

- variables have value 1 or 0 (true or false), and there are three operators:
 - OR: $A + B$, $A \mid B$
 - result is 1 if either A or B are 1 (or both)
 - AND: AB , $A \cdot B$, $A \& B$
 - result is 1 if both A and B are 1
 - NOT: A' , $\sim A$, \bar{A} , $!A$
 - result is 1 if A is 0
- every logic circuit can be represented as an expression in Boolean algebra with these three operators



George Boole
(1815-1864)

Boolean algebra

- recall our earlier logic function with three inputs: A, B, and C, and 3 outputs: D, E, and F. D is true if at least one input is true. E is true if exactly two inputs are true. F is true only if all three inputs are true.
how can we write these using Boolean algebra?

OR: $A + B$

$$D = A + B + C$$

AND: AB

$$E = ABC' + AB'C + A'BC$$

NOT: A'

$$F = ABC$$

identities

- Identities are useful to manipulate logic equations

- $A + 0 =$

- $A + 1 =$

- $A + A' =$

- $A0 =$

- $A1 =$

- $AA' =$

identities

- Identities are useful to manipulate logic equations

- $A + 0 = A$

- $A + 1 = 1$

- $A + A' = 1$

- $A0 = 0$

- $A1 = A$

- $AA' = 0$

identities

- Identity law: $A+0=A$ and $A1=A$
- Zero and one law: $A+1=1$ and $A0=0$
- Inverse laws: $A+A'=1$ and $AA'=0$
- Commutative law: $A+B=B+A$ and $AB=BA$
- Associative law: $A+(B+C)=(A+B)+C$ and $A(BC)=(AB)C$
- Distributive law: $A(B+C)=AB+AC$ and $A+BC=(A+B)(A+C)$
- DeMorgan's Theorem: $(A + B)' = A'B'$ and $(AB)' = A' + B'$

distributive law

- $A(B+C)=AB+AC$ and $A+BC=(A+B)(A+C)$

inputs			results			
A	B	C	$A(B+C)$	$AB+AC$	$A+BC$	$(A+B)(A+C)$
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

distributive law

- $A(B+C)=AB+AC$ and $A+BC=(A+B)(A+C)$

inputs			results			
A	B	C	$A(B+C)$	$AB+AC$	$A+BC$	$(A+B)(A+C)$
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	0	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

logic circuit minimization

- given a logic function, create a logic circuit which implements the logic function, with the fewest number of gates possible

identities

- Identities can be used to minimize logic equations

- $A + AB$

$$= A1 + AB = A(1+B) = A1 = A$$

- Hint: distributive

$$A(B+C) = AB + AC$$

A	B	A+AB
0	0	0
0	1	0
1	0	1
1	1	1

another example

- $(C + B')(B + C')$

$$= CB + CC' + B'B + B'C'$$

$$= CB + 0 + 0 + B'C'$$

$$= CB + B'C'$$

- Hint: multiply it out first

standardizing minimization

- how can we standardize minimization of logic circuits?

A	B	C	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

standardizing minimization

- write **minterms** - product (AND) of all input variables

A	B	C	out	minterm
0	0	0	0	
0	0	1	1	
0	1	0	0	
0	1	1	1	
1	0	0	0	
1	0	1	1	
1	1	0	0	
1	1	1	0	

standardizing minimization

- write **minterms** - product (AND) of all input variables

A	B	C	out	minterm
0	0	0	0	$A'B'C'$
0	0	1	1	$A'B'C$
0	1	0	0	$A'BC'$
0	1	1	1	$A'BC$
1	0	0	0	$AB'C'$
1	0	1	1	$AB'C$
1	1	0	0	ABC'
1	1	1	0	ABC

standardizing minimization

- sum of products (SOP): sum (OR) of all minterms where output is 1

A	B	C	out	minterm
0	0	0	0	$A'B'C'$
0	0	1	1	$A'B'C$
0	1	0	0	$A'BC'$
0	1	1	1	$A'BC$
1	0	0	0	$AB'C'$
1	0	1	1	$AB'C$
1	1	0	0	ABC'
1	1	1	0	ABC

$$A'B'C + A'BC + AB'C$$

another example

- what is the Boolean algebra expression associated with this truth table?

A	B	C	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

$$A'B'C + A'BC + AB'C' + AB'C$$

Gray code

- binary system in which successive values differ by only one bit
- 00, 01, 11, 10

Karnaugh maps

- Karnaugh maps identify which inputs are relevant / irrelevant to the output. cells ordered using a Gray code

A	B	C	out	minterm
0	0	0	0	$A'B'C'$
0	0	1	1	$A'B'C$
0	1	0	0	$A'BC'$
0	1	1	1	$A'BC$
1	0	0	1	$AB'C'$
1	0	1	1	$AB'C$
1	1	0	0	ABC'
1	1	1	0	ABC

$$A'B'C + A'BC + AB'C' + AB'C$$

		AB			
		00	01	11	10
C	0				
	1				

Karnaugh maps

$$A'B'C + A'BC + AB'C' + AB'C$$

- To minimize:

A	B	C	out	minterm
0	0	0	0	$A'B'C'$
0	0	1	1	$A'B'C$
0	1	0	0	$A'BC'$
0	1	1	1	$A'BC$
1	0	0	1	$AB'C'$
1	0	1	1	$AB'C$
1	1	0	0	ABC'
1	1	1	0	ABC

- form rectangles which contain 1's and contain no 0's to cover all the 1's in the map. the size of the rectangles must be a power of 2 (1, 2, 4, 8, etc.)
- use the largest rectangles possible, and use the fewest amount of rectangles possible
- encode the result: $AB' + A'C$

		AB			
		00	01	11	10
C	0	0	0	0	1
	1	1	1	0	1

Karnaugh maps

C \ AB	AB			
	00	01	11	10
0	0	1	1	1
1	0	0	1	0

- rectangles can overlap
- result:

C \ AB	AB			
	00	01	11	10
0	1	1	1	1
1	0	0	1	0

- rectangles can span 2, 4, 8 cells
- result:

Karnaugh maps

AB		00	01	11	10
C	0	0	1	1	1
	1	0	0	1	0

- rectangles can overlap
- result: $AB + BC' + AC'$

AB		00	01	11	10
C	0	1	1	1	1
	1	0	0	1	0

- rectangles can span 2, 4, 8 cells
- result: $C' + AB$

Karnaugh maps

CD \ AB		AB			
		00	01	11	10
CD	00	0	0	0	0
	01	1	0	0	1
	11	1	0	0	1
	10	0	0	0	0

- rectangles can wrap around

- result:

CD \ AB		AB			
		00	01	11	10
CD	00	1	0	0	1
	01	0	0	0	0
	11	0	0	0	0
	10	1	0	0	1

- result:

Karnaugh maps

CD \ AB		AB			
		00	01	11	10
CD	00	0	0	0	0
	01	1	0	0	1
	11	1	0	0	1
	10	0	0	0	0

- map can wrap around
- result: $B'D$

CD \ AB		AB			
		00	01	11	10
CD	00	1	0	0	1
	01	0	0	0	0
	11	0	0	0	0
	10	1	0	0	1

- result: $B'D'$

Karnaugh maps

AB					
CD		00	01	11	10
00		0	0	0	0
01		1	x	x	x
11		1	x	x	1
10		0	0	0	0

- "Don't care" values, represented by an "x", can be interpreted whichever way we like

- assume all x's are 1
- result:

AB					
CD		00	01	11	10
00		1	0	0	x
01		0	x	x	0
11		0	x	x	0
10		1	0	0	1

- assume middle x's are 0, upper right x is 1.
- result:

Karnaugh maps

AB		00	01	11	10
CD	00	0	0	0	0
	01	1	x	x	x
	11	1	x	x	1
	10	0	0	0	0

- "Don't care" values, represented by an "x", can be interpreted whichever way we like

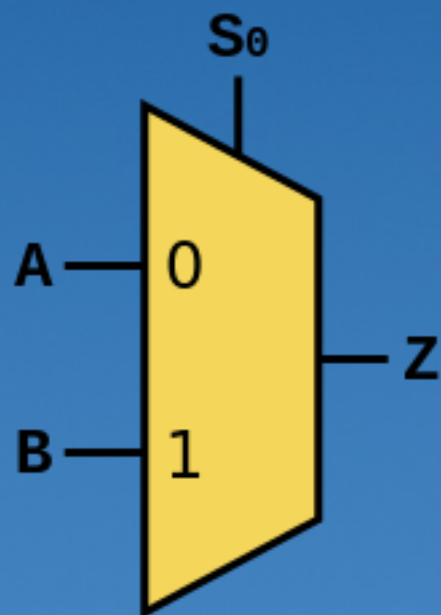
- assume all x's are 1
- result: D

AB		00	01	11	10
CD	00	1	0	0	x
	01	0	x	x	0
	11	0	x	x	0
	10	1	0	0	1

- assume middle x's are 0, upper right x is 1.
- result: $B'D'$

multiplexer

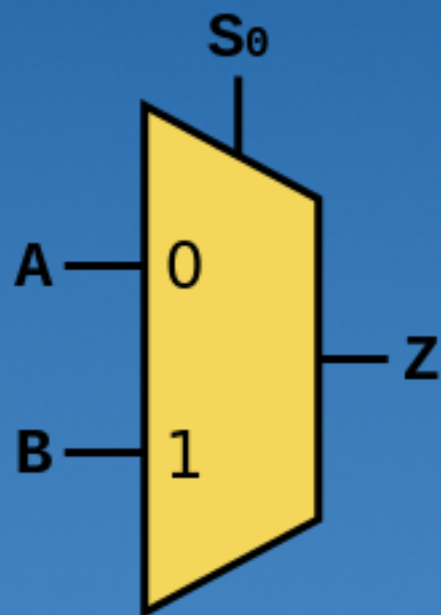
- selects between multiple inputs
- $Z = A$, if S is 0
- $Z = B$, if S is 1



A	B	S	z
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

multiplexer

- selects between multiple inputs
- $Z = A$, if S is 0
- $Z = B$, if S is 1



A	B	S	z
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

multiplexer

A	B	S	z
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

AB		00	01	11	10
S	0				
	1				

multiplexer

A	B	S	z
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

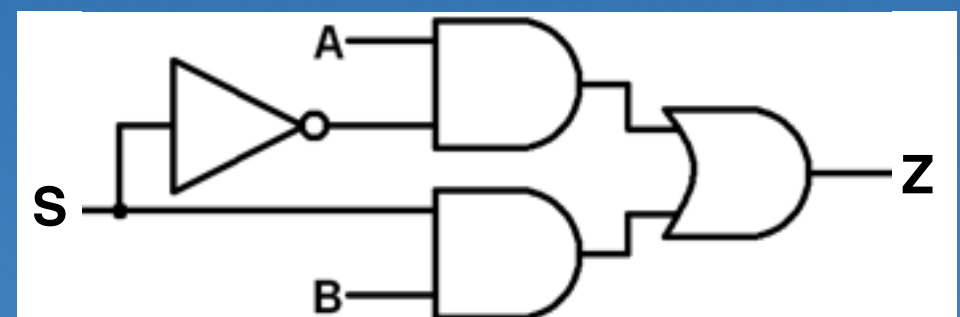
AB		00	01	11	10
S	0	0	0	1	1
	1	0	1	1	0

multiplexer

A	B	S	z
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

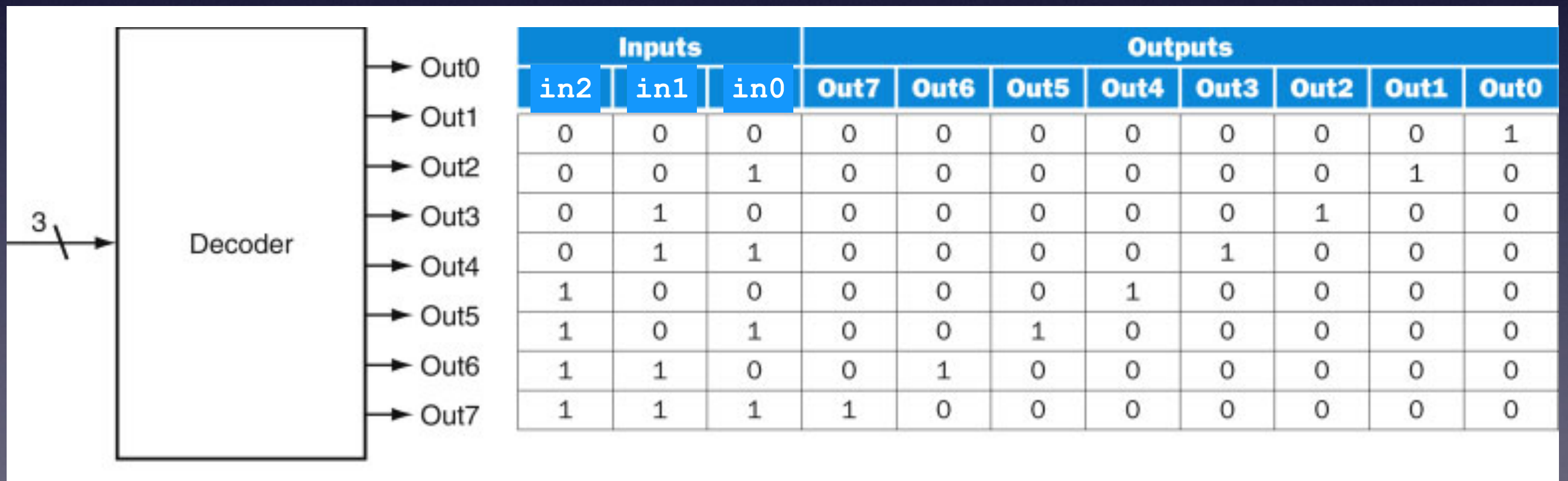
S	AB			
	00	01	11	10
0	0	0	1	1
1	0	1	1	0

$$AS' + BS$$



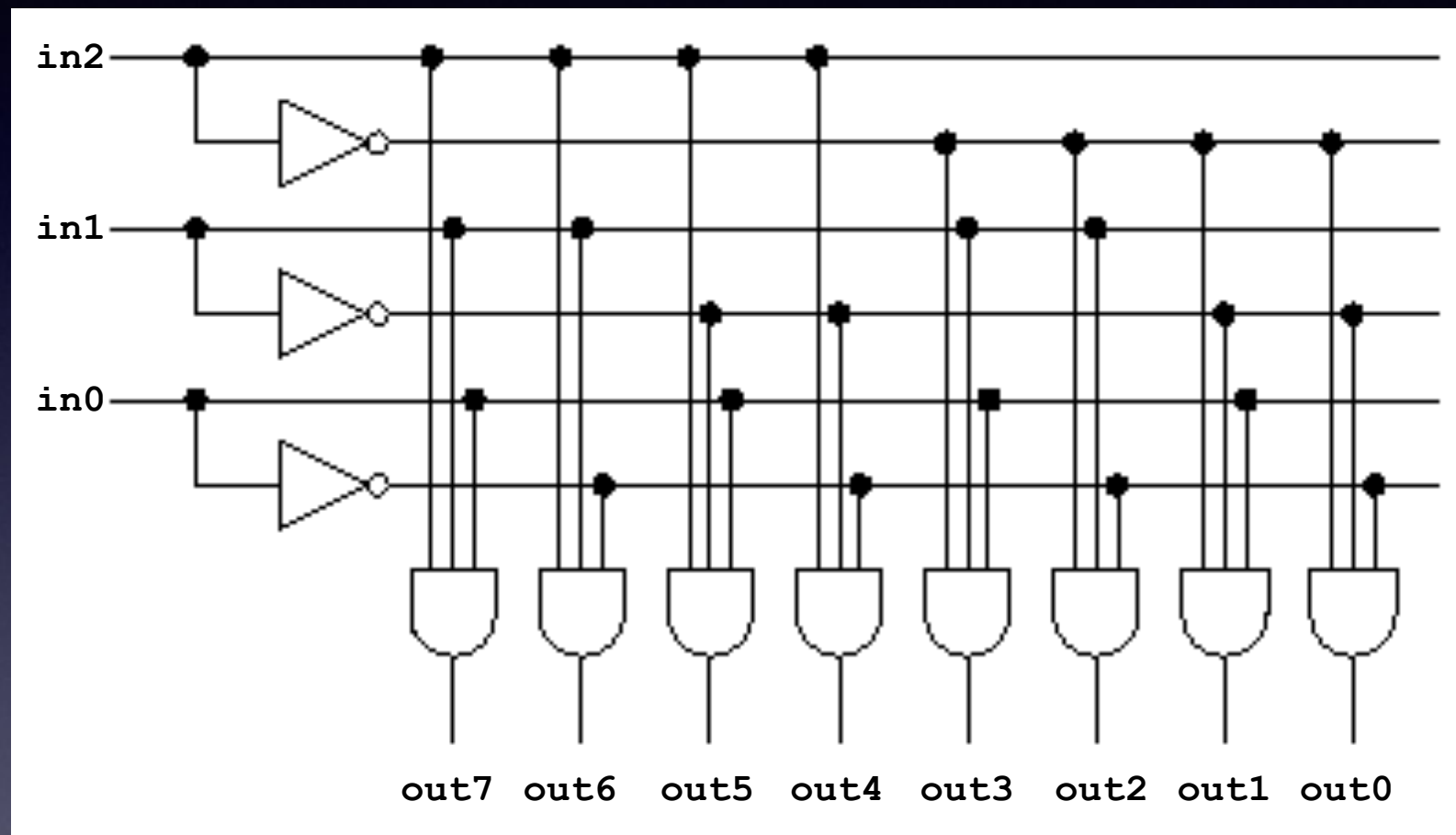
decoder

- n input bits, 2^n output bits
- exactly one output is 1 for each input combination



a 3 to 8 decoder

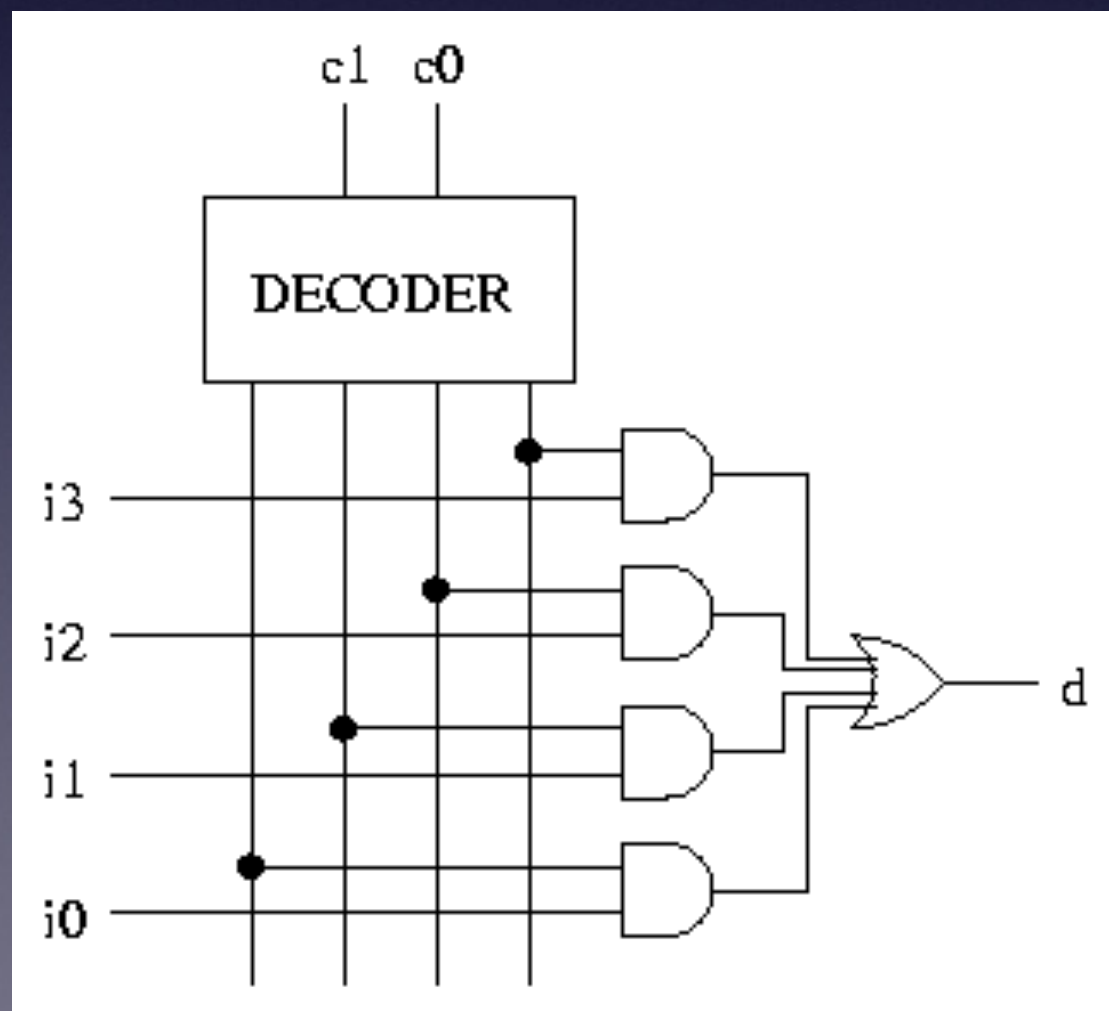
decoder



a 3 to 8 decoder

make a multiplexer using a decoder

- recall: a 2-1 multiplexer selects between two inputs.
- we can also make a 4-1 multiplexer to select between 4 inputs



$i3-i0$: input bits

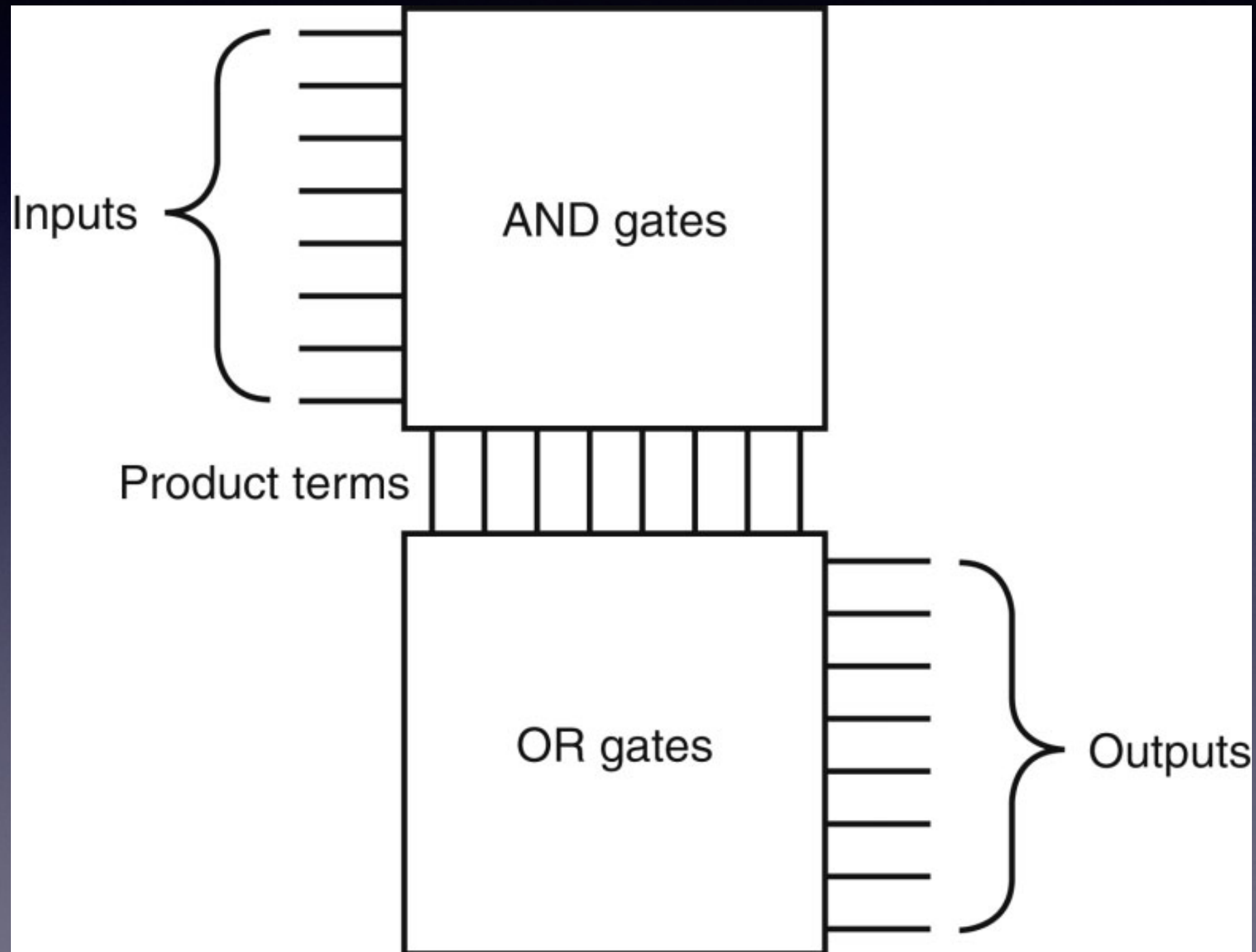
$c1-c0$: select bits

d : output

programmable logic arrays (PLA)

- we have seen that any logic circuit can be written as a sum of products (SOP)
- a programmable logic array (PLA) has a set of inputs (with their complements) which then passes through two stages of logic:
 - AND gates for the minterms
 - OR gates to sum all the minterms
- we can use a PLA to implement the truth table of a logic function

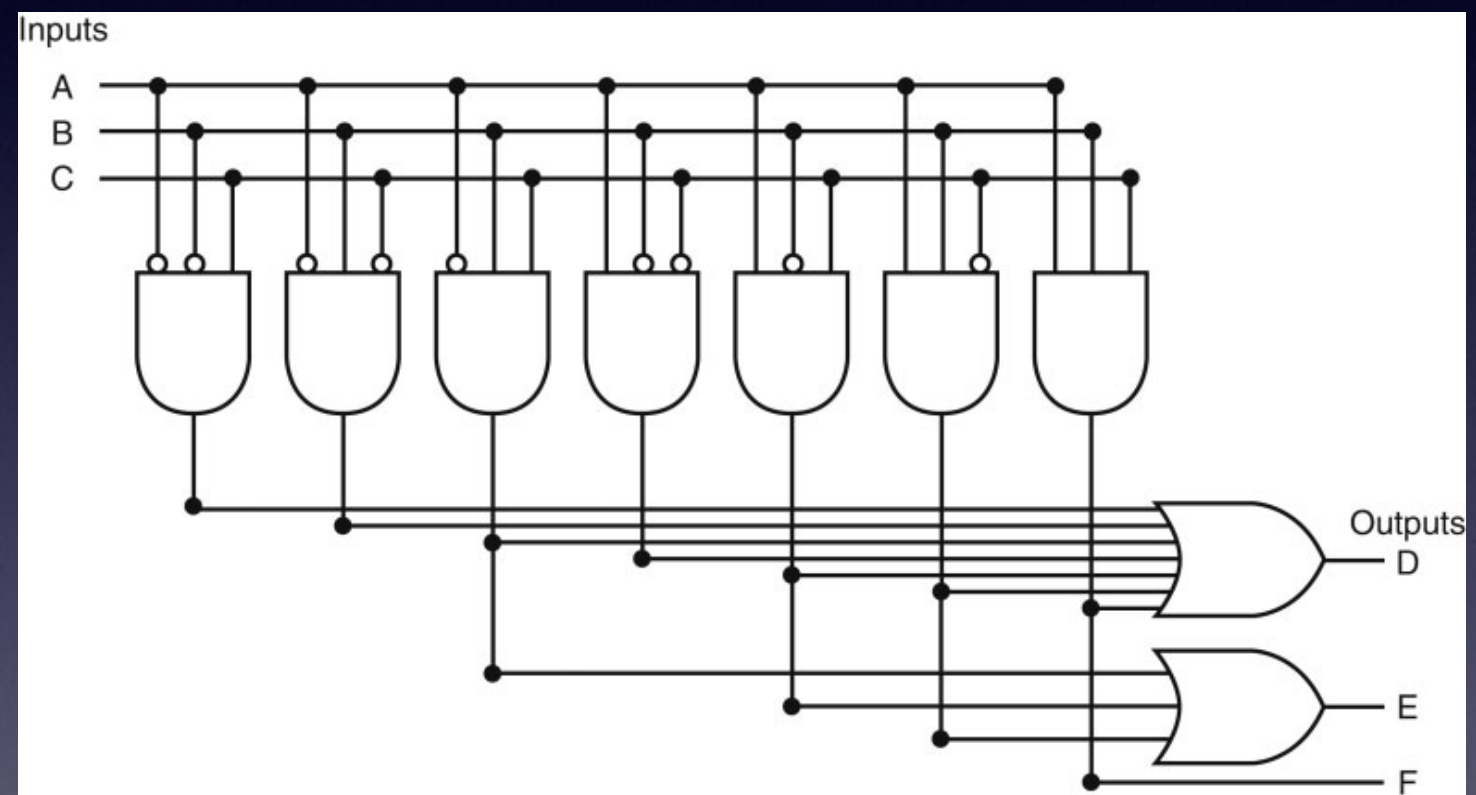
programmable logic arrays (PLA)



use a PLA for truth table

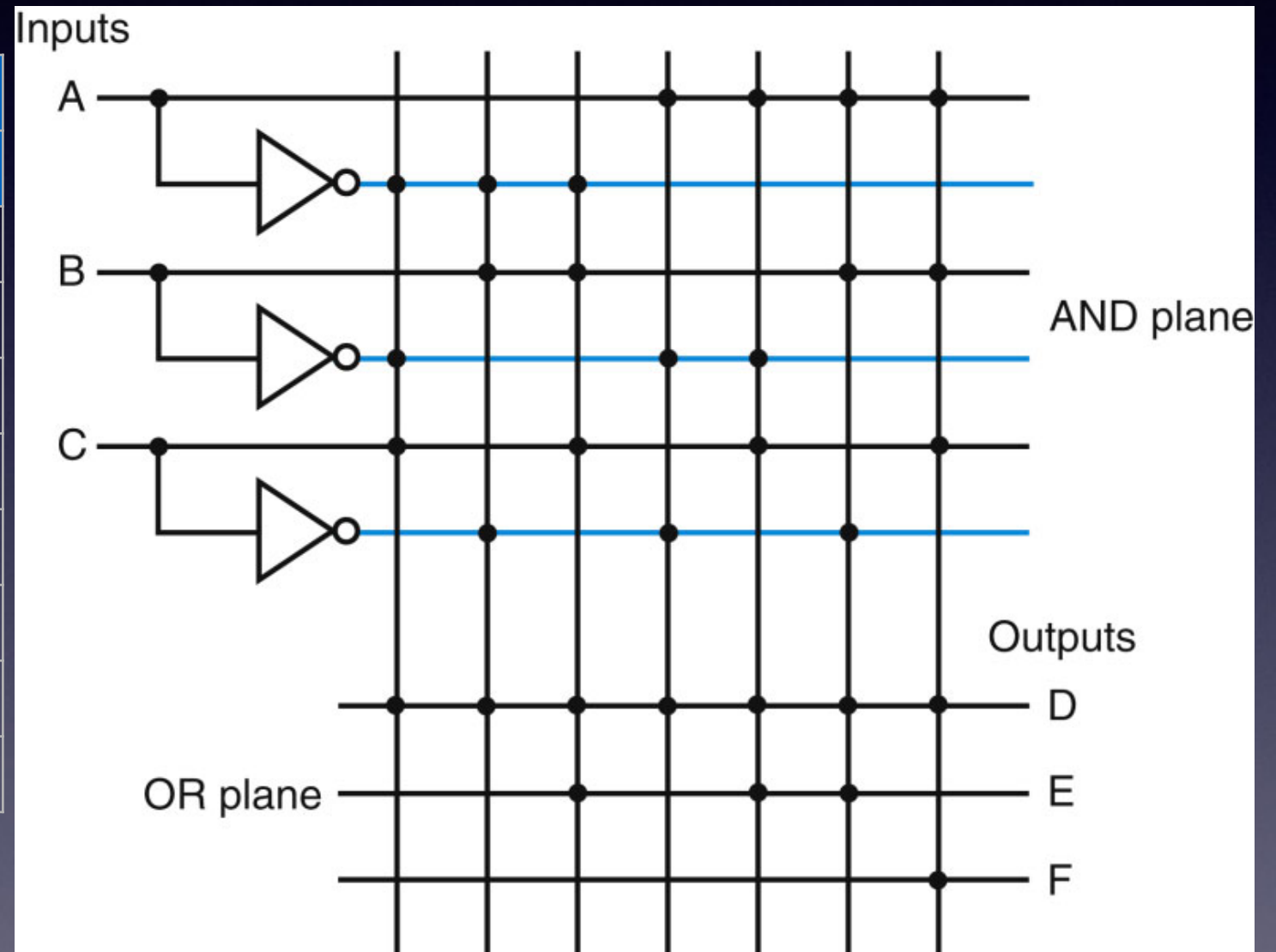
- recall our truth table from an earlier example, with corresponding PLA

inputs			outputs		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1



another PLA notation

inputs			outputs		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1



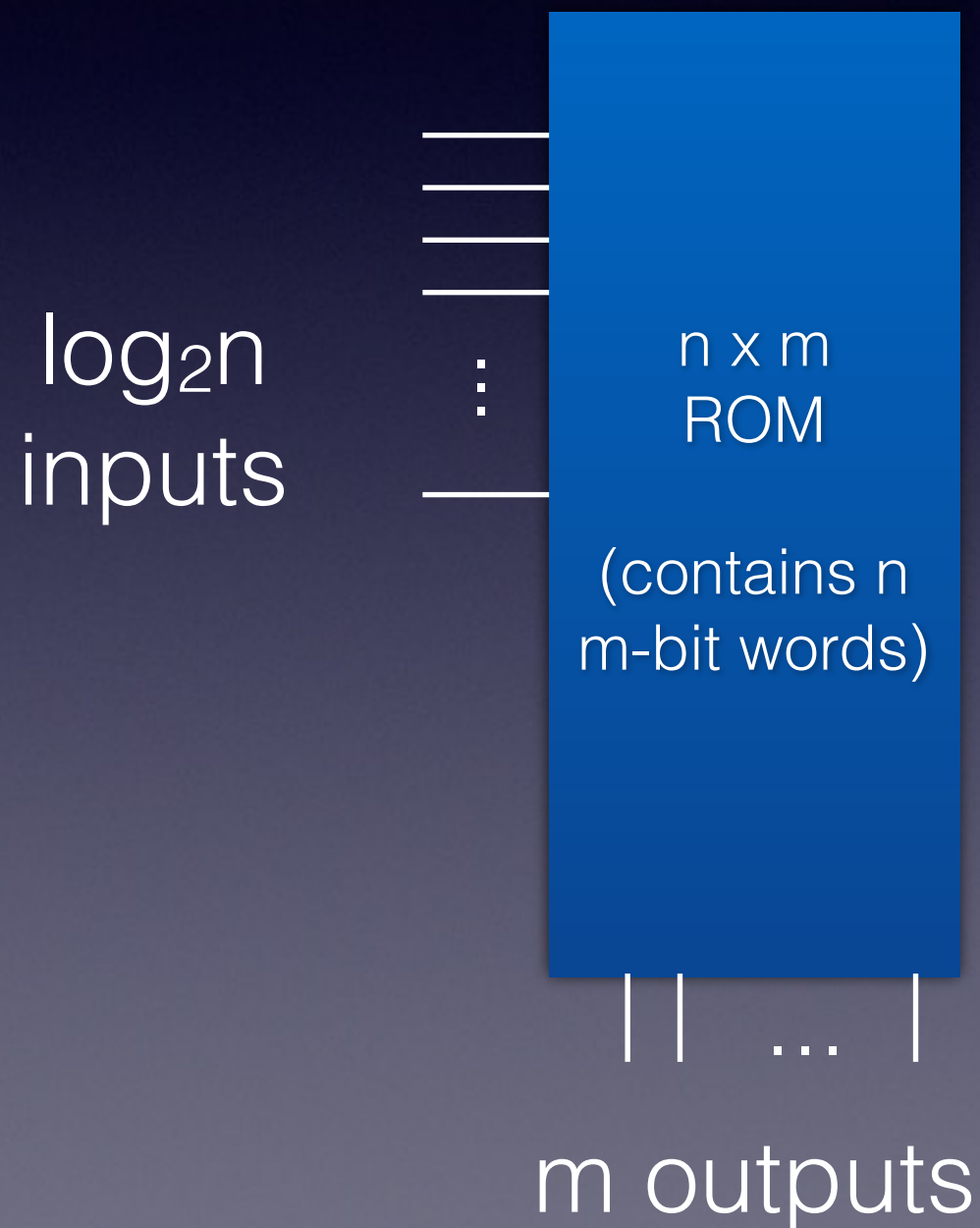
Read Only Memory (ROM)

- a Read Only Memory (ROM) is similar to a PLA, but has an output for each input combination
- our example:
 - 3 inputs A, B, C means $2^3 = 8$ input combinations
 - 3 outputs D, E, F
 - 8x3 ROM

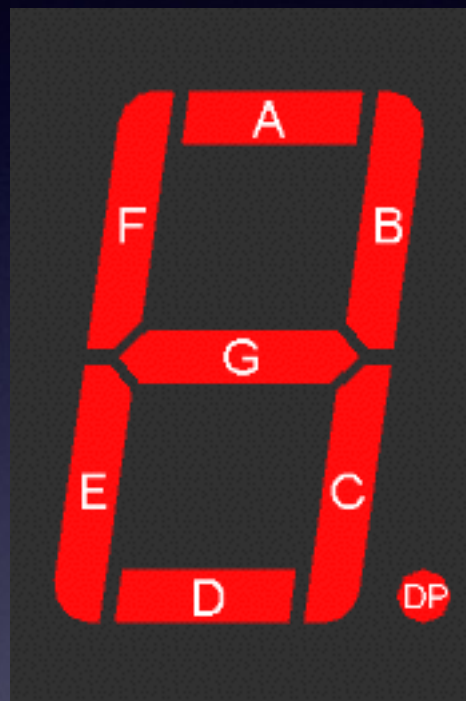
inputs			outputs		
A	B	C	D	E	F
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	0	1

Read Only Memory (ROM)

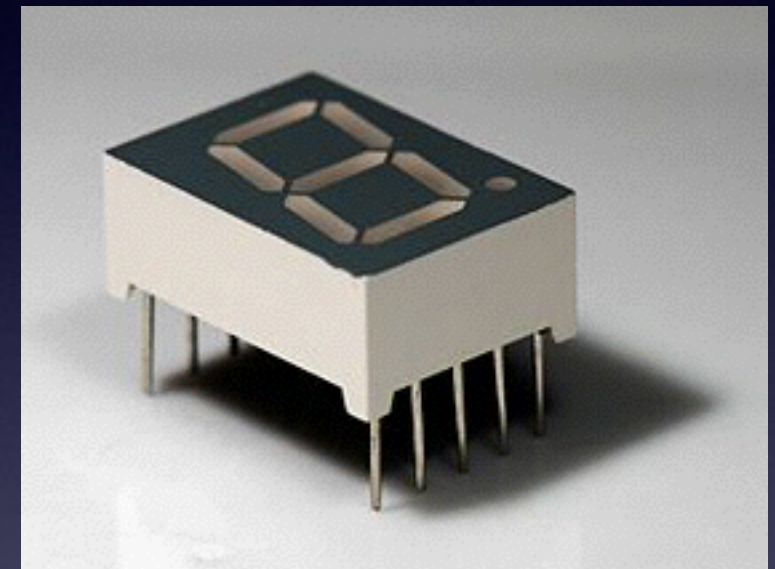
- in general



7 segment display



#	A	B	C	D	E	F	G
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1



segment E

input					output	
dec	binary				E	minterm
#	a	b	c	d		
0	0	0	0	0	1	$a'b'c'd'$
1	0	0	0	1	0	
2	0	0	1	0	1	$a'b'cd'$
3	0	0	1	1	0	
4	0	1	0	0	0	
5	0	1	0	1	0	
6	0	1	1	0	1	$a'bcd'$
7	0	1	1	1	0	
8	1	0	0	0	1	$ab'c'd'$
9	1	0	0	1	0	

		ab			
		00	01	11	10
cd	00	1	0		1
	01	0	0		0
	11	0	0		
	10	1	1		

segment E

input					output	
dec	binary				E	minterm
#	a	b	c	d		
0	0	0	0	0	1	$a'b'c'd'$
1	0	0	0	1	0	
2	0	0	1	0	1	$a'b'cd'$
3	0	0	1	1	0	
4	0	1	0	0	0	
5	0	1	0	1	0	
6	0	1	1	0	1	$a'bcd'$
7	0	1	1	1	0	
8	1	0	0	0	1	$ab'c'd'$
9	1	0	0	1	0	

		ab			
cd		00	01	11	10
	00	1	0	x	1
	01	0	0	x	0
	11	0	0	x	x
	10	1	1	x	x

segment E

input					output	
dec	binary				E	minterm
#	a	b	c	d		
0	0	0	0	0	1	$a'b'c'd'$
1	0	0	0	1	0	
2	0	0	1	0	1	$a'b'cd'$
3	0	0	1	1	0	
4	0	1	0	0	0	
5	0	1	0	1	0	
6	0	1	1	0	1	$a'bcd'$
7	0	1	1	1	0	
8	1	0	0	0	1	$ab'c'd'$
9	1	0	0	1	0	

cd \ ab				
	00	01	11	10
00	1	0	x	1
01	0	0	x	0
11	0	0	x	x
10	1	1	x	x

segment E

input					output	
dec	binary				E	minterm
#	a	b	c	d		
0	0	0	0	0	1	$a'b'c'd'$
1	0	0	0	1	0	
2	0	0	1	0	1	$a'b'cd'$
3	0	0	1	1	0	
4	0	1	0	0	0	
5	0	1	0	1	0	
6	0	1	1	0	1	$a'bcd'$
7	0	1	1	1	0	
8	1	0	0	0	1	$ab'c'd'$
9	1	0	0	1	0	

cd \ ab				
	00	01	11	10
00	1	0	x	1
01	0	0	x	0
11	0	0	x	x
10	1	1	x	x

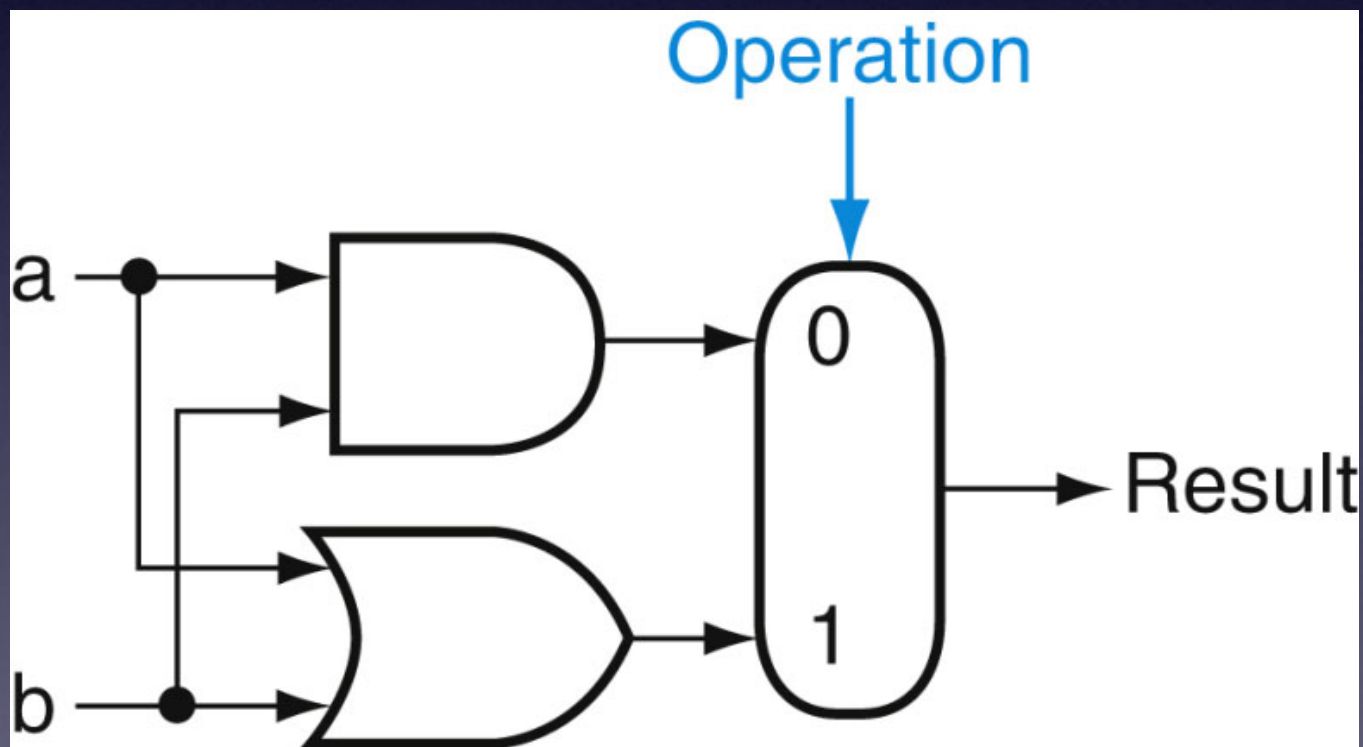
$$E = cd' + b'd'$$

arithmetic logic unit (ALU)

- an arithmetic logic unit (ALU) is the heart of a computer, and performs operations, including:
 - arithmetic: addition, subtraction, etc.
 - logical: AND, OR, etc.
 - bit: shifting, masking
 - and more...

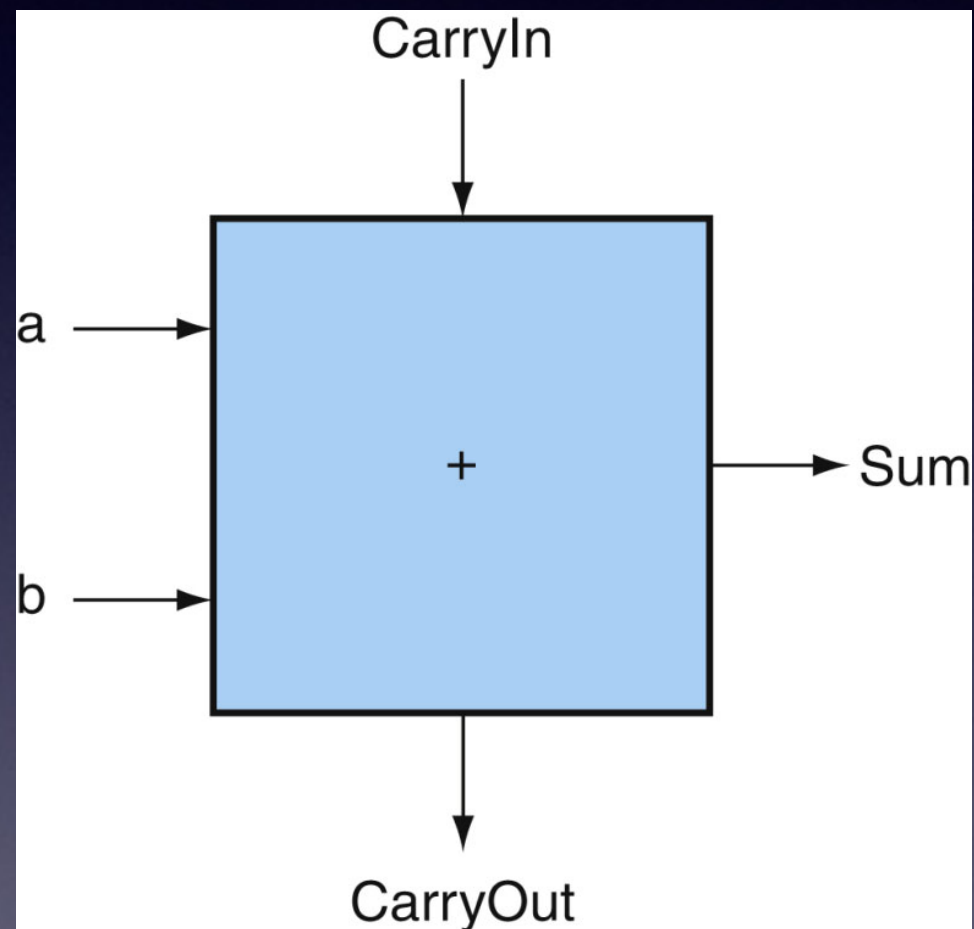
1 bit ALU

- let's start with a 1 bit ALU that supports logical AND and OR operations



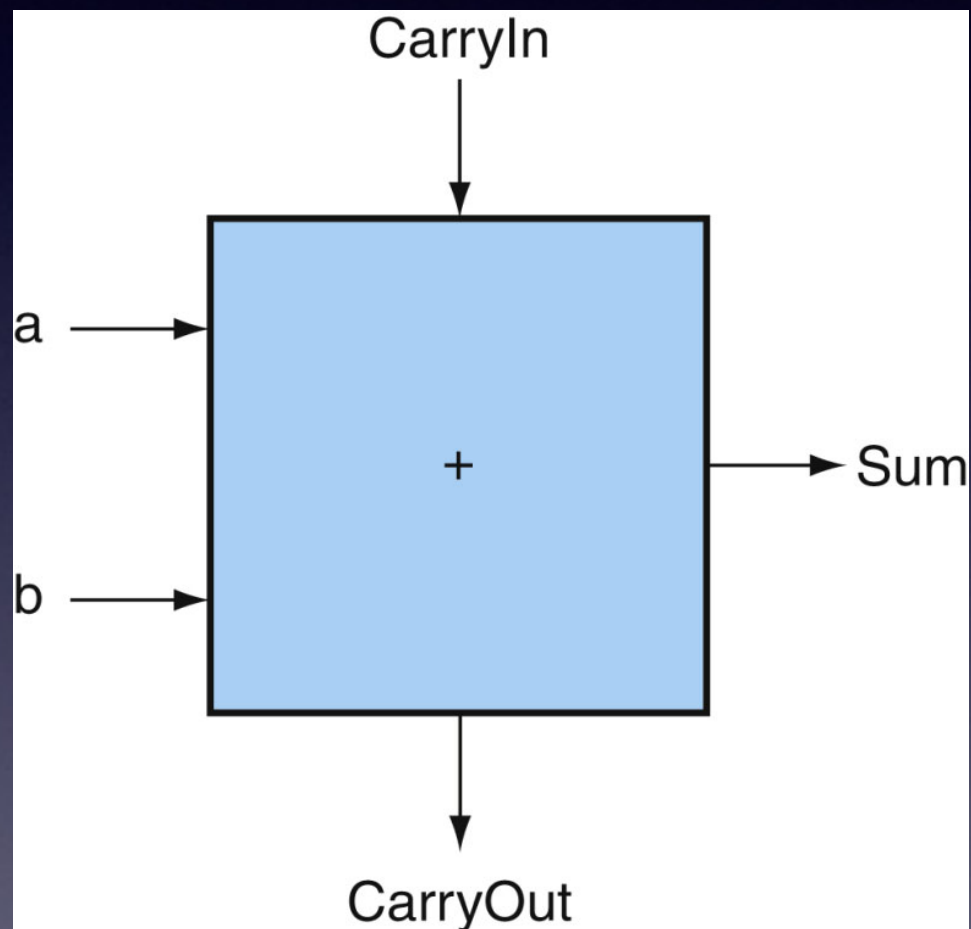
operation	result
0	$a \text{ AND } b$
1	$a \text{ OR } b$

1-bit adder



inputs			outputs	
a	b	CarryIn	CarryOut	Sum
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

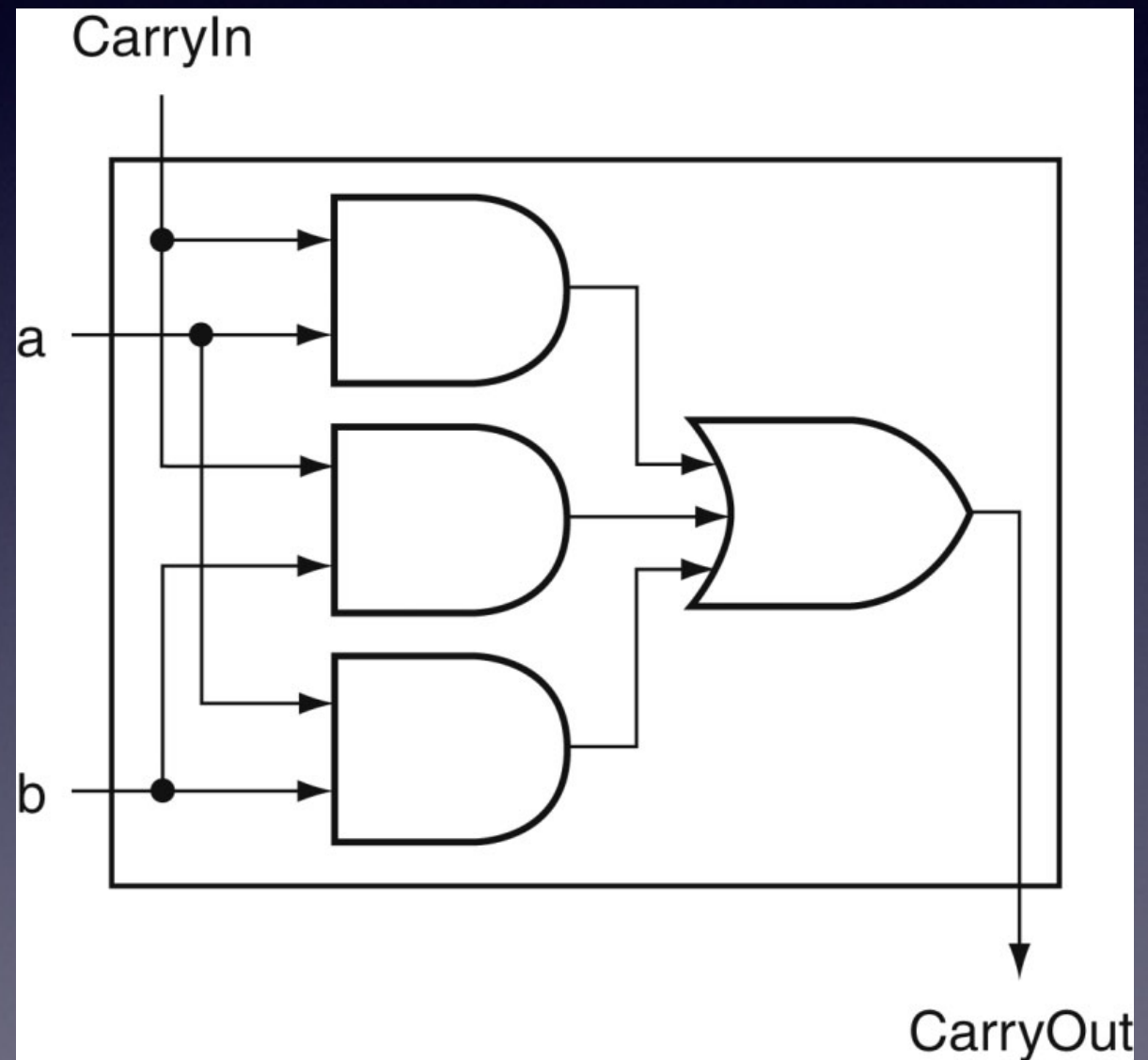
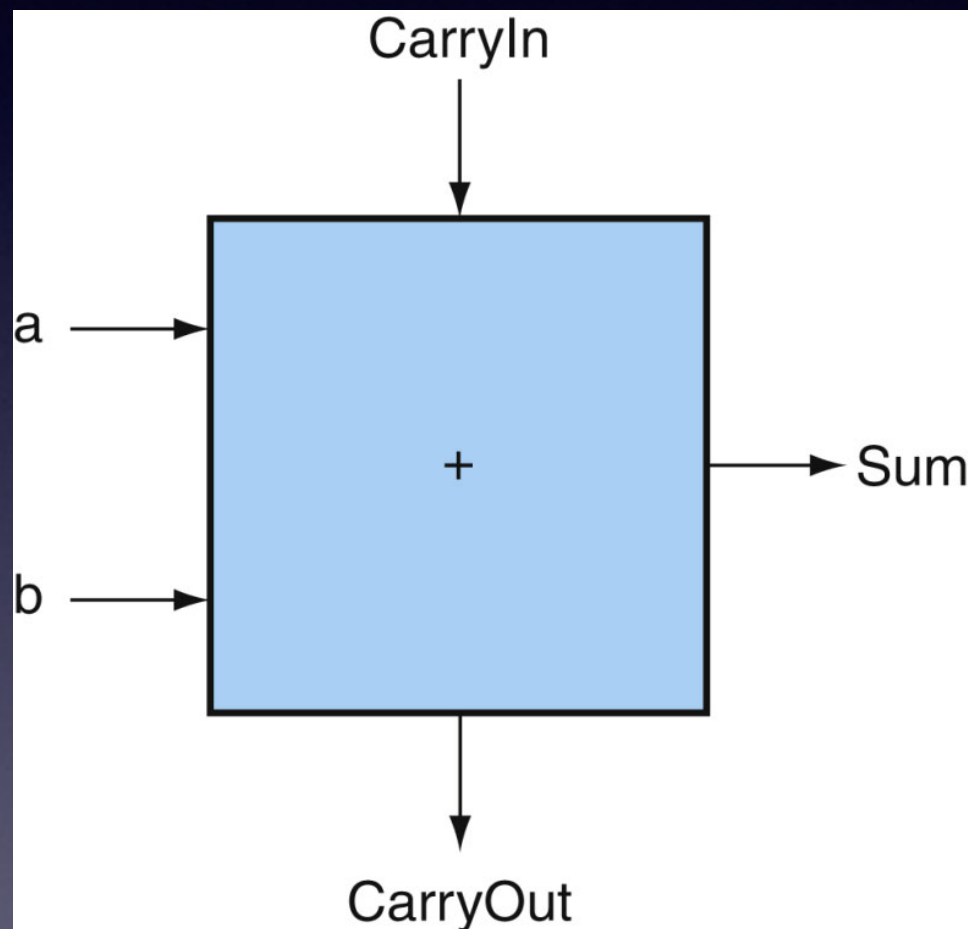
1-bit adder



inputs			outputs	
a	b	CarryIn	CarryOut	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

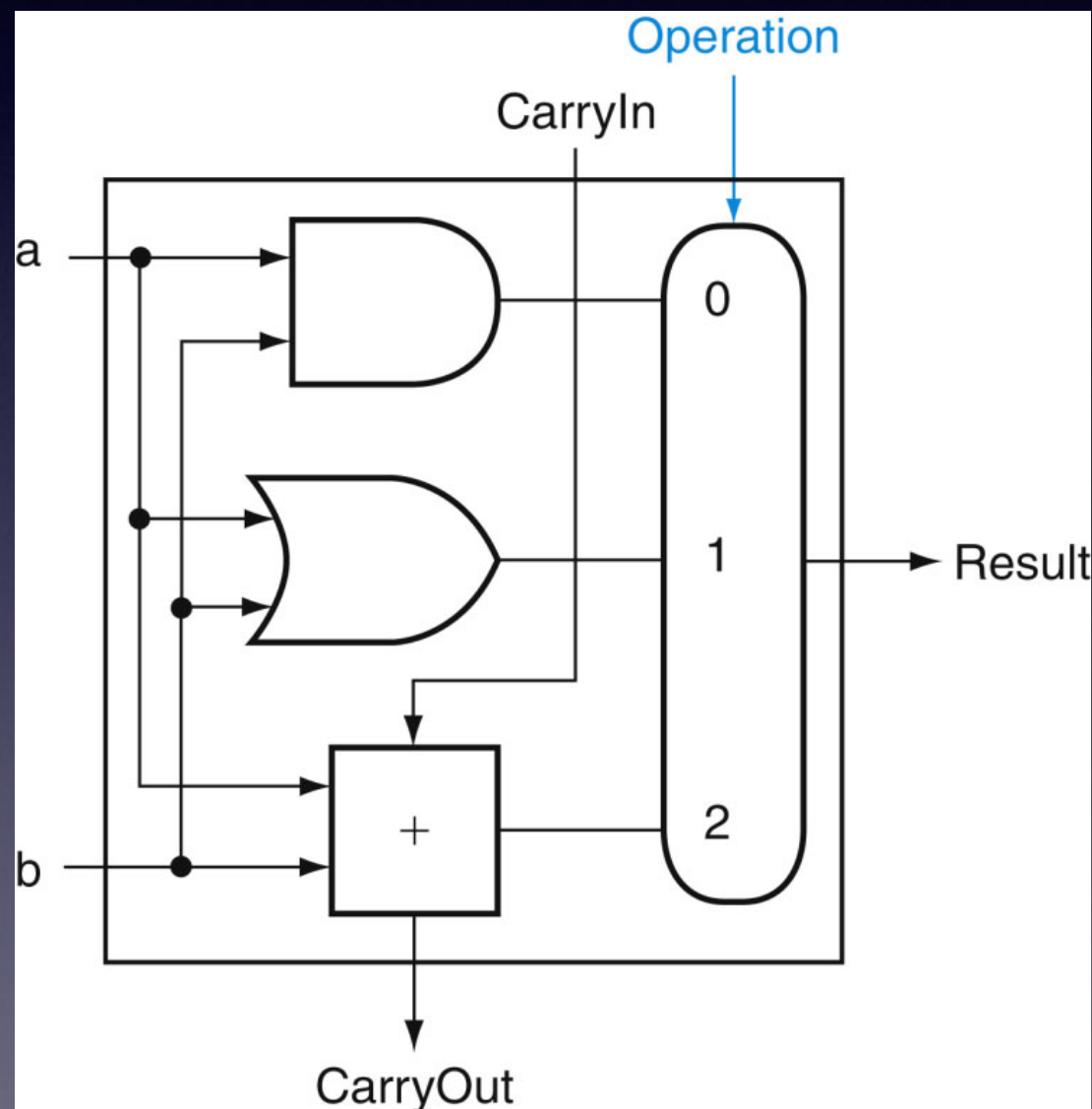
1-bit adder: CarryOut

$$\text{CarryOut} = b(\text{CarryIn}) + a(\text{CarryIn}) + ab$$



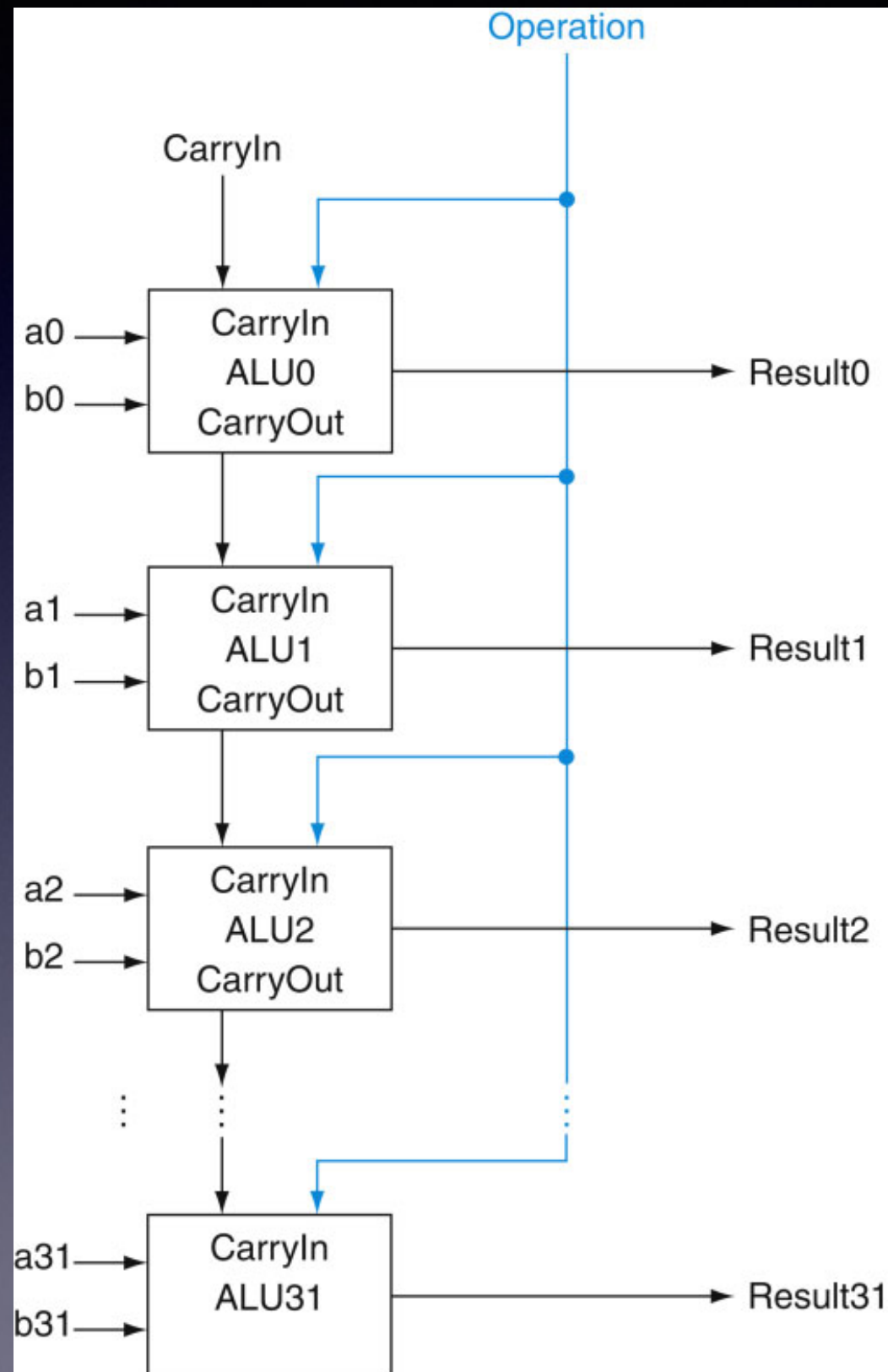
1 bit ALU

- we've added 1-bit addition to our 1-bit ALU



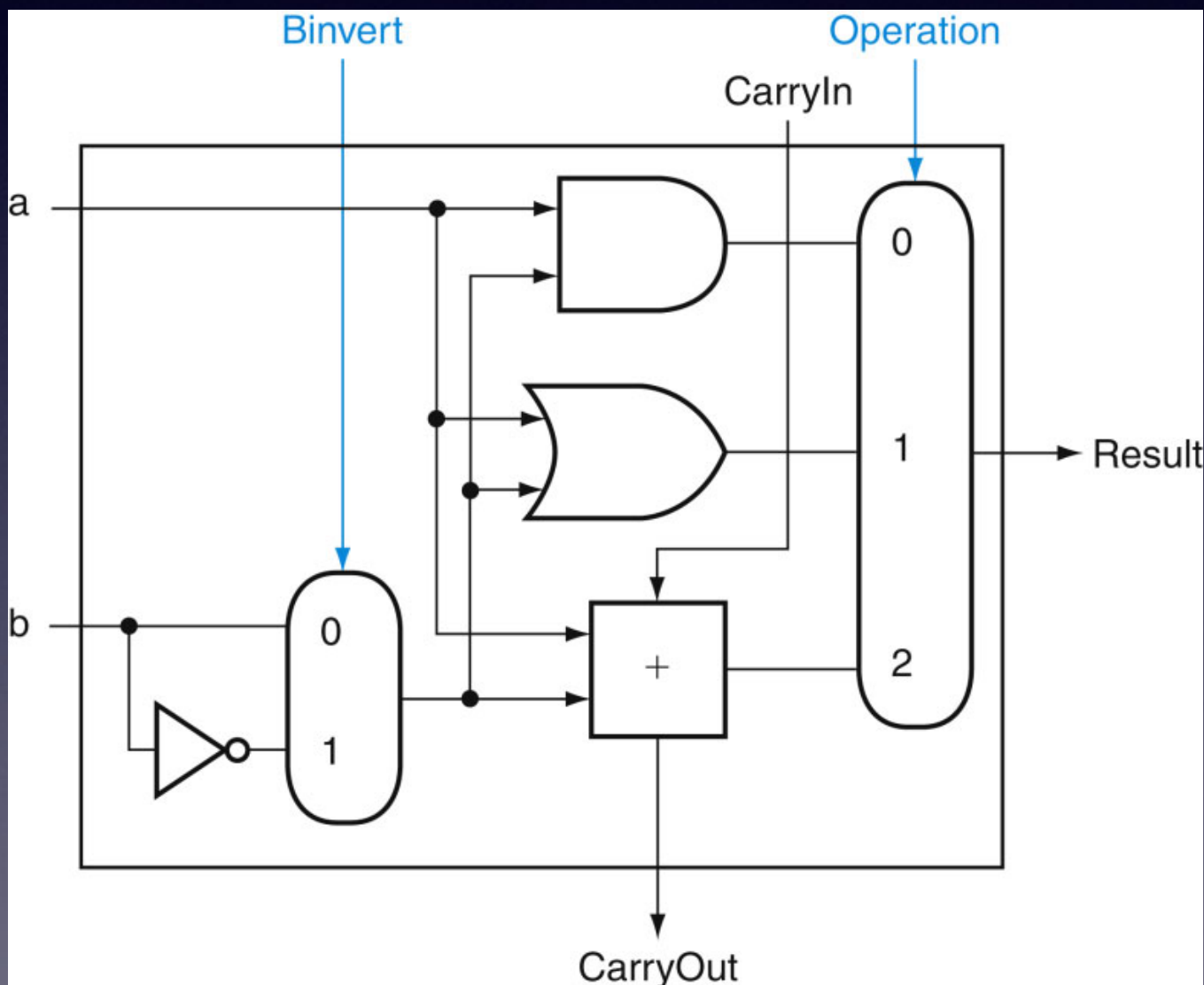
operation	result
0	<i>a</i> AND <i>b</i>
1	<i>a</i> OR <i>b</i>
2	<i>a</i> plus <i>b</i>

32-bit ALU



subtraction

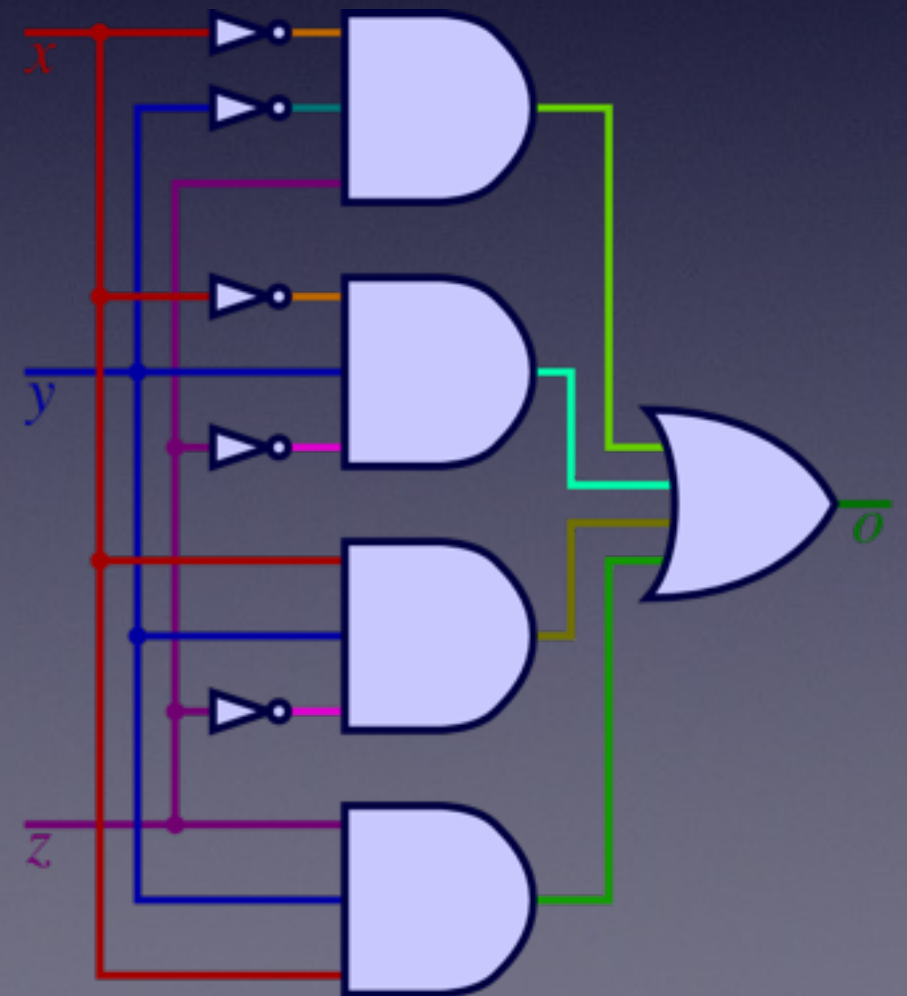
- we have added subtraction to our 1-bit ALU:
add Binvert and a NOT gate to B



operation	Binvert	CarryIn	result
0	0	0	<i>a</i> AND <i>b</i>
1	0	0	<i>a</i> OR <i>b</i>
2	0	0	<i>a</i> plus <i>b</i>
2	1	1	<i>a</i> minus <i>b</i>

Exercise 8A

1. Construct a circuit using 2-input AND gates that implements a 3-input AND gate. You may use as many 2-input AND gates as you would like. A 3-input AND gate has three inputs and one output. The output is 1 if all three inputs are 1. Provide its truth table.
2. Construct a circuit using as many 2-input OR gates as you want that implements a 4-input OR gate, i.e. its output is 1 if at least one of the 4 inputs are 1. Provide its truth table.
3. Complete a truth table for the circuit below. Note that it uses your new circuits from parts 1 and 2!



Exercise 8B

1. Design and draw a combinational circuit for equations A and B below using standard gates (AND, OR, NOT)
2. Reduce the equation as much as you can using Boolean algebra.
hint: use the identities, distributive, and associative laws
3. For equation A, using a truth table, verify that your reduced equation is equivalent to the original. don't need to do this for eq B.
4. For equation A, draw a combinational circuit for the reduced equation using standard gates (AND, OR, NOT). don't need to do this for eq B.

A. $AC' + BD' + AB + BD + AC + BC + CD$ (hint: rearrange and "factor")

B. $AB + CAB + BA' + A'CB$ (hint: rearrange so that terms have A before B before C, then combine)

5. For equations C and D, minimize as much as you can. De Morgan's Theorem will come in handy.

C. $(A + (BC' + B'C)')' + (B + C')'$

D. $((DB + C' + (AB)')' D)'$ (hint: $DB = (DB)'' = (D' + B')'$)

Exercise 8B

1. Reduce the following equations as much as you can using Boolean algebra.
 - A. $AC' + BD' + AB + BD + AC + BC + CD$ (hint: rearrange and "factor")
 - B. $AB + CAB + BA' + A'CB$ (hint: rearrange so that terms have A before B before C, then combine)
 - C. $(A + (BC' + B'C)')' + (B + C')'$ (hint: DeMorgan's Theorem will be handy)
 - D. $((DB + C' + (AB)')' D)'$ (hint: DeMorgan's Theorem will be handy.
 $DB = (DB)'' = (D' + B')'$)
2. Design and draw a combinational circuit for equations A and B using standard gates (AND, OR, NOT).
3. For equation A, using a truth table, verify that your reduced equation is equivalent to the original.
4. For equation A, draw a combinational circuit for the reduced equation using standard gates (AND, OR, NOT).

Exercise 8C

1. Minimize the following equations using a Karnaugh map. Don't forget overlaps/wraparounds!

A. $A'B'C'D' + AB'C'D + A'B'CD' + ABCD' + A'B'C'D + A'BCD'$

B. $A'B'C'D' + ABCD + A'BCD' + ABC'D + AB'CD' + A'BCD + A'BC'D'$

		AB			
		00	01	11	10
CD	00				
	01				
	11				
	10				

Exercise 8D

1. Using a PLA, implement the following 8x4 ROM. The address bits are labeled abc and the output bits are labeled ABCD. If you like, use the notation with the AND plane, OR plane, and dots to indicate connections. Alternately, draw all the gates.

address (abc)	output (ABCD)	address (abc)	output (ABCD)
000	1010	100	1101
001	1011	101	0101
010	0101	110	1001
011	0010	111	0110

2. Implement a circuit for a 2 to 4 decoder using AND, OR, and NOT gates. Once you have this, redesign it using only NAND gates. Hint: make a NOT gate using a NAND gate first, then you can use it throughout your logic circuit.

Exercise 8E

1. Determine the sum of product (SOP) form for the logic circuit for segment **A** of a 7-segment display. hint: use a Karnaugh map, and don't forget to take advantage of "don't care's" and wrap-arounds.
2. Determine the SOP form for a function U such that U is 1 if $AB < CD$ where AB and CD are two-bit, unsigned numbers. Hint: fill in a truth table for all combinations of the input values, then do a Karnaugh map.