UC Berkeley
Teaching Professor
Dan Garcia

CS61C

Great Ideas in Computer Architecture

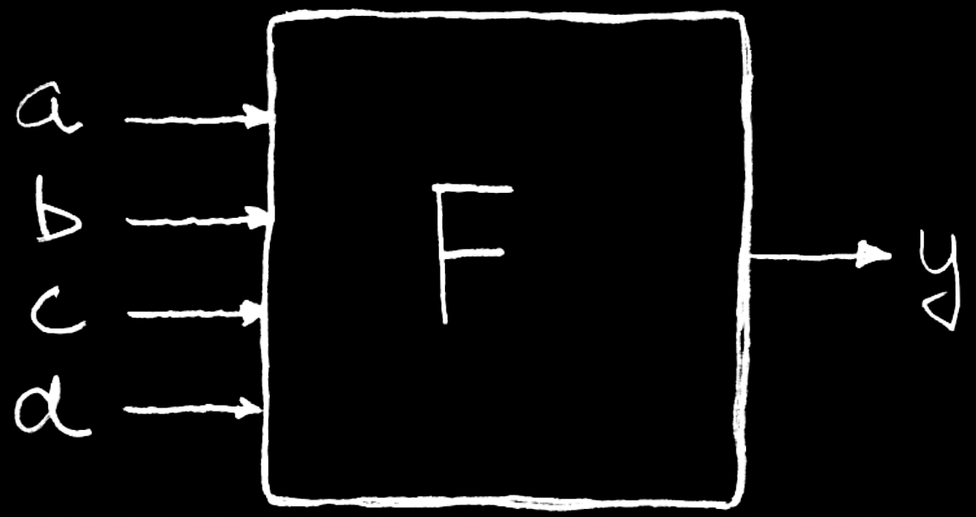
(a.k.a. Machine Structures)

UC Berkeley
Professor
Bora Nikolić

Combinational Logic

Truth Tables

Truth Tables



a	b	c	d	y
0	0	0	0	F(0,0,0,0)
0	0	0	1	F(0,0,0,1)
0	0	1	0	F(0,0,1,0)
0	0	1	1	F(0,0,1,1)
0	1	0	0	F(0,1,0,0)
0	1	0	1	F(0,1,0,1)
0	1	1	0	F(0,1,1,0)
0	1	1	1	F(0,1,1,1)
1	0	0	0	F(1,0,0,0)
1	0	0	1	F(1,0,0,1)
1	0	1	0	F(1,0,1,0)
1	0	1	1	F(1,0,1,1)
1	1	0	0	F(1,1,0,0)
1	1	0	1	F(1,1,0,1)
1	1	1	0	F(1,1,1,0)
1	1	1	1	F(1,1,1,1)

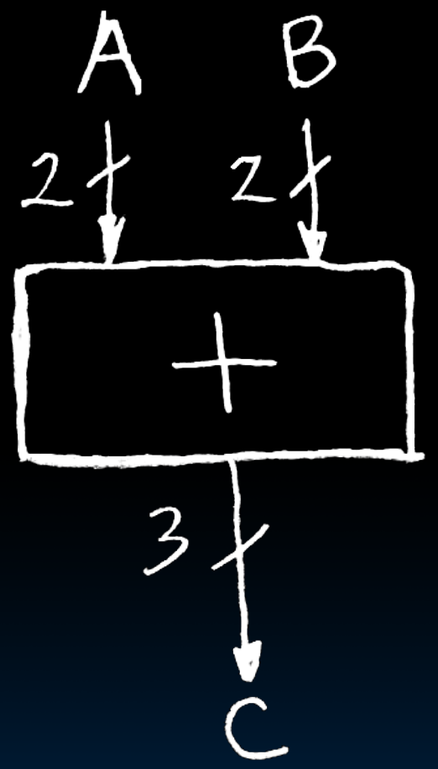
How many Fs (4-input devices) @ Fry's?

TT Example #1: 1 iff one (not both) a,b=1

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

a	y
0	$\mathbf{b}$
1	$\mathbf{\overline{b}}$

TT Example #2: 2-bit adder



A	B	C
a_1a_0	b_1b_0	$c_2c_1c_0$
00	00	000
00	01	001
00	10	010
00	11	011
01	00	001
01	01	010
01	10	011
01	11	100
10	00	010
10	01	011
10	10	100
10	11	101
11	00	011
11	01	100
11	10	101
11	11	110

How Many Rows?

TT Example #3: 32-bit unsigned adder

A	B	C
000 ... 0	000 ... 0	000 ... 00
000 ... 0	000 ... 1	000 ... 01
.	.	.
.	.	.
.	.	.
111 ... 1	111 ... 1	111 ... 10

How Many Rows?

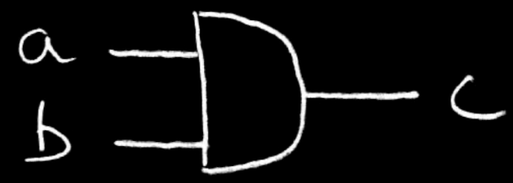
TT Example #4: 3-input majority circuit

a	b	c	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Logic Gates

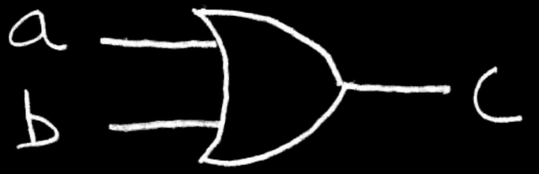
Logic Gates (1/2)

AND



ab	c
00	0
01	0
10	0
11	1

OR



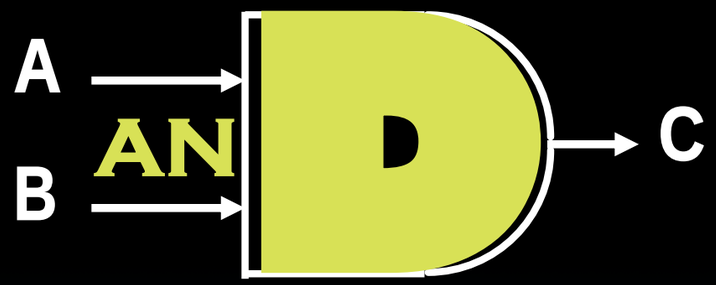
ab	c
00	0
01	1
10	1
11	1

NOT

a	b
0	1
1	0

and vs. or … how to recall which is which

and gate symbol



a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

Logic Gates (2/2)

XOR

ab	c
00	0
01	1
10	1
11	0

NAND

ab	c
00	1
01	1
10	1
11	0

NOR

ab	c
00	1
01	0
10	0
11	0

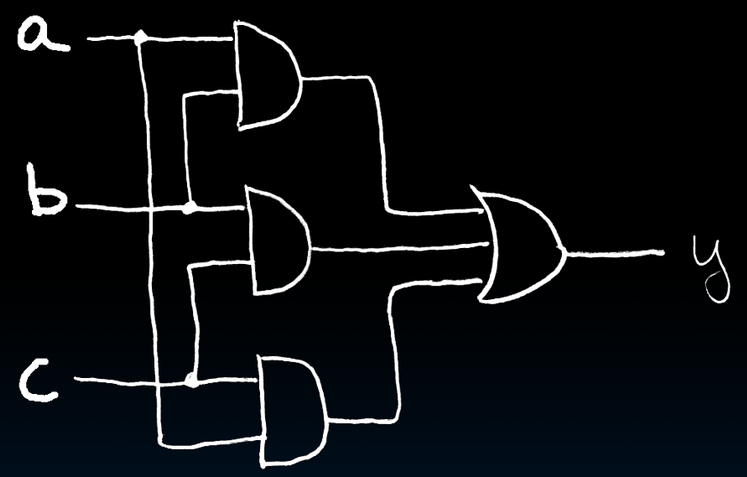
2-input gates extend to n-inputs

- N-input XOR is the only one which isn't so obvious
- It's actually simple…
  - XOR is a 1 iff the # of 1s at its input is odd

a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

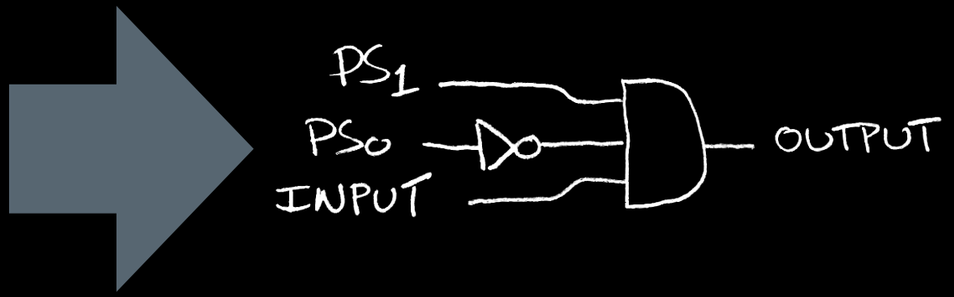
Truth Table → Gates (e.g., majority circ.)

a	b	c	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Truth Table → Gates (e.g., FSM circuit)

PS	Input	NS	Output
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	00	1

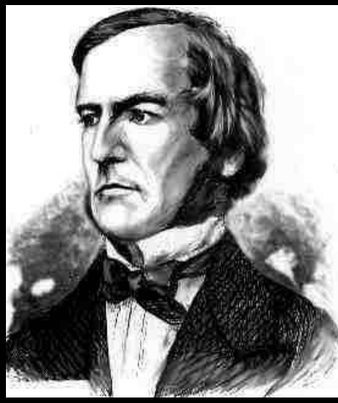


or equivalently…

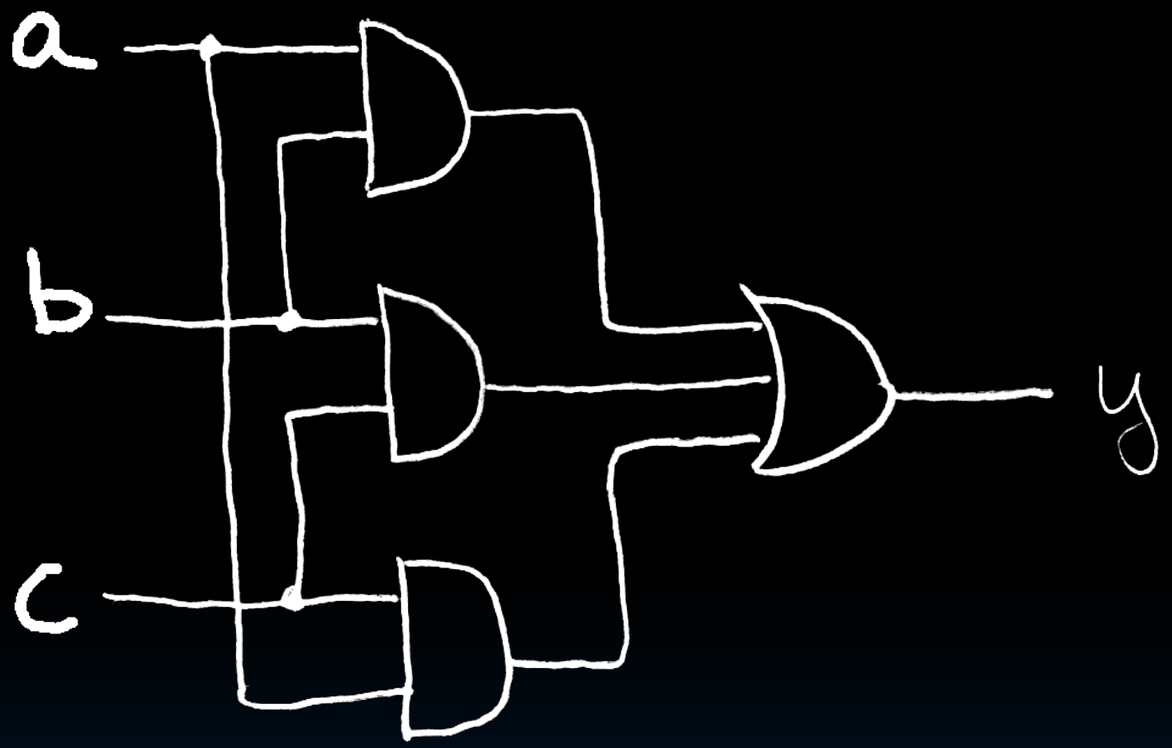
Boolean Algebra

Boolean Algebra

- George Boole, 19th Century mathematician
- Developed a mathematical system (algebra) involving logic
  - later known as "Boolean Algebra"
- Primitive functions: AND, OR and NOT
- Power of Boolean Algebra
  - there's a one-to-one correspondence between circuits made up of AND, OR and NOT gates and equations in BA
- + means OR, • means AND, $\bar{x}$ means NOT



Boolean Algebra (e.g., for majority fun.)

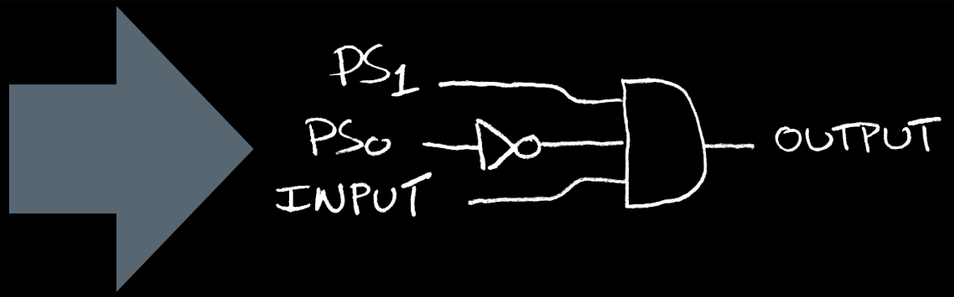


$$y = a \cdot b + a \cdot c + b \cdot c$$

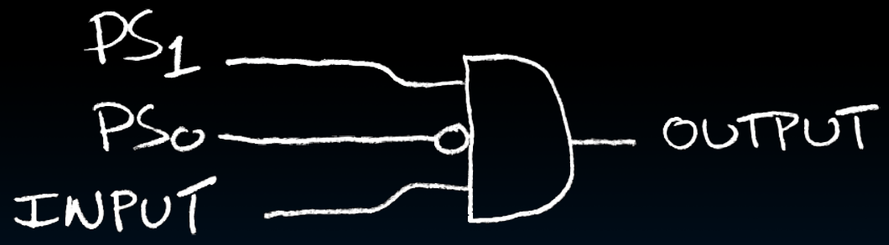
$$y = ab + ac + bc$$

Boolean Algebra (e.g., for FSM)

PS	INPUT	NS	OUTPUT
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	00	1

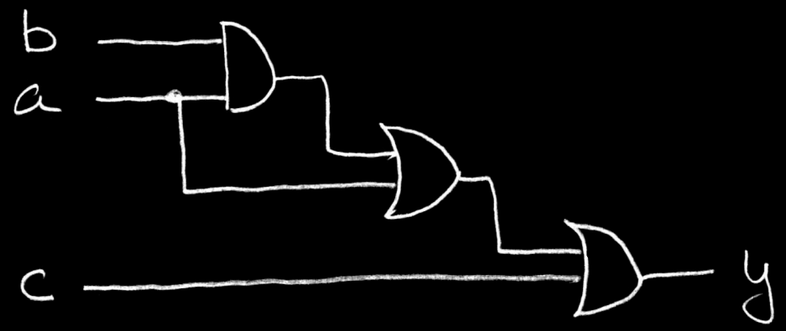


or equivalently…



$$OUTPUT = PS_1 \bullet \overline{PS_0} \bullet INPUT$$

BA: Circuit & Algebraic Simplification



original circuit

$$\downarrow$$

$$y = ab + a + c$$

equation derived from original circuit

$$\downarrow$$

$$ab + a + c$$
$$= a(b + 1) + c$$
$$= a(1) + c$$
$$= a + c$$

algebraic simplification

$$\downarrow$$

simplified circuit

BA also great for circuit verification
Circ X = Circ Y? Use BA to prove!

Laws of Boolean Algebra

Laws of Boolean Algebra

$x \cdot \overline{x} = 0$	$x + \overline{x} = 1$	complementarity
$x \cdot 0 = 0$	$x + 1 = 1$	laws of 0's and 1's
$x \cdot 1 = x$	$x + 0 = x$	identities
$x \cdot x = x$	$x + x = x$	idempotent law
$x \cdot y = y \cdot x$	$x + y = y + x$	communitive law
$(xy)z = x(yz)$	$(x + y) + z = x + (y + z)$	associativity
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	distribution
$xy + x = x$	$(x + y)x = x$	uniting theorem
$\overline{x \cdot y} = \overline{x} + \overline{y}$	$\overline{(x + y)} = \overline{x} \cdot \overline{y}$	DeMorgan's Law

Boolean Algebraic Simplification Example

$$
\begin{aligned}
y &= ab + a + c \\
&= a(b + 1) + c && \textit{distribution, identity} \\
&= a(1) + c && \textit{law of 1's} \\
&= a + c && \textit{identity}
\end{aligned}
$$

Canonical Forms

Canonical forms (1/2)

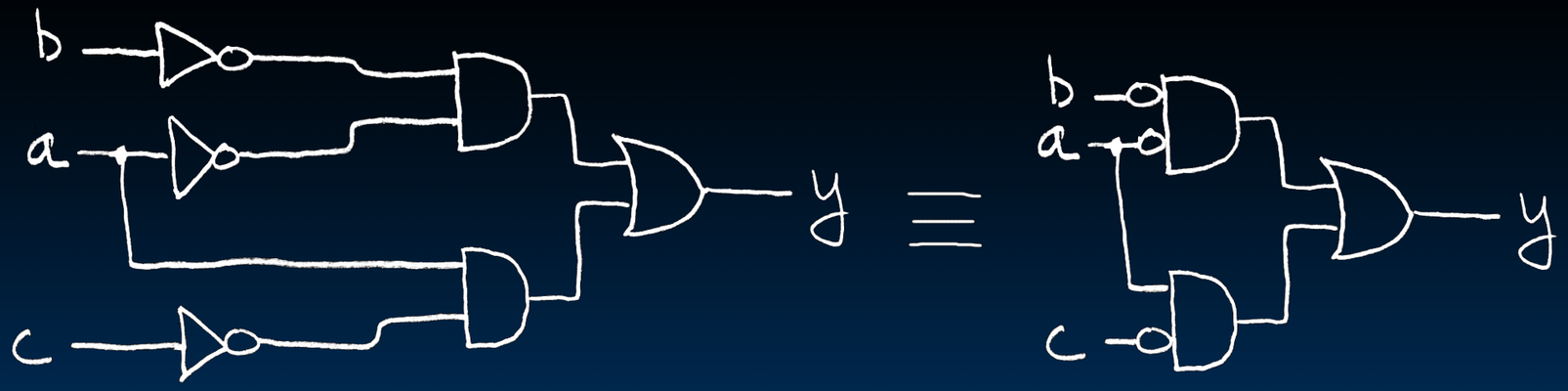
	abc	y
$\overline{a} \cdot \overline{b} \cdot \overline{c}$	000	1
$\overline{a} \cdot \overline{b} \cdot c$	001	1
	010	0
	011	0
$a \cdot \overline{b} \cdot \overline{c}$	100	1
	101	0
$a \cdot b \cdot \overline{c}$	110	1
	111	0

Sum-of-products (ORs of ANDs)

$$y = \overline{a}\overline{b}\overline{c} + \overline{a}\overline{b}c + a\overline{b}\overline{c} + ab\overline{c}$$

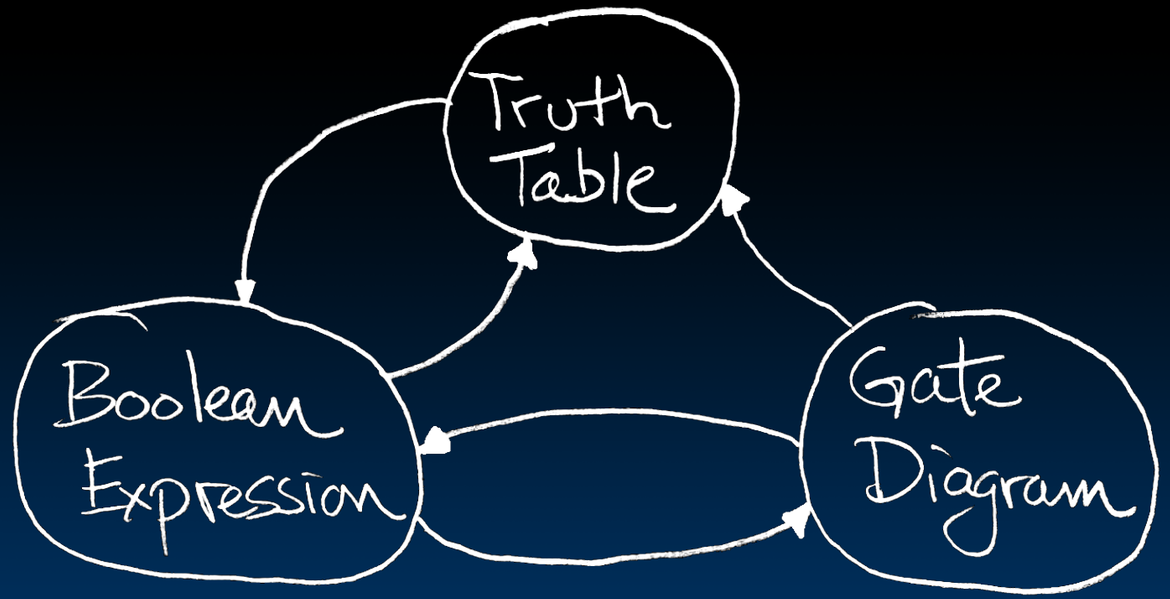
Canonical forms (2/2)

$$
\begin{aligned}
y &= \bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}c + a\bar{b}\bar{c} + ab\bar{c} \\
&= \bar{a}\bar{b}(\bar{c} + c) + a\bar{c}(\bar{b} + b) && \textit{distribution} \\
&= \bar{a}\bar{b}(1) + a\bar{c}(1) && \textit{complementarity} \\
&= \bar{a}\bar{b} + a\bar{c} && \textit{identity}
\end{aligned}
$$



“And In conclusion…”

- **Pipeline big-delay CL for faster clock**
- **Finite State Machines extremely useful**
  - You’ll see them again in (at least) 151A, 152 & 164
- **Use this table and techniques we learned to transform from 1 to another**



UC Berkeley
Teaching Professor
Dan Garcia

CS61C

Great Ideas in Computer Architecture

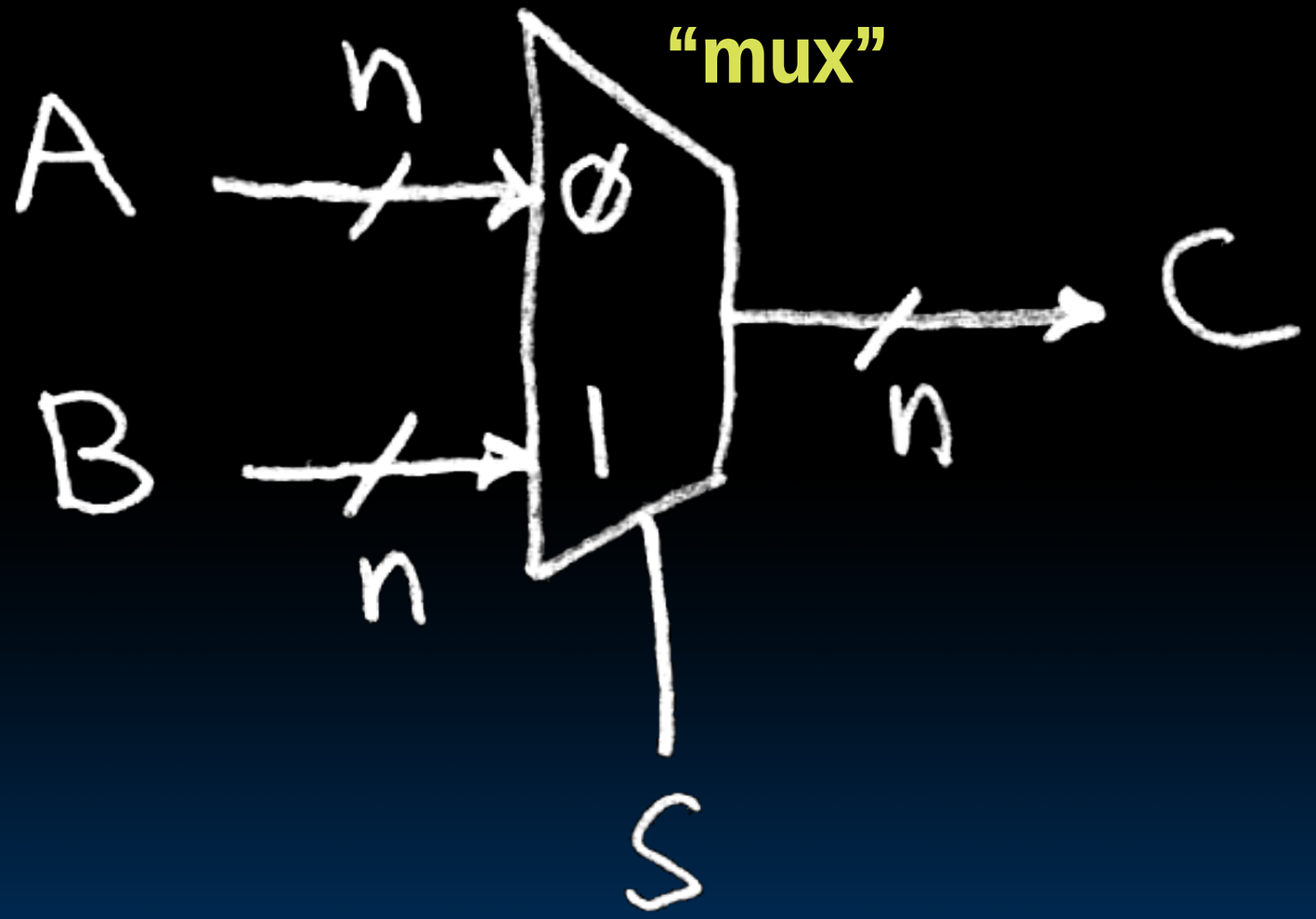
(a.k.a. Machine Structures)

UC Berkeley
Professor
Bora Nikolić

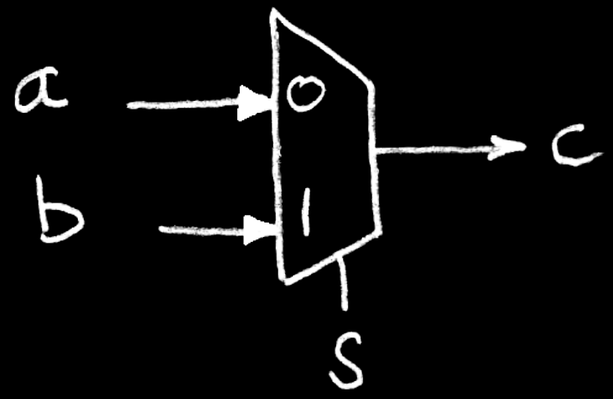
Combinational Logic Blocks

Data Multiplexors

Data Multiplexor (here 2-to-1, n-bit-wide)



N instances of 1-bit-wide mux



How many rows in TT?

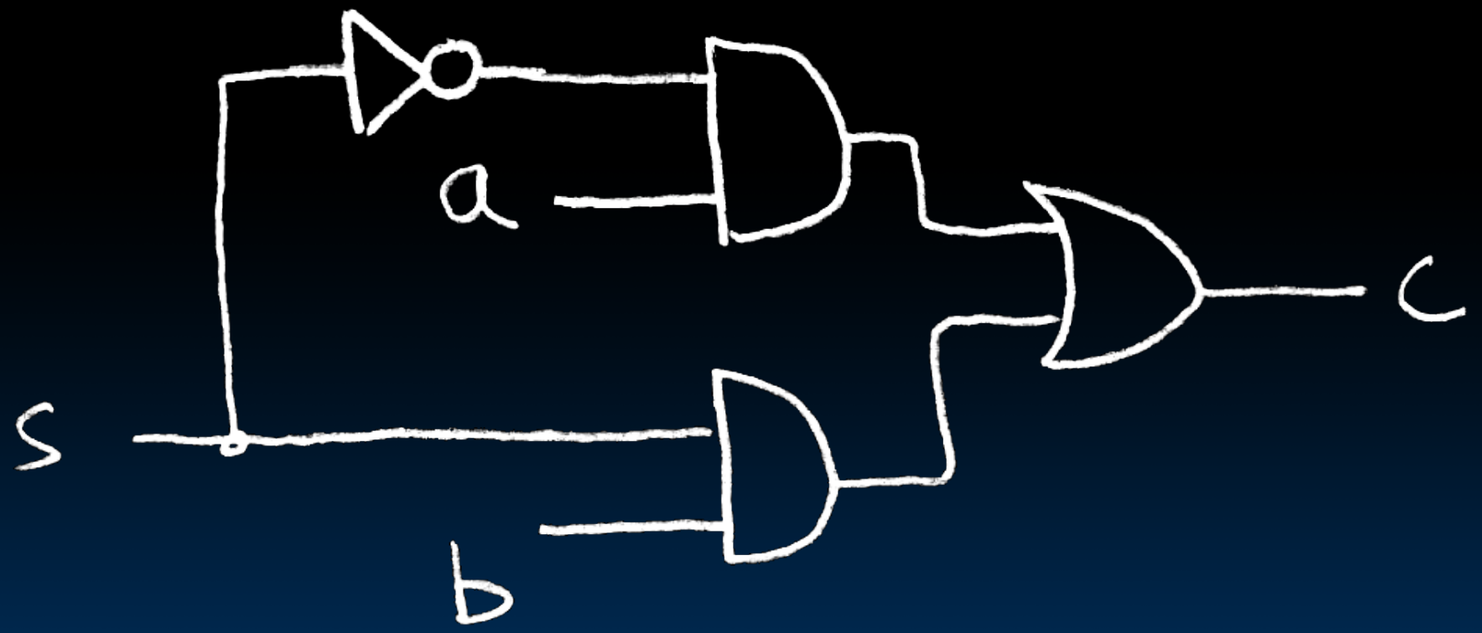
s	ab	c
0	00	0
0	01	0
0	10	1
0	11	1
1	00	0
1	01	1
1	10	0
1	11	1

$$
\begin{aligned}
c &= \overline{s}a\overline{b} + \overline{s}ab + s\overline{a}b + sab \\
&= \overline{s}(a\overline{b} + ab) + s(\overline{a}b + ab) \\
&= \overline{s}(a(\overline{b} + b)) + s((\overline{a} + a)b) \\
&= \overline{s}(a(1) + s((1)b) \\
&= \overline{s}a + sb
\end{aligned}
$$

s	c
0	a
1	b

How do we build a 1-bit-wide mux?

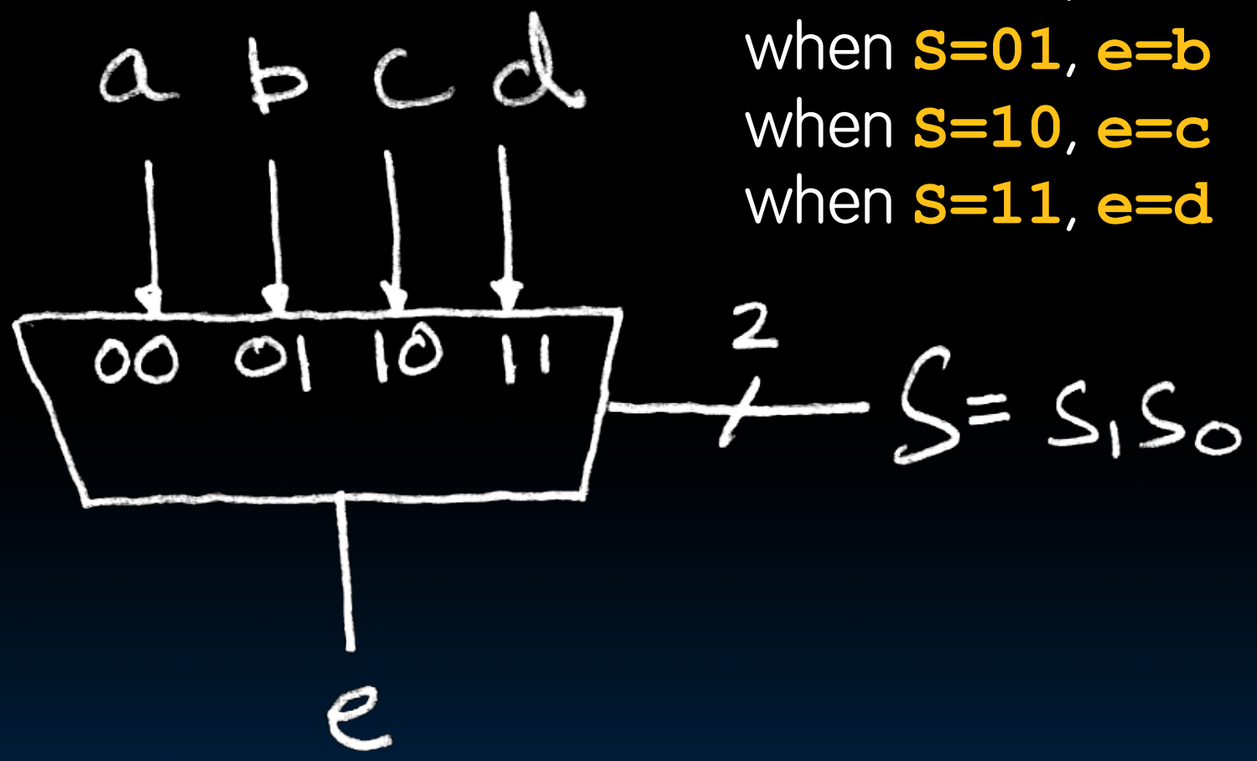
$$\overline{s}a + sb$$



4-to-1 Multiplexor?

- **How many rows in the Truth Table?**

when **S=00**, **e=a**
when **S=01**, **e=b**
when **S=10**, **e=c**
when **S=11**, **e=d**

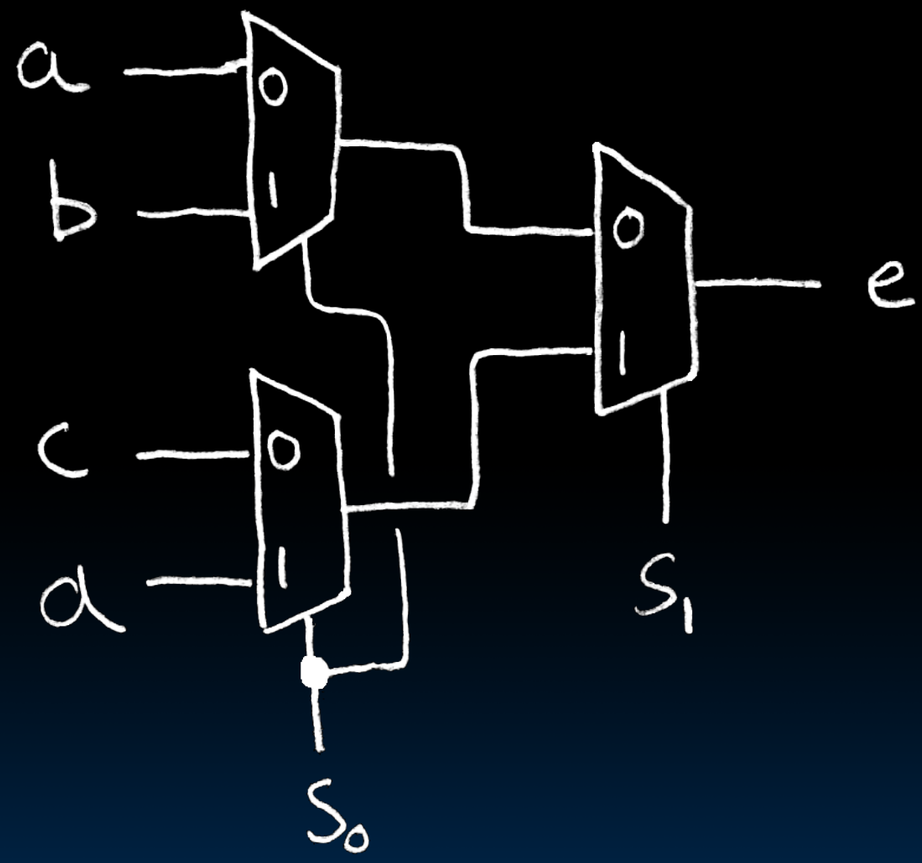


$s_1 s_0$	c
0 0	a
0 1	b
1 0	c
1 1	d

$$e = \overline{s_1} \cdot \overline{s_0} a + \overline{s_1} s_0 b + s_1 \overline{s_0} c + s_1 s_0 d$$

Mux: is there any other way to do it?

Hint: NCAA tourney!

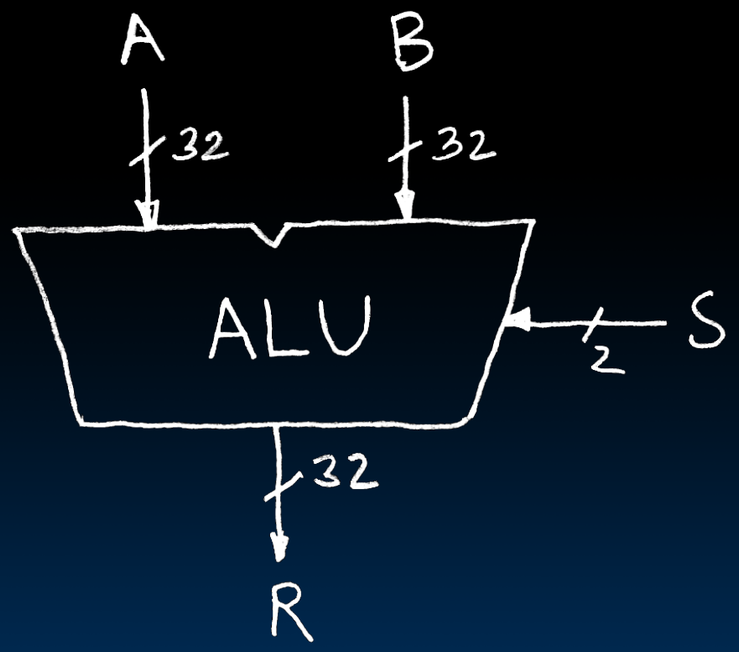


Ans: Hierarchically!

Arithmetic Logic Unit (ALU)

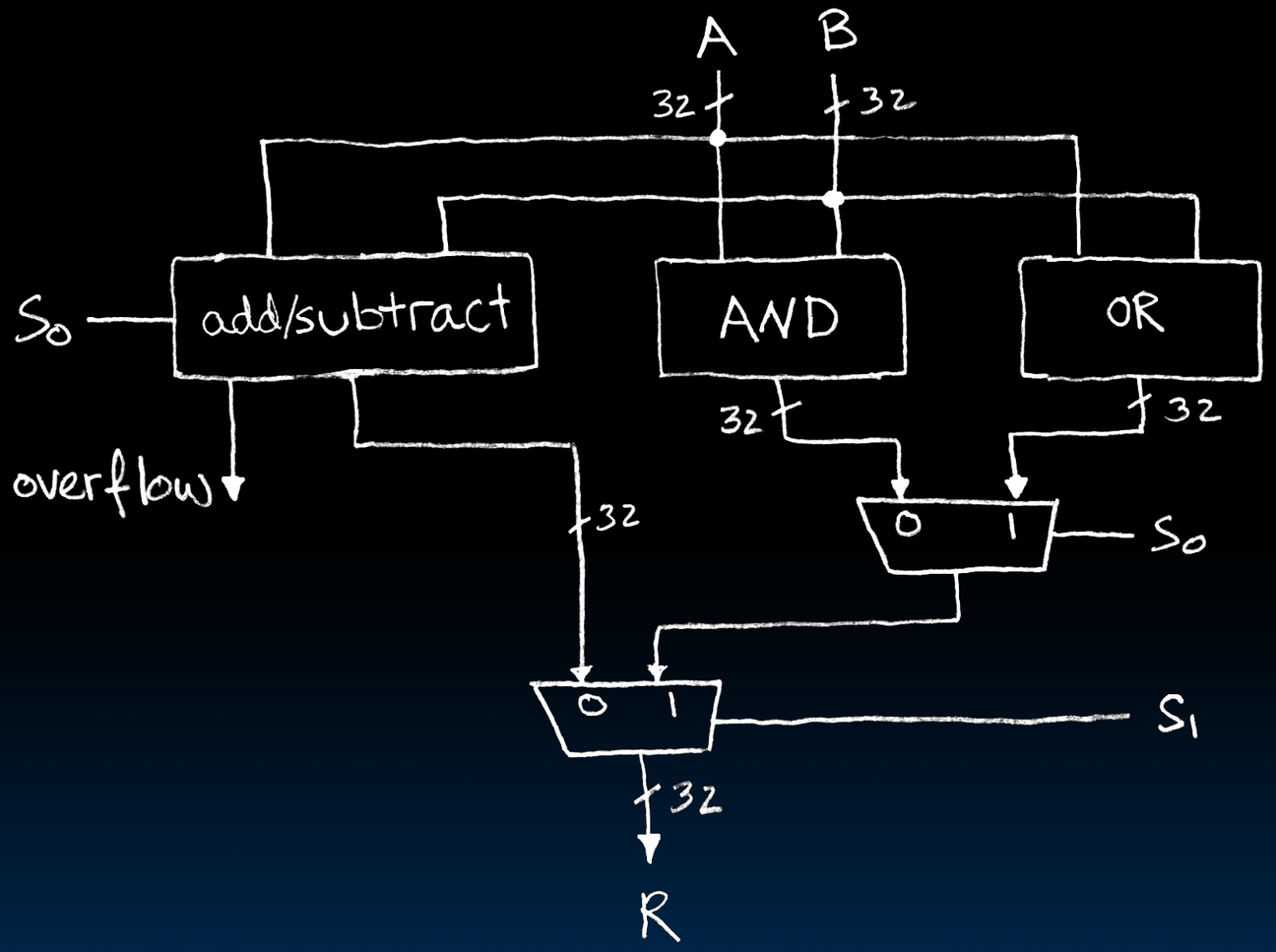
Arithmetic and Logic Unit

- Most processors contain a special logic block called "Arithmetic and Logic Unit" (ALU)
- We'll show you an easy one that does ADD, SUB, bitwise AND (`&`), bitwise OR (`|`)



when `S=00`, `R=A+B`
when `S=01`, `R=A-B`
when `S=10`, `R=A&B`
when `S=11`, `R=A|B`

Our simple ALU



Adder / Subtractor

Adder / Subtracter Design – how?

- Truth-table, then determine canonical form, then minimize and implement as we've seen before
- Look at breaking the problem down into smaller pieces that we can cascade or hierarchically layer

Adder / Subtractor – One-bit adder LSB…

$$
\begin{array}{cccc|c}
 & a_3 & a_2 & a_1 & a_0 \\
+ & b_3 & b_2 & b_1 & b_0 \\
\hline
 & s_3 & s_2 & s_1 & s_0
\end{array}
$$

a_0	b_0	s_0	c_1
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$s_0 = a_0 \text{ XOR } b_0$$
$$c_1 = a_0 \text{ AND } b_0$$

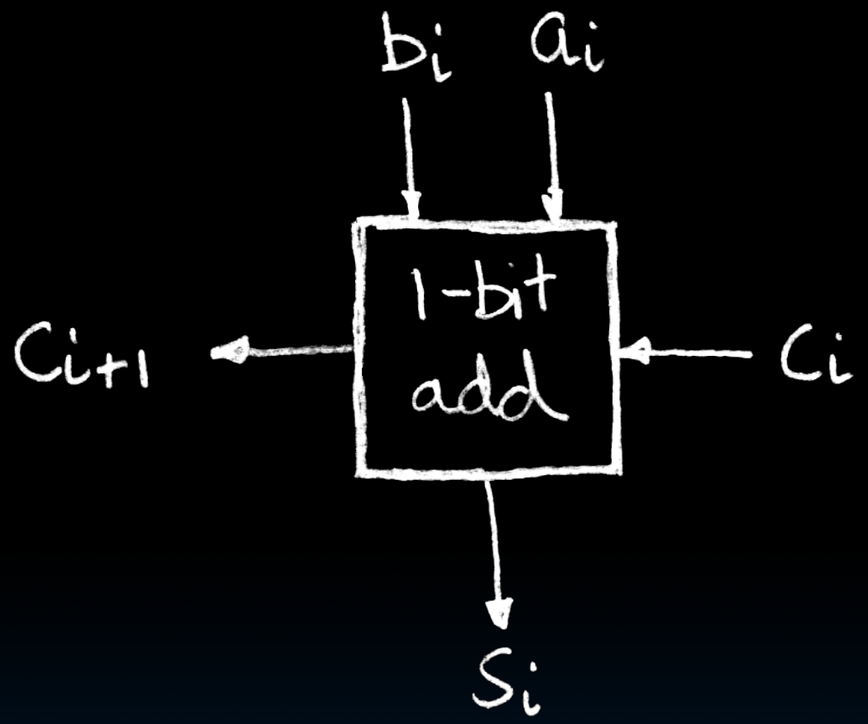
Adder / Subtractor – One-bit adder (1/2)...

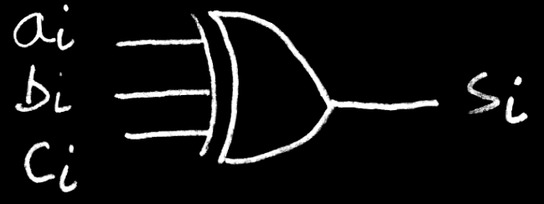
$$\begin{array}{rcccc} & a_3 & a_2 & \boxed{a_1} & a_0 \\ + & b_3 & b_2 & \boxed{b_1} & b_0 \\ \hline & s_3 & s_2 & \boxed{s_1} & s_0 \end{array}$$

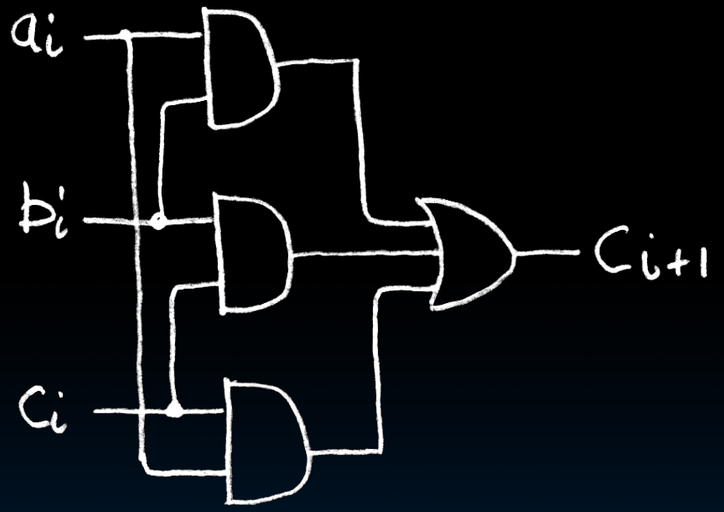
a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$\begin{aligned} s_i &= \mathrm{XOR}(a_i, b_i, c_i) \\ c_{i+1} &= \mathrm{MAJ}(a_i, b_i, c_i) = a_i b_i + a_i c_i + b_i c_i \end{aligned}$$

Adder / Subtractor – One-bit adder (2/2)…

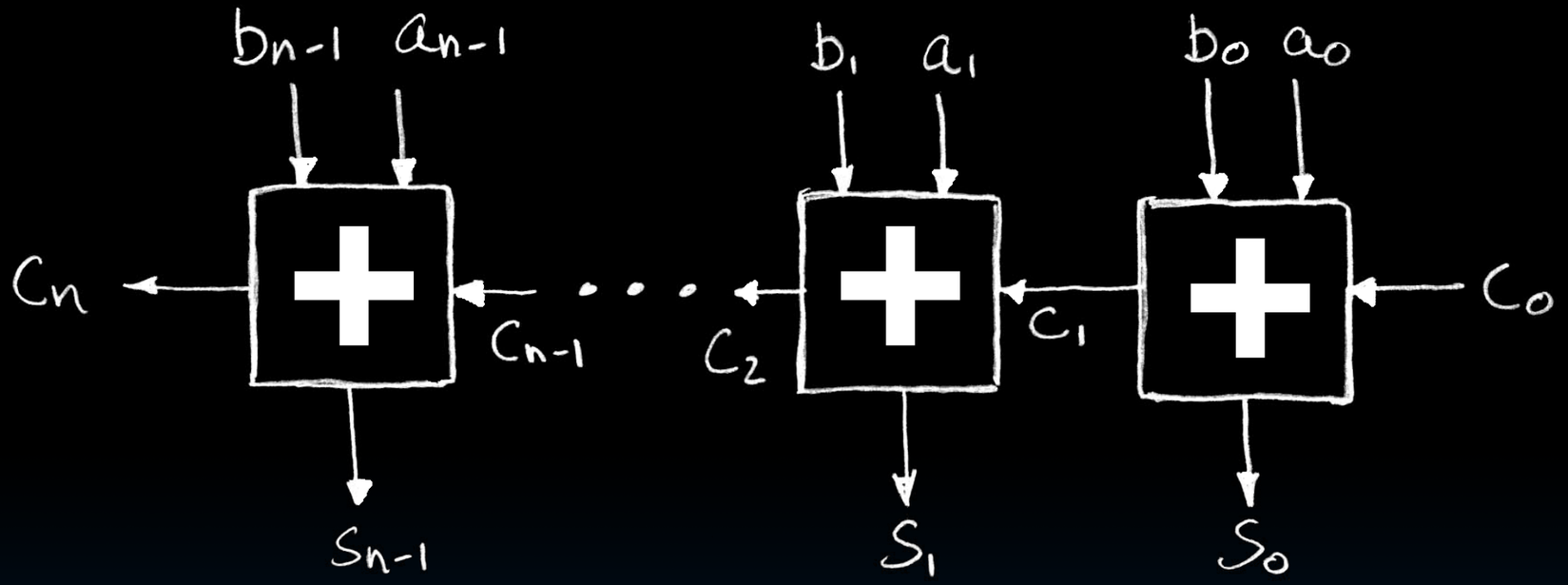






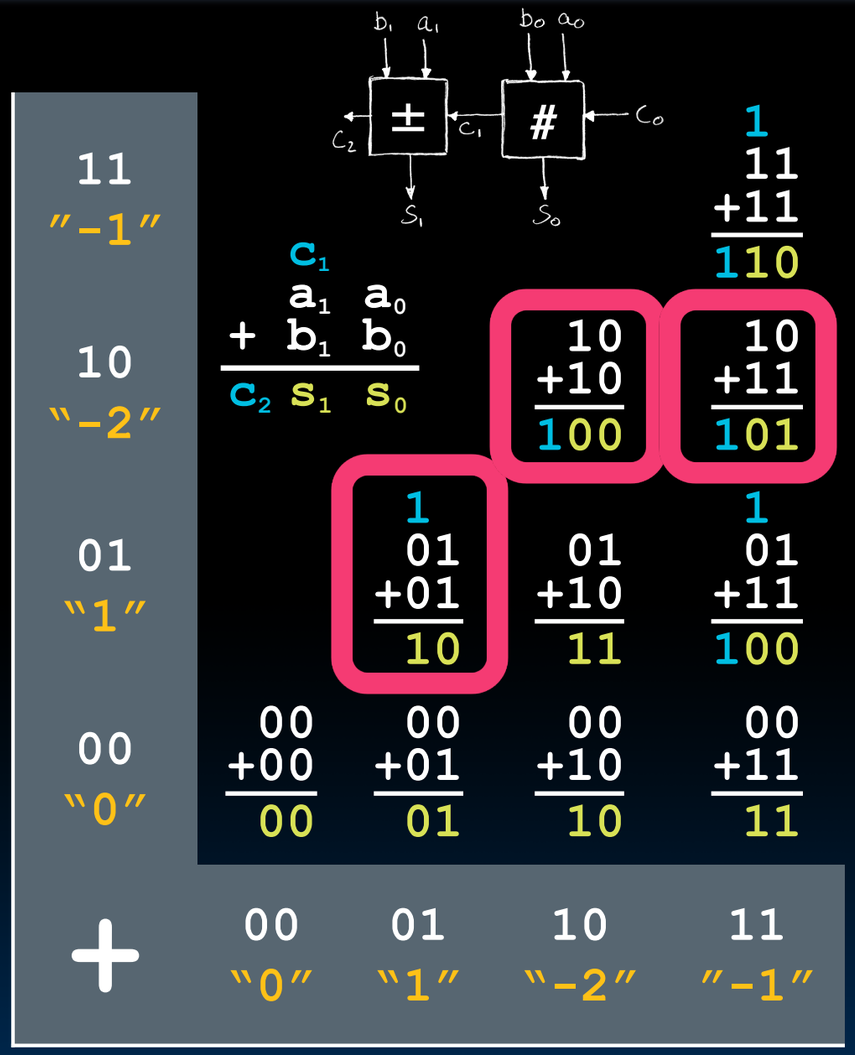
$$s_i = \mathrm{XOR}(a_i, b_i, c_i)$$
$$c_{i+1} = \mathrm{MAJ}(a_i, b_i, c_i) = a_i b_i + a_i c_i + b_i c_i$$

N 1-bit adders → 1 N-bit adder



What about overflow?
Overflow = c_n?

Sum of two 2-bit numbers…



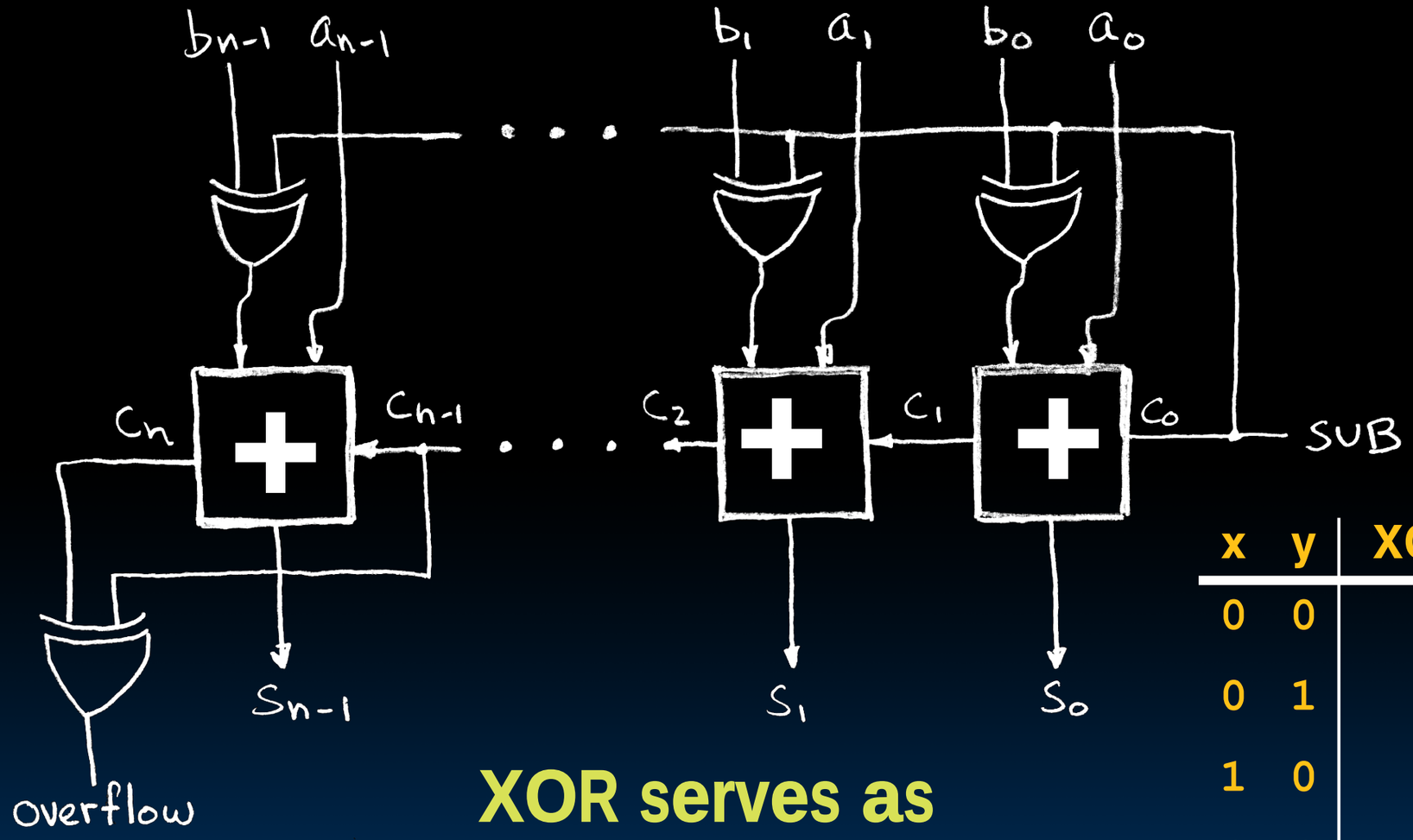
- **Let's add**
  - First unsigned
  - Then signed (Two's Complement)
    - When do the lowest 2 bits of sum not represent correct sum?
    - Is there a pattern of when this happens? Hint: Check out the carry-bit and the sum-4s-column-bit
- **Highest adder**
  - C_{in} = Carry-in = c_1, C_{out} = Carry-out = c_2
  - No C_{out} or C_{in} → NO overflow!
  - C_{in} and C_{out} → NO overflow!
  - C_{in}, but no C_{out} → A,B both > 0, overflow!
  - C_{out}, but no C_{in} → A or B are -2, overflow!

What operation is this?

$$\text{overflow} = c_n \text{ XOR } c_{n-1}$$

Subtractor Design

Extremely Clever Subtractor: A-B = A + (-B)



x	y	XOR(x,y)
0	0	0
0	1	1
1	0	1
1	1	0

XOR serves as conditional inverter!

"And In conclusion…"

- **Use muxes to select among input**
  - S input bits selects 2^S inputs
  - Each input can be n-bits wide, indep of S
- **Can implement muxes hierarchically**
- **ALU can be implemented using a mux**
  - Coupled with basic block elements
- **N-bit adder-subtractor done using N 1-bit adders with XOR gates on input**
  - XOR serves as conditional inverter