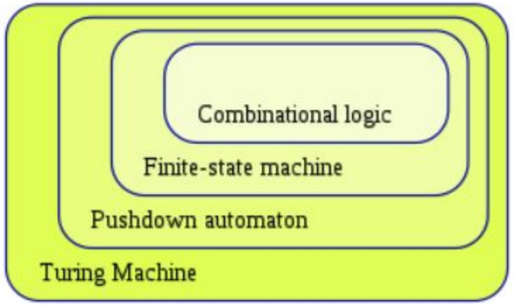
Lecture

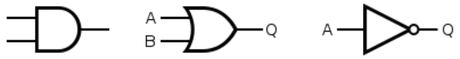


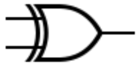
- Last time:
  - Boolean Algebra ⇔ Digital Circuits
    - Point: We can do a lot with just Combinational logic -- all true functions can be evaluated
    - Point: Digital Circuits can be built to evaluate all of these functions.
  - All we need is And (*), Or (+) and Not (')
  - Truth Table → Boolean Algebra
  - Boolean Algebra → Circuits → Boolean Algebra
  - Minimization of Circuits
    - Algebraic Transformations:
    - Karnaugh Maps

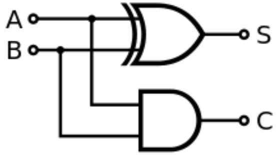
- Today: More Combinational Circuits

Combinational Logic

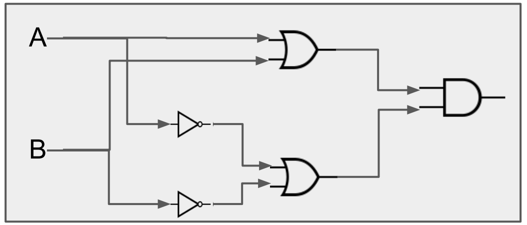
- Using just is tedious: AND (*), OR (+), NOT (')
- Solution: Build components and reuse!
  - XOR: A ⊕ B is equivalent to (A + B) * (A' + B')
  - Half-Adder: S = A ⊕ B, C = A * B
- Bigger components and with more bits!
  - Binary Addition
  - Binary Subtraction
  - BCD Addition
  - Decoder
  - Multiplexer







XOR:



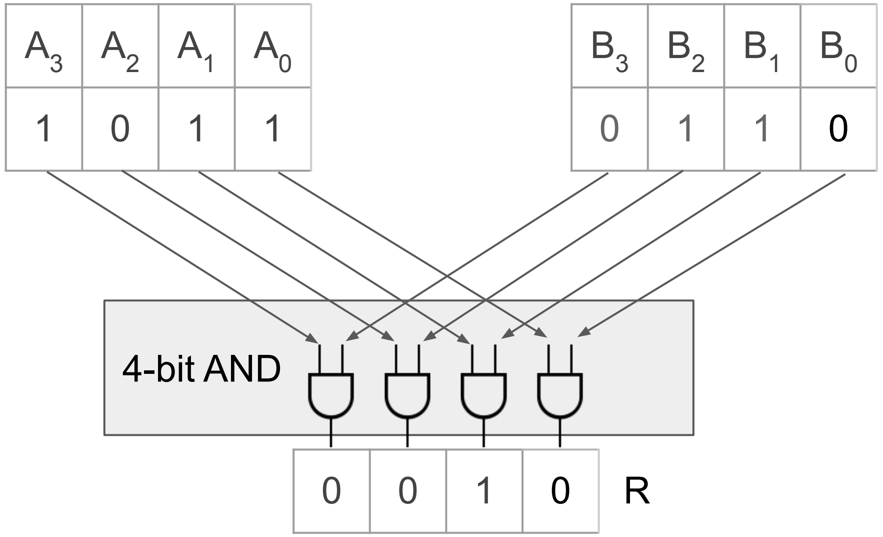
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

4-bit Bitwise AND

- R = A & B

	1	0	1	1	A
&	0	1	1	0	B
	0	0	1	0	R

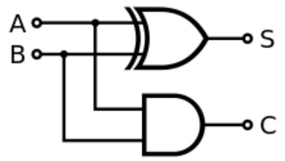
- For a n-bit operation,
    - create n-duplicates of the base circuitry
    - layout duplicates in parallel
    - package it up



1-bit Binary Addition

- Recall:
  - A + B → S, C

			1	1	A
+			1	0	B
					S



A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

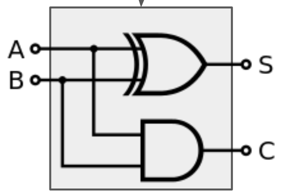
4-bit Binary Addition

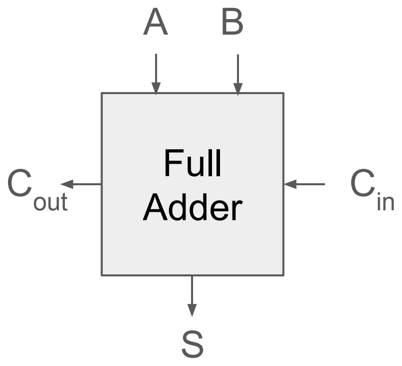
- Recall:
  - $C_{in} + A_x + B_x \rightarrow C_{out}, \; S_{x+1}$

	1	1	0	0	
	1	0	1	1	A
+	0	1	1	0	B
1	0	0	0	1	S

- Half-Adder is not sufficient!
  We need a Full-Adder

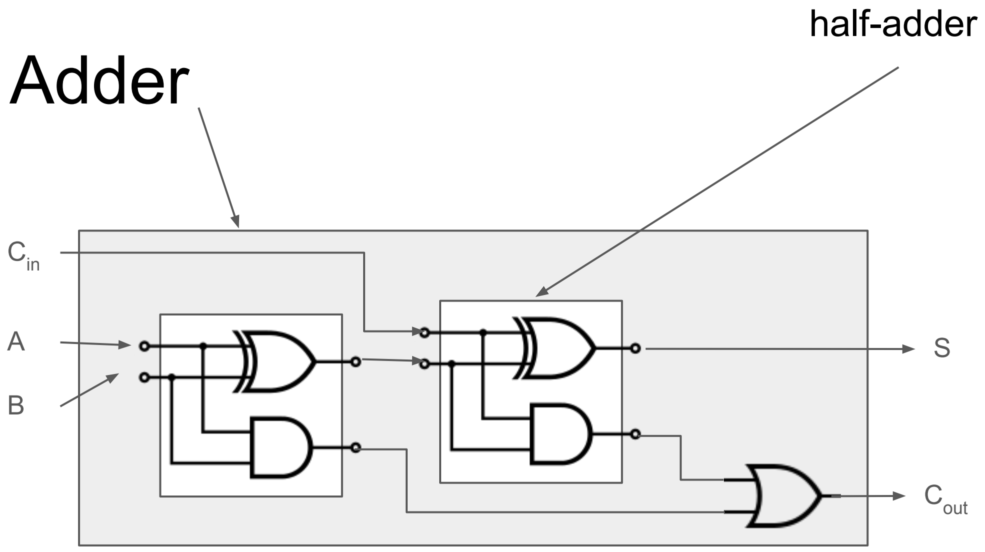
half-adder





C_{in}	A	B	C_{out}	S
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

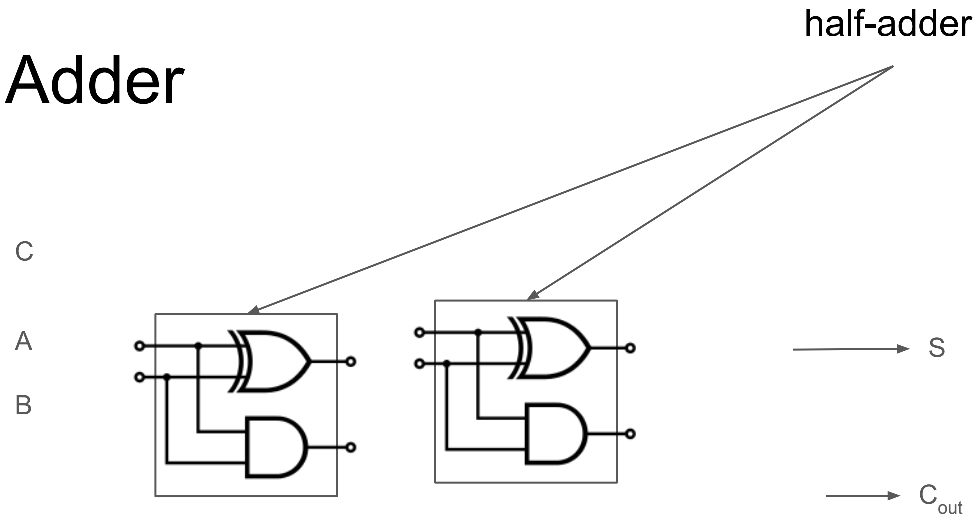


Got It? Any Questions

C_{in}	A	B	C_{out}	S
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

- $C_{out} = AB + C_{in}(A \oplus B)$
- $S = C_{in} \oplus A \oplus B$

Full Adder



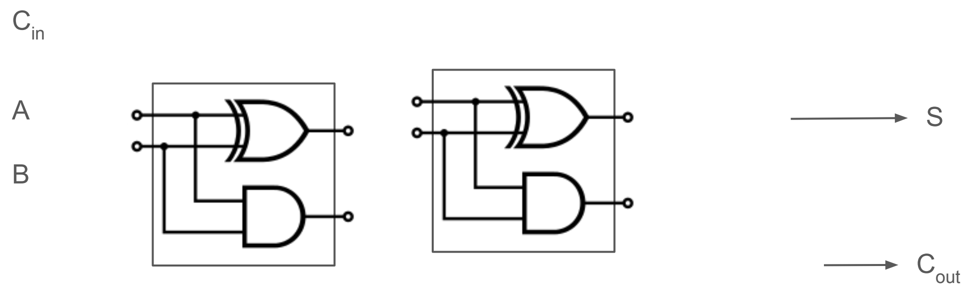
- C_{out} = C'AB +

Note: Renamed C_{in} to be C

C	A	B	C_{out}	S	
0	0	0	0	0	
0	0	1	0	1	
0	1	0	0	1	
0	1	1	1	0	C'AB
1	0	0	0	1	
1	0	1	1	0	
1	1	0	1	0	
1	1	1	1	1	

Use Sum of Products

Full Adder

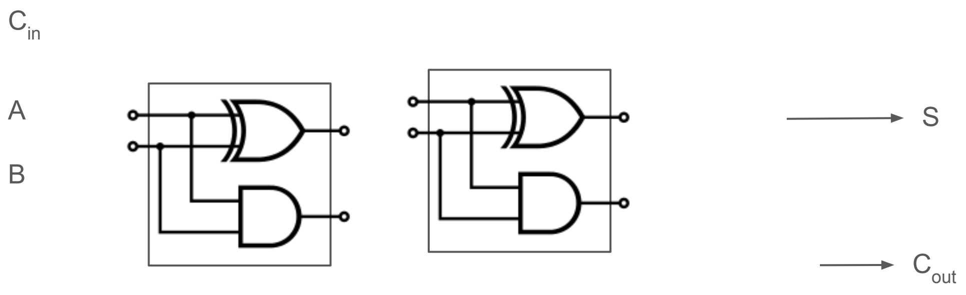


C	A	B	C_{out}	S	
0	0	0	0	0	
0	0	1	0	1	
0	1	0	0	1	
0	1	1	1	0	C'AB
1	0	0	0	1	
1	0	1	1	0	CA'B
1	1	0	1	0	CAB'
1	1	1	1	1	CAB

Use Sum of Products

- C_{out} = C'AB + CA'B + CAB' + CAB

Full Adder



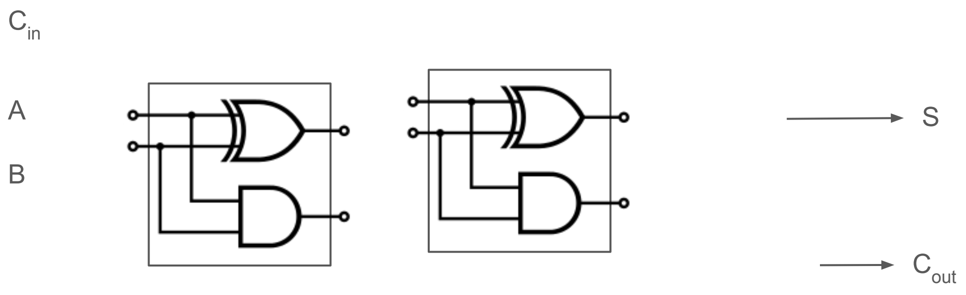
C	A	B	C_{out}	S	
0	0	0	0	0	
0	0	1	0	1	
0	1	0	0	1	
0	1	1	1	0	C'AB
1	0	0	0	1	
1	0	1	1	0	CA'B
1	1	0	1	0	CAB'
1	1	1	1	1	CAB

- $C_{out} = C'AB + \boxed{CA'B + CAB'} + \boxed{CAB}$

➢ $C_{out} = C'AB + CAB + CA'B + CAB'$

Use Commutative Property

Full Adder



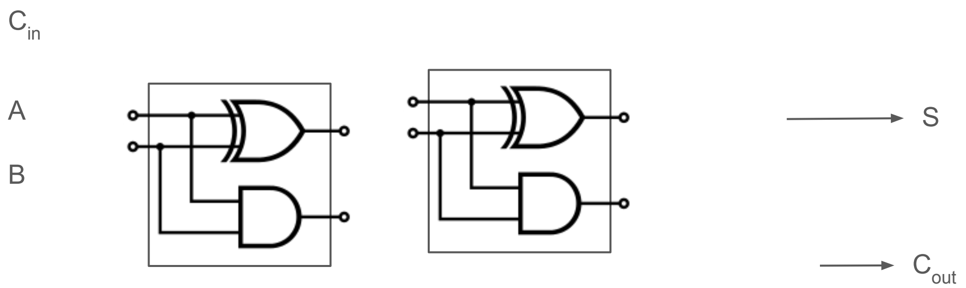
C	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

C'AB
CA'B
CAB'
CAB

- $C_{out} = \boxed{C'AB} + \boxed{CAB} + CA'B + CAB'$
  - ➢ $C_{out} = (C' + C)AB + CA'B + CAB'$

Use Distributive Property

Full Adder



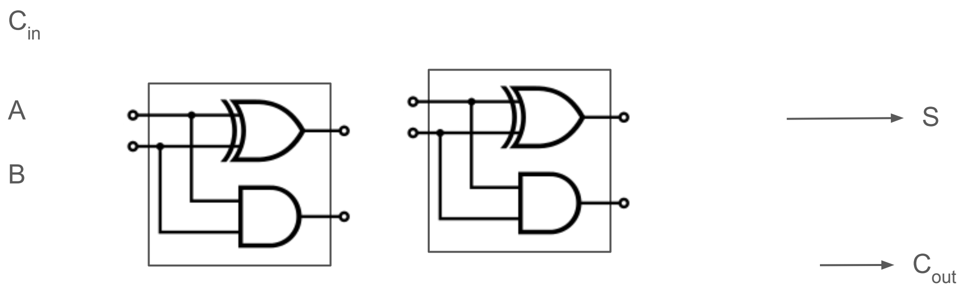
C	A	B	C_{out}	S	
0	0	0	0	0	
0	0	1	0	1	
0	1	0	0	1	
0	1	1	1	0	C'AB
1	0	0	0	1	
1	0	1	1	0	CA'B
1	1	0	1	0	CAB'
1	1	1	1	1	CAB

- $C_{out} = (\boxed{C' + C})AB + CA'B + CAB'$

➢ $C_{out} = (\text{true})AB + CA'B + CAB'$

Use Complement Property

Full Adder



C	A	B	C_{out}	S
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

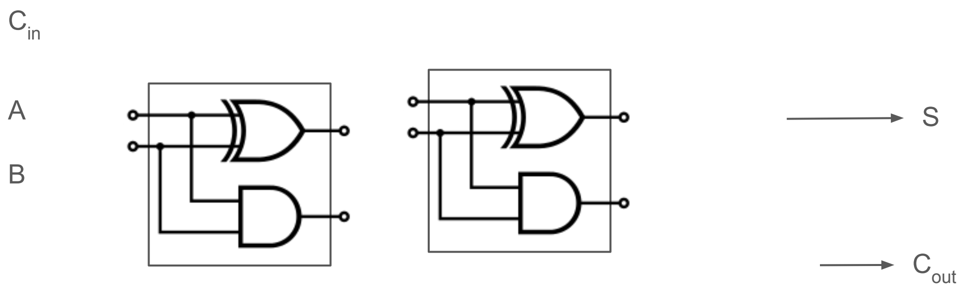
C'AB
CA'B
CAB'
CAB

- C_{out} = $\boxed{\text{(true)AB}}$ + CA'B + CAB'

➢ C_{out} = AB + CA'B + CAB'

Use Identity Property

Full Adder



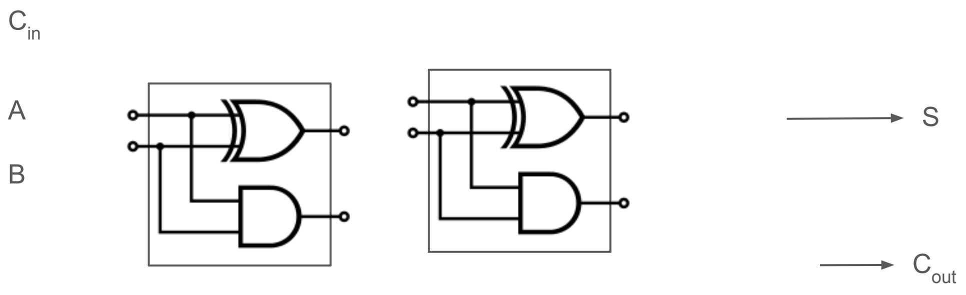
C	A	B	C_{out}	S	
0	0	0	0	0	
0	0	1	0	1	
0	1	0	0	1	
0	1	1	1	0	C'AB
1	0	0	0	1	
1	0	1	1	0	CA'B
1	1	0	1	0	CAB'
1	1	1	1	1	CAB

- $C_{out} = AB + \boxed{CA'B + CAB'}$

➢ $C_{out} = AB + C(A'B + AB')$

Use Distributive Property

Full Adder



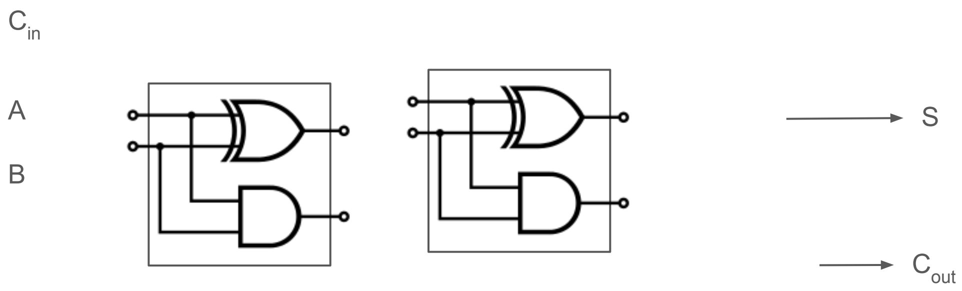
C	A	B	C_{out}	S	
0	0	0	0	0	
0	0	1	0	1	
0	1	0	0	1	
0	1	1	1	0	C'AB
1	0	0	0	1	
1	0	1	1	0	CA'B
1	1	0	1	0	CAB'
1	1	1	1	1	CAB

$A \oplus B \Leftrightarrow A'B + AB'$

- $C_{out} = AB + \boxed{C(A'B + AB')}$

➢ $C_{out} = AB + C(A \oplus B)$

Full Adder



C	A	B	C_{out}	S	
0	0	0	0	0	
0	0	1	0	1	C'A'B
0	1	0	0	1	C'AB'
0	1	1	1	0	
1	0	0	0	1	CA'B'
1	0	1	1	0	
1	1	0	1	0	
1	1	1	1	1	CAB

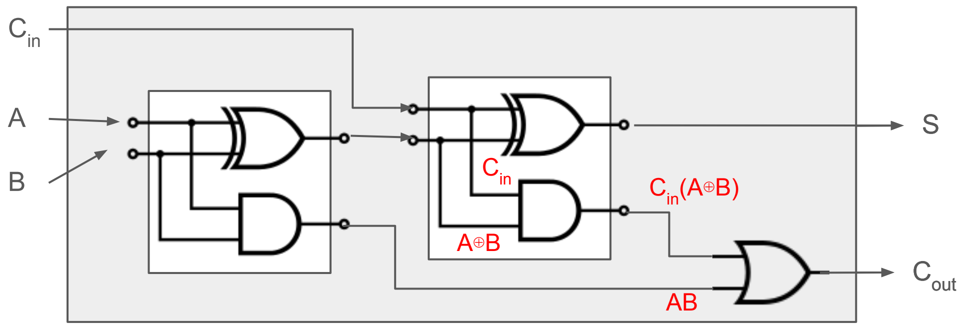
- $C_{out} = AB + C(A \oplus B)$
- $S \quad = C \oplus A \oplus B$

Sum of Products:

$$
\begin{aligned}
& C'A'B + C'AB' + CA'B' + CAB \\
&= C'(A'B + AB') + C(A'B' + AB) \\
&= C'(A \oplus B) + C(A \oplus B)' \\
&= C \oplus (A \oplus B) \\
&= C \oplus A \oplus B
\end{aligned}
$$

$A \oplus B \Leftrightarrow A'B + AB'$

Full Adder



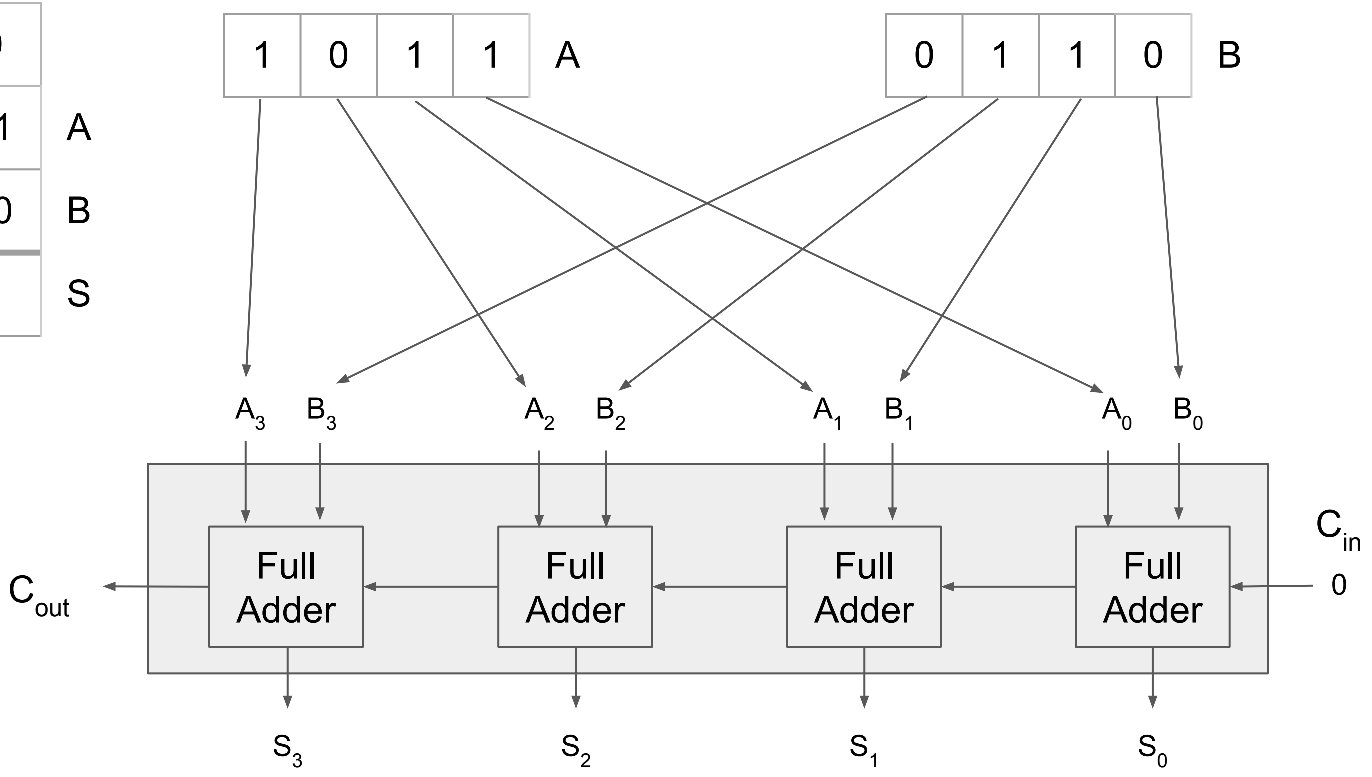
C_{in}	A	B	C_{out}	S	
0	0	0	0	0	
0	0	1	0	1	
0	1	0	0	1	
0	1	1	1	0	
1	0	0	0	1	
1	0	1	1	0	
1	1	0	1	0	
1	1	1	1	1	

Note: Renamed C to be C_{in}

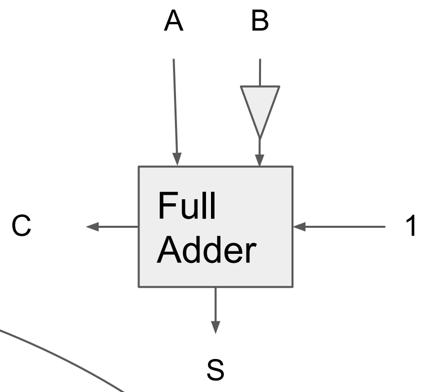
- $C_{out} = AB + C_{in}(A \oplus B)$
- $S = C_{in} \oplus A \oplus B$

4-bit Binary Addition (aka: 4-bit Full Adder)

	1	1	0	0	
	1	0	1	1	A
+	0	1	1	0	B
1	0	0	0	1	S



Binary Subtractor

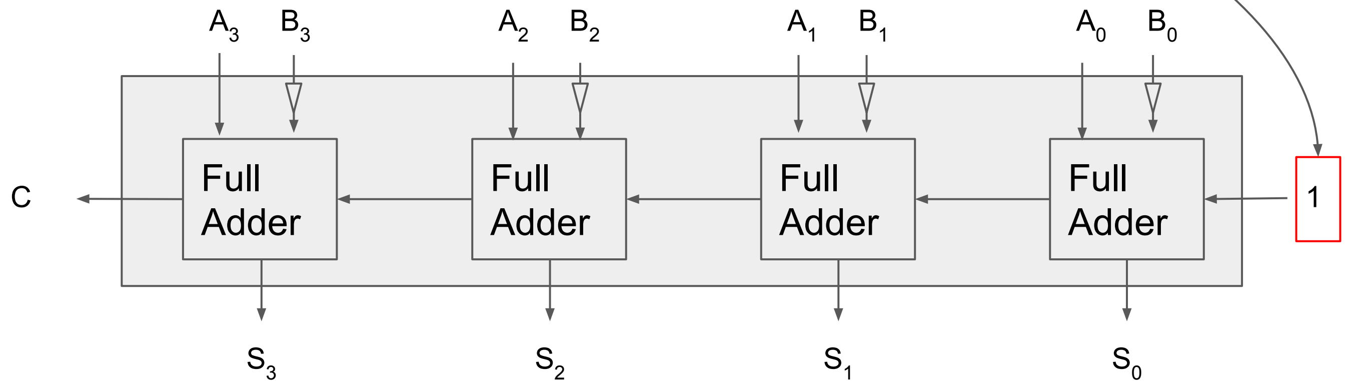


- Recall: A - B

  = A + (2's complement of B)

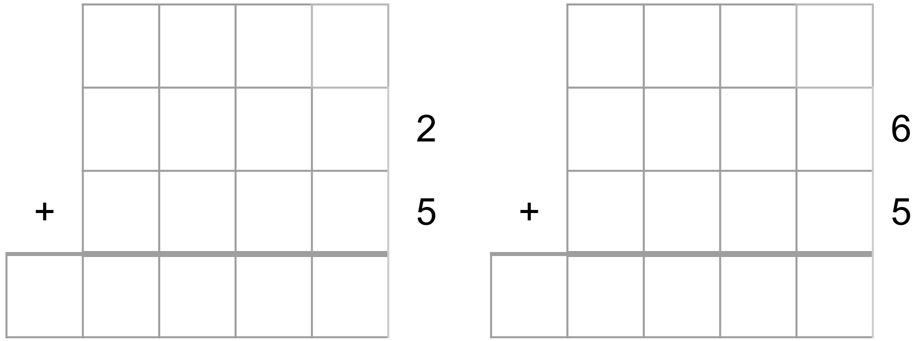
  = A + (1's complement of B) + $\boxed{1}$

- 4-bit Binary Subtractor



4-bit BCD Addition

- BCD: 26 + 55



#	Encoding $S_3S_2S_1S_0$		Encoding $S_3S_2S_1S_0$
0	0 0 0 0	8	1 0 0 0
1	0 0 0 1	9	1 0 0 1
2	0 0 1 0		1 0 1 0
3	0 0 1 1		1 0 1 1
4	0 1 0 0	invalid	1 1 0 0
5	0 1 0 1		1 1 0 1
6	0 1 1 0		1 1 1 0
7	0 1 1 1		1 1 1 1

- Perform Regular Binary Addition, but account for the invalid patterns
- Add six upon whenever you are in the deadzone or there is overflow
  - Invalid = $S_3 * (S_2 + S_1)$
  - Overflow = C_{out}

CISC -versus- RISC

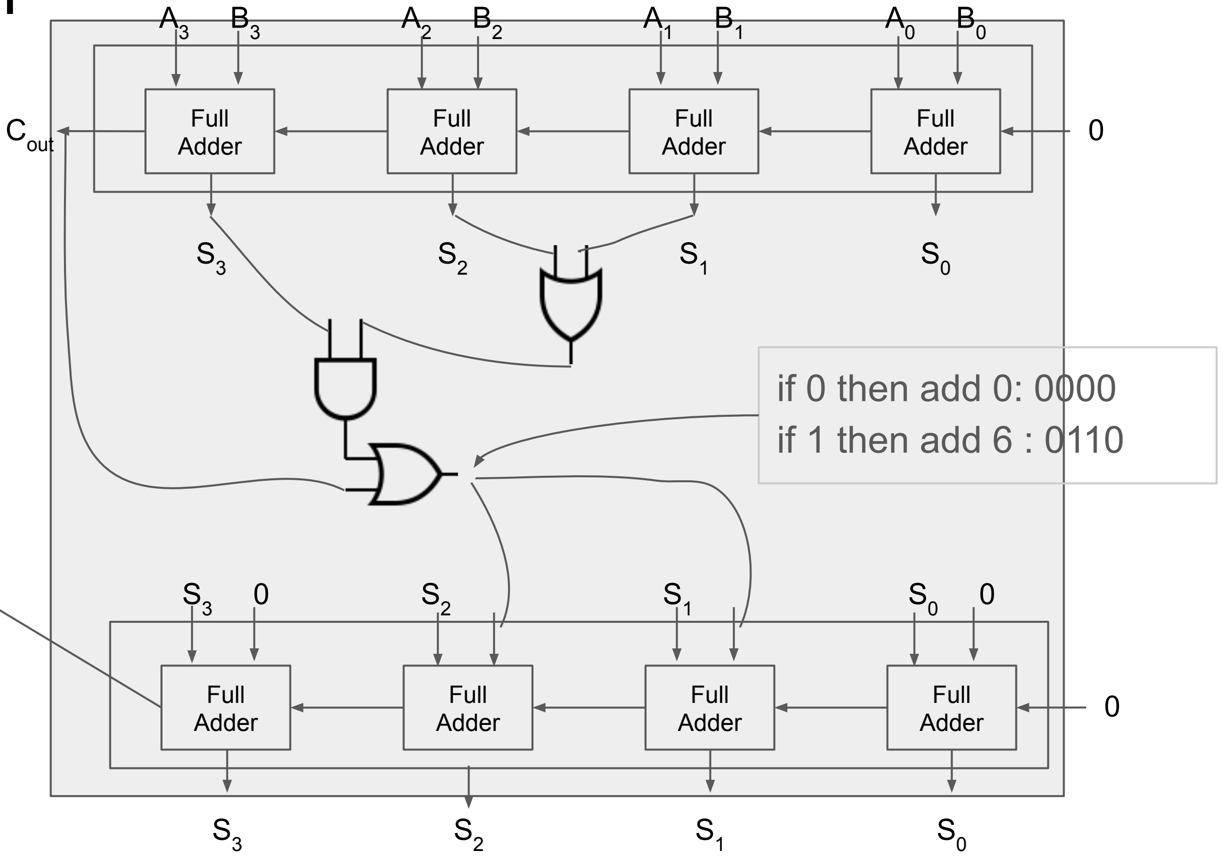
Deadzone $= S_3 * (S_2 + S_1)$

Overflow $= C_{out}$

4-bit BCD Addition

	1	0	0	0	
	0	1	1	0	6
+	0	1	0	1	5
1	0	0	0	1	

	1	1	1	1	
	0	0	1	0	2
+	0	1	0	1	5
0	1	0	0	0	

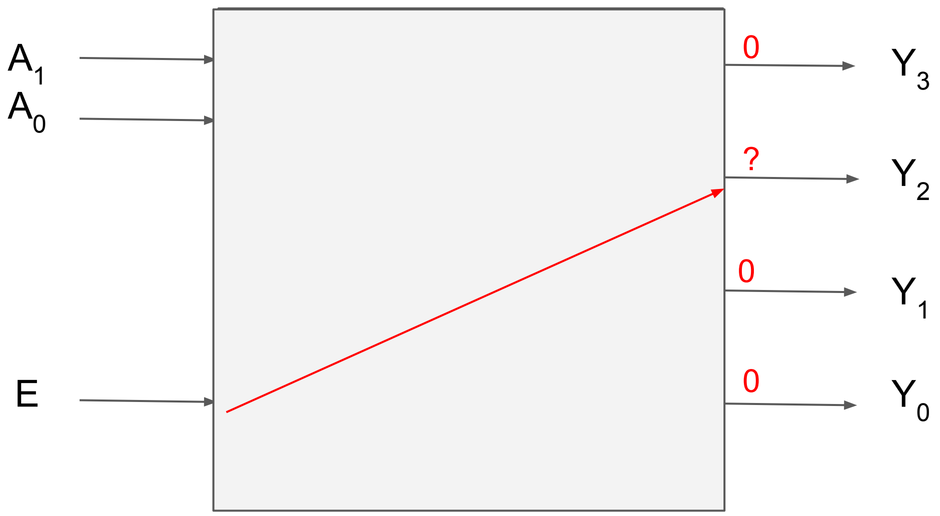


if 0 then add 0: 0000
if 1 then add 6 : 0110

Decoder: 2-to-4

- N+1 Inputs:
  - E: A single data line, which is referred to as "Enable"
  - A: Think of it as a binary number
- 2^n Outputs: A output line is set

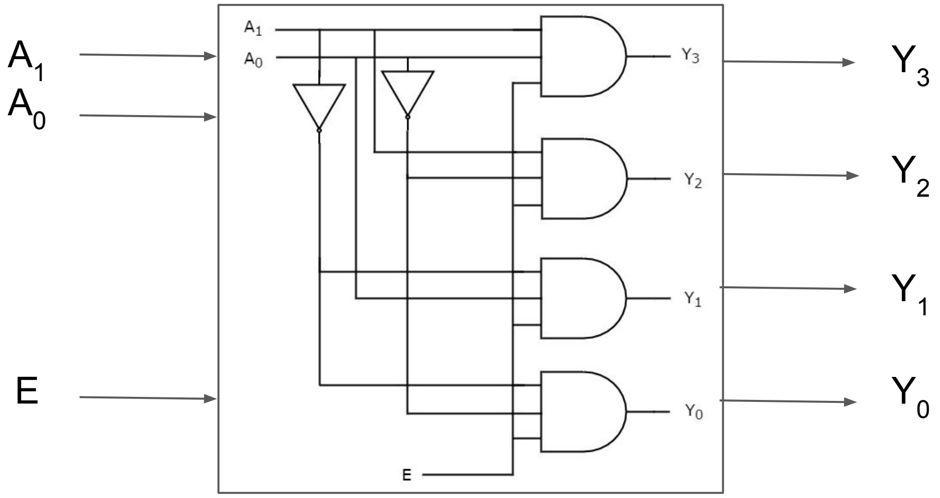
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	X	X	0	0	0	0
1	0	0				1
1	0	1			1	
1	1	0		1		
1	1	1	1			

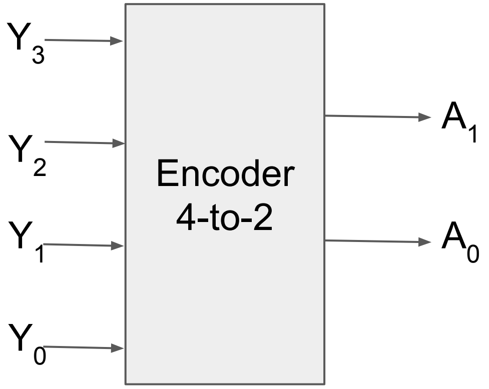


Decoder: 2-to-4, 3-to-8, 4-to-16, 5-to-32

- N+1 Inputs:
    - E: An "enable" line to active the circuit
    - A: Think of it as a binary number
- 2^n Outputs: A output line is set

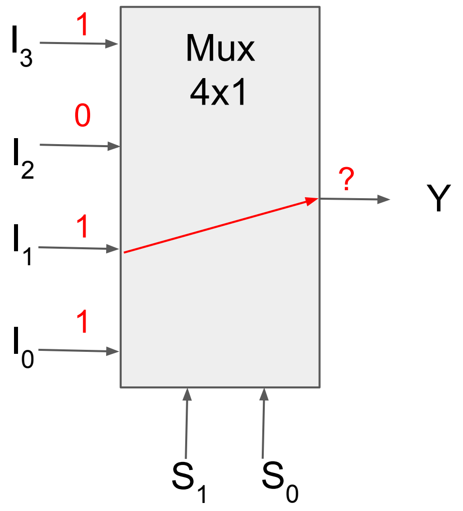
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	X	X	0	0	0	0
1	0	0				1
1	0	1			1	
1	1	0		1		
1	1	1	1			





Multiplexer (Data Selector)

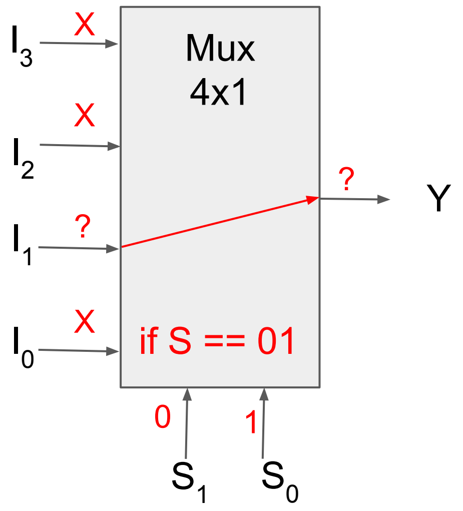
- Inputs:
  - 2^N data input lines
  - N selector lines
- Outputs:
  - 1 dataline

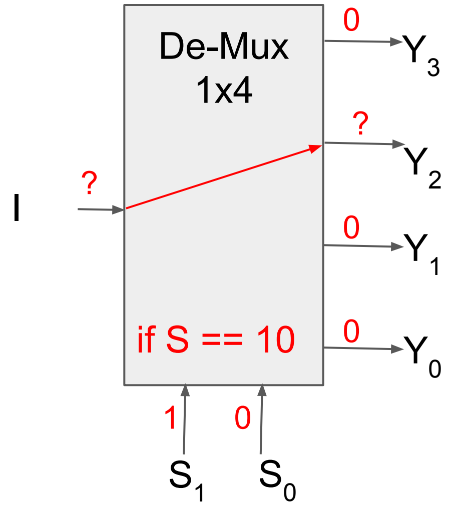


S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Multiplexer (Data Selector)

- Inputs:
  - 2^N data input lines
  - N selector lines
- Outputs:
  - 1 dataline


Mux
4x1
I_3
X
I_2
X
I_1
?
I_0
X
?
Y
if S == 01
0
1
S_1
S_0


De-Mux
1x4
I
?
0
Y_3
?
Y_2
0
Y_1
0
Y_0
if S == 10
1
0
S_1
S_0