Encoder:

- A n-to-m-line *encoder* is a combinational circuit that converts information from *n* input lines to a minimum of m = $\lceil log_2 n \rceil$ output lines.

4:2 Encoders:

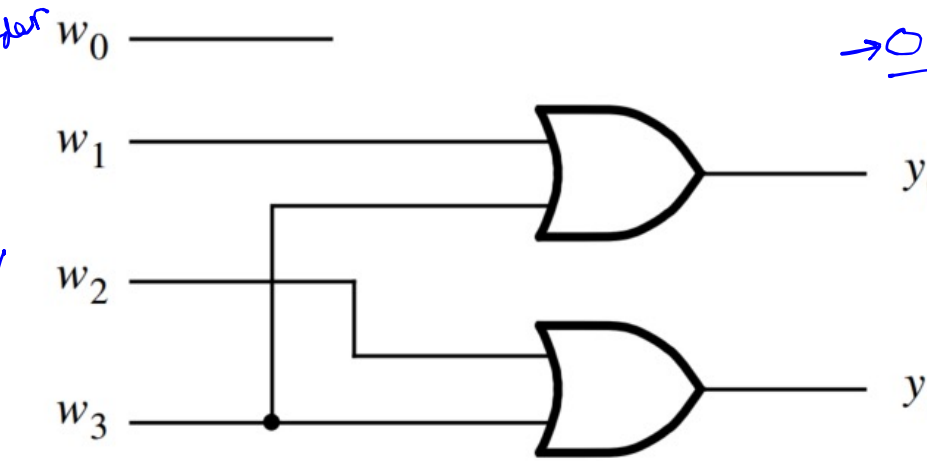
w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

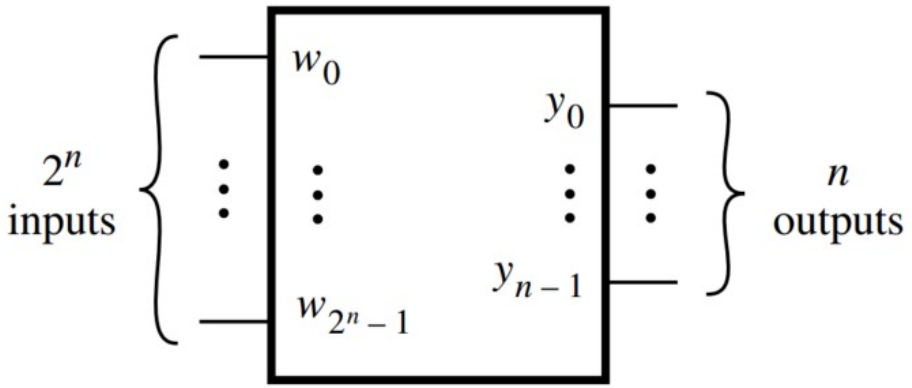
w_3w_2 \ w_1w_0	ϕ	0	ϕ	0
1	ϕ	ϕ	ϕ	
ϕ	ϕ	ϕ	ϕ	
1	ϕ	ϕ	ϕ	

$$y_1 = w_3 + w_2$$

w_3w_2 \ w_1w_0	ϕ	0	ϕ	1
0	ϕ	ϕ	ϕ	
ϕ	ϕ	ϕ	ϕ	
1	ϕ	ϕ	ϕ	

$$y_0 = w_3 + w_1$$



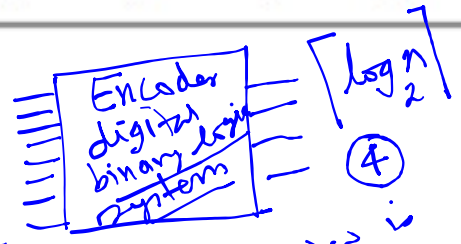


8:3 Octal - Binary Encoder:

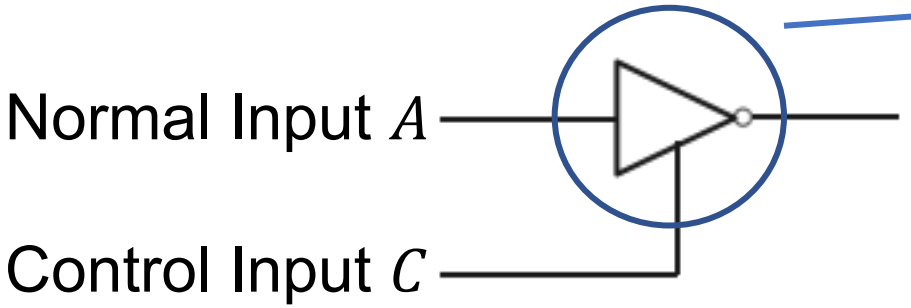
W_7	W_6	W_5	W_4	W_3	W_2	W_1	W_0	Y_2	Y_1	Y_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

8 possible

8



THREE STATE GATE or TRISTATED GATE:



If $C = 1 \rightarrow Y = \bar{A}$

If $C = 0 \rightarrow$ Output in High Impedance state.

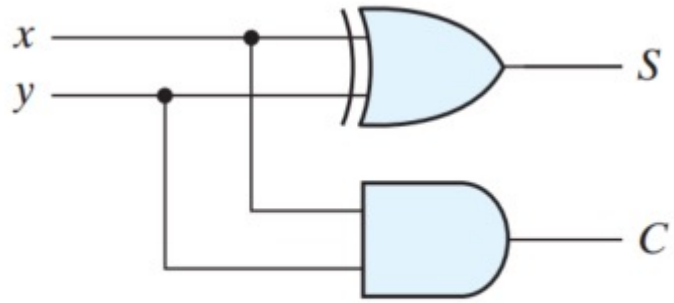
- The third state is a *high-impedance* state in which (1) the logic behaves as an open circuit, which means that the output appears to be disconnected, (2) the circuit has no logic significance, and (3) the circuit connected to the output of the three-state gate is not affected by the inputs to the gate.

- The high-impedance state of a three-state gate provides a special feature not available in other gates. Because of this feature, many three-state gate outputs can be connected using wires to form a common line without endangering loading effects.

Adders (Recall)

Half Adder

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

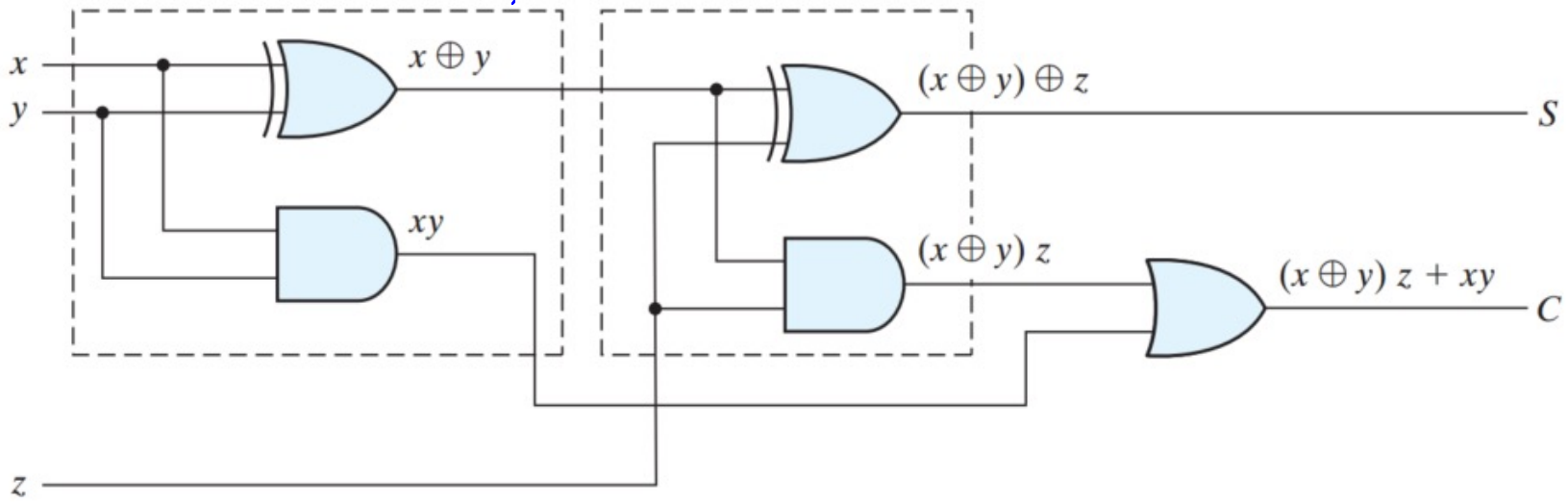


$S = x \oplus y$
$C = xy$

Full Adder

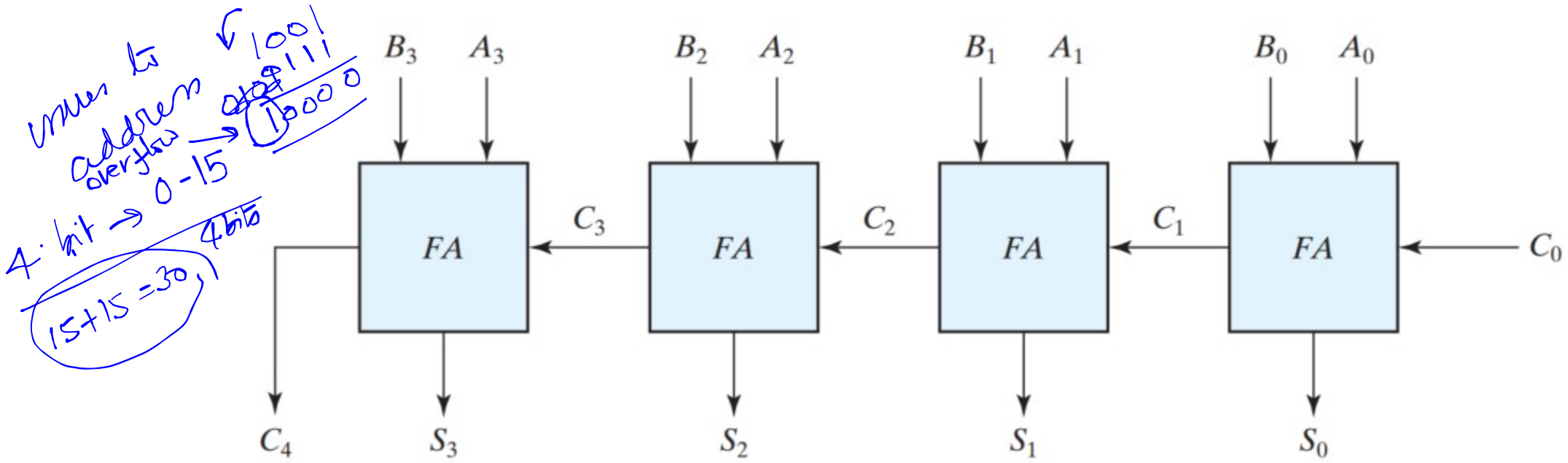
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

2 ip XOR



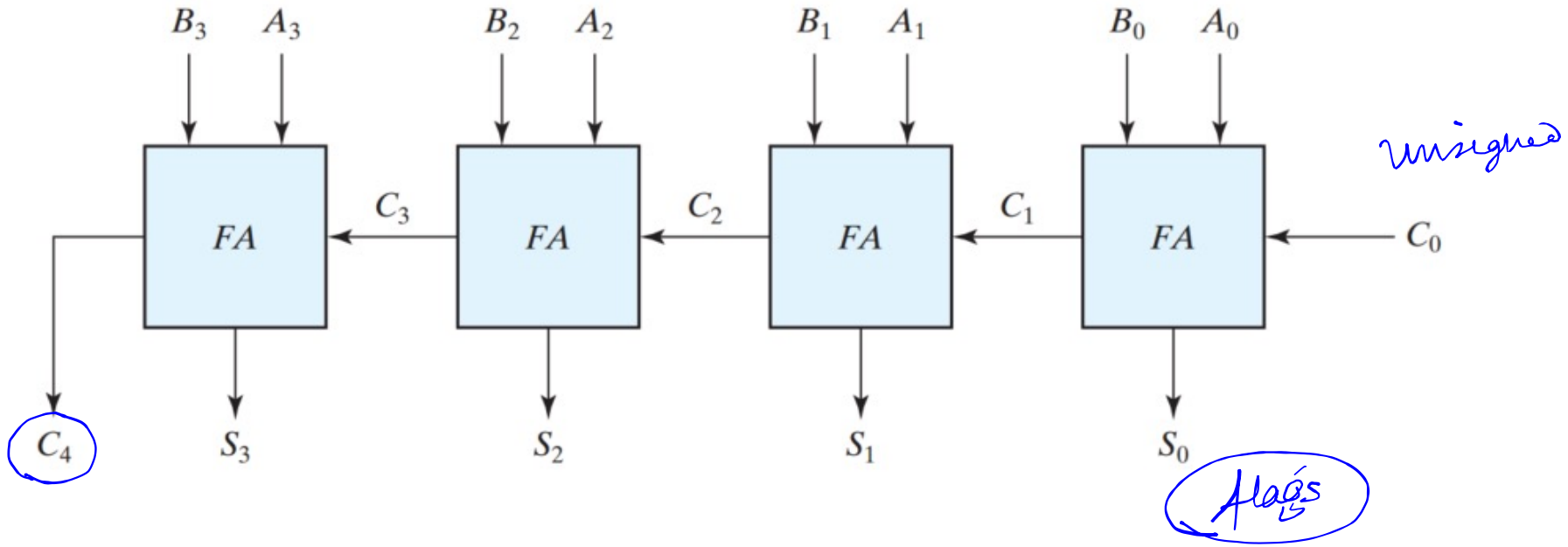
$S = z \oplus (x \oplus y)$ $C = xy + xz + yz$ $C = z(xy' + x'y) + xy$

Adders (Issues):



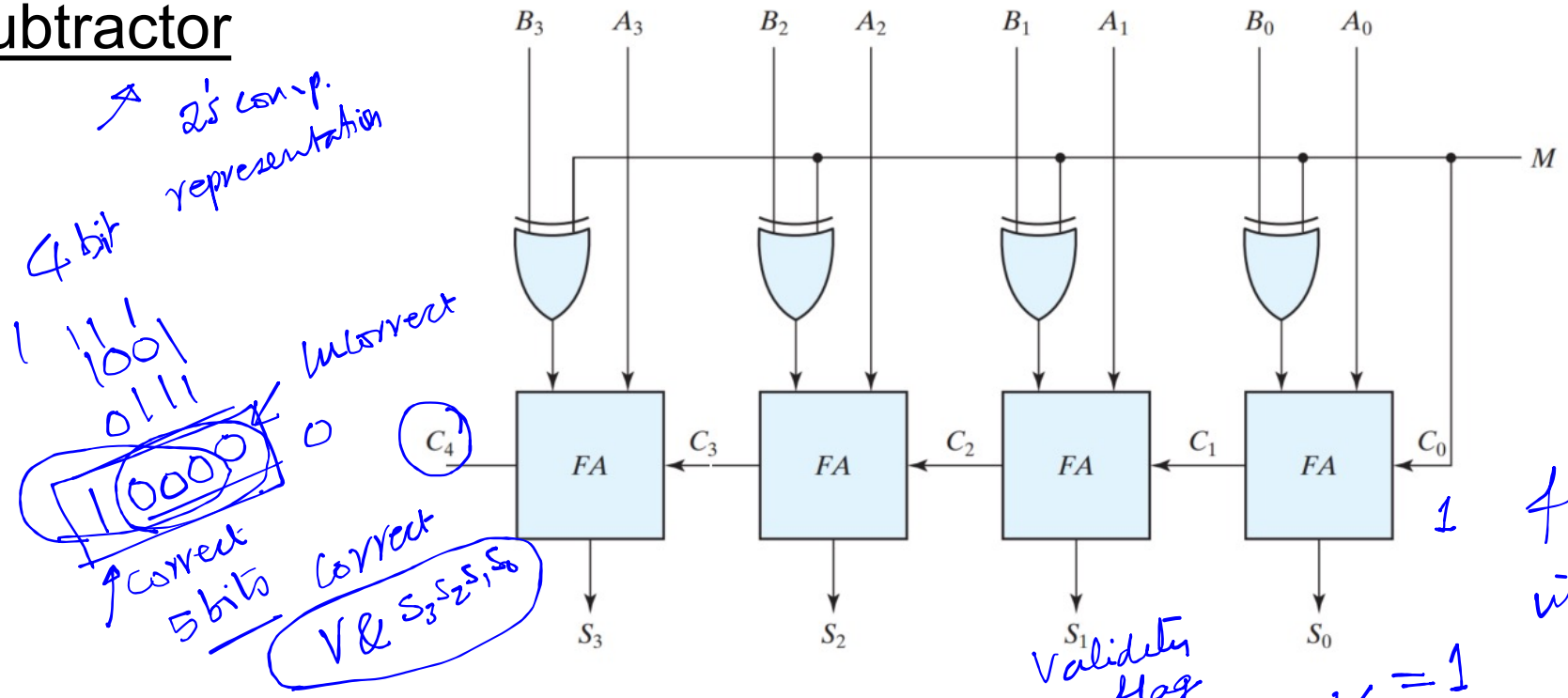
- When two numbers with *n* bits each are added and the sum is a number occupying *n* + 1 bits, we say that an overflow has occurred.
- Overflow is a problem in digital computers because the number of bits that hold the number is finite and a result that contains *n* + 1 bits cannot be accommodated by an *n* -bit word.

Adders (Issues):



- For adder with unsigned numbers as inputs, introduce an output flag, F, which goes high when overflow occurs
- When two unsigned numbers are added, say sum of two 4-bit numbers, an overflow is detected from the end carry out of the most significant position i.e., $F = C_4$.

Adder/Subtractor

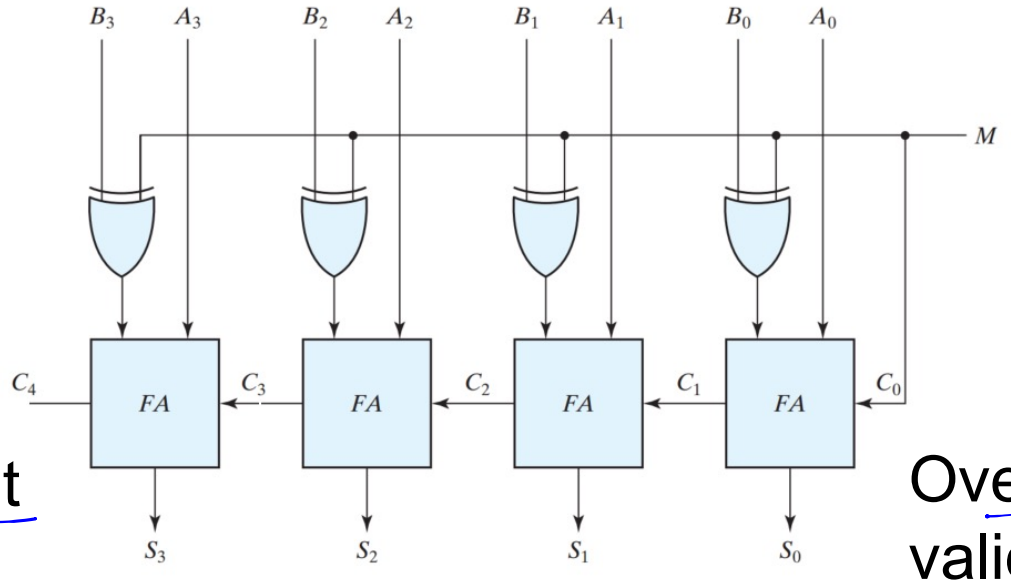


- For adder/subtractor with signed number inputs, we would need to introduce three independent output flags 1) Output flag, V, goes high when the output is invalid, 2) Output flag, F_1, goes high when the sum is negative, and 3) Output flag, F_2, goes high when the sum is zero.

$F_2 = \bar{S}_3 \cdot \bar{S}_2 \cdot \bar{S}_1 \cdot \bar{S}_0$

Overflow:

4-bit Signed 2's Complement addition



	1 1 0	
(+7)	0 1 1 1	
+ (−2)	+ 1 1 1 0	
(+5)	1 0 1 0 1	

$C_3 = 1$
$C_4 = 1$

Overflow exists, the result is valid and positive

	1 1 1	
(−7)	1 0 0 1	
+ (−1)	+ 1 1 1 1	
(−8)	1 1 0 0 0	

$F_1 = S_3$

$C_3 = 1$
$C_4 = 1$

Overflow exists, the result is valid and negative

	0 0 0	
(−7)	1 0 0 1	
+ (+2)	+ 0 0 1 0	
(−5)	1 0 1 1	

$F_1 = S_3$

$C_3 = 0$
$C_4 = 0$

No Overflow exists, the result is valid and negative

	0 0 0	
(−7)	1 0 0 1	
+ (−2)	+ 1 1 1 0	
(−9)	1 0 1 1 1	

$C_3 = 0$
$C_4 = 1$

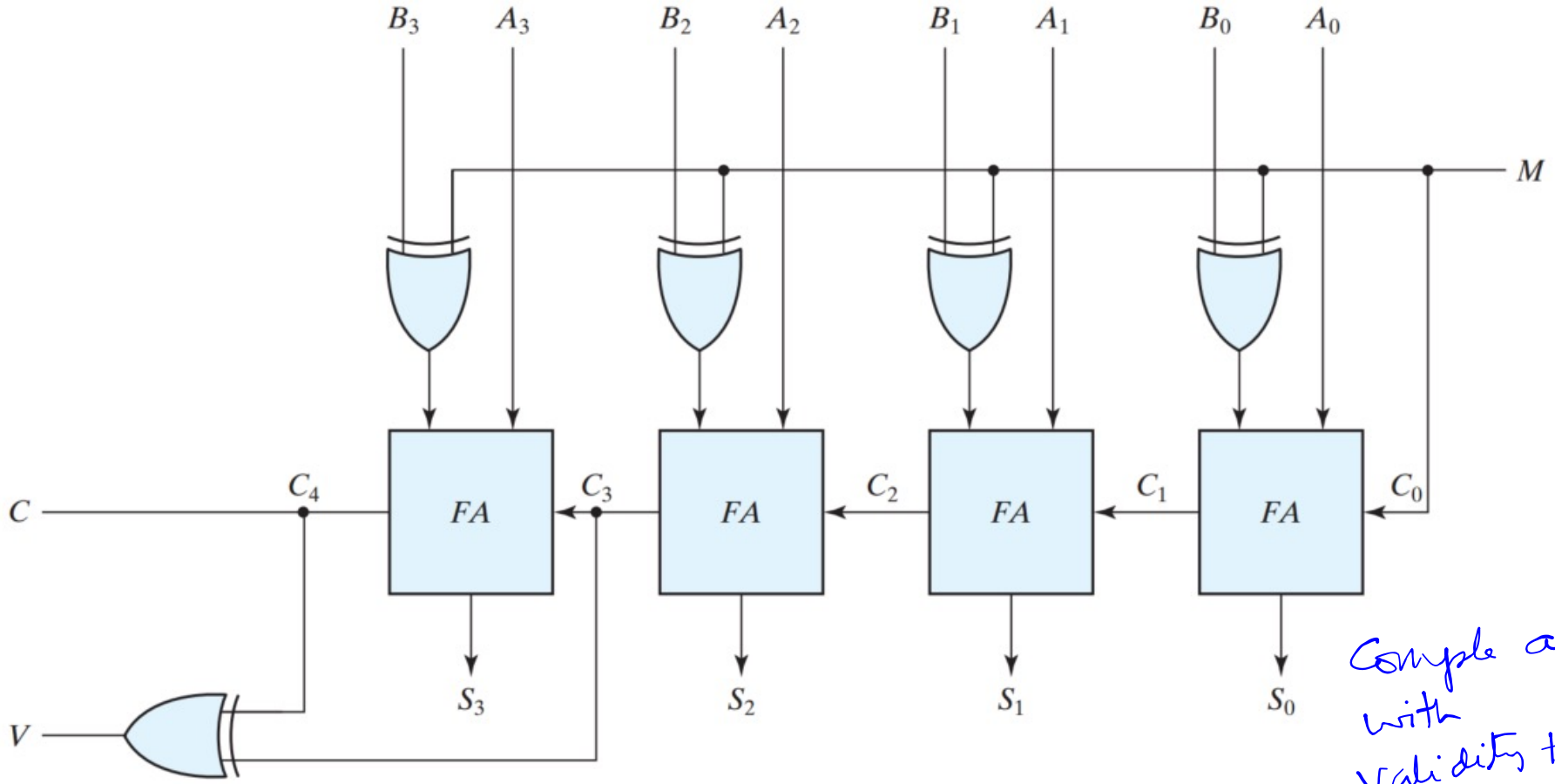
	1 1 0	
(+7)	0 1 1 1	
+ (+2)	+ 0 0 1 0	
(+9)	1 0 0 1	

$V = C_4 \oplus C_3$

$C_3 = 1$
$C_4 = 0$

The results are invalid (C_3 and C_4 are exclusive)

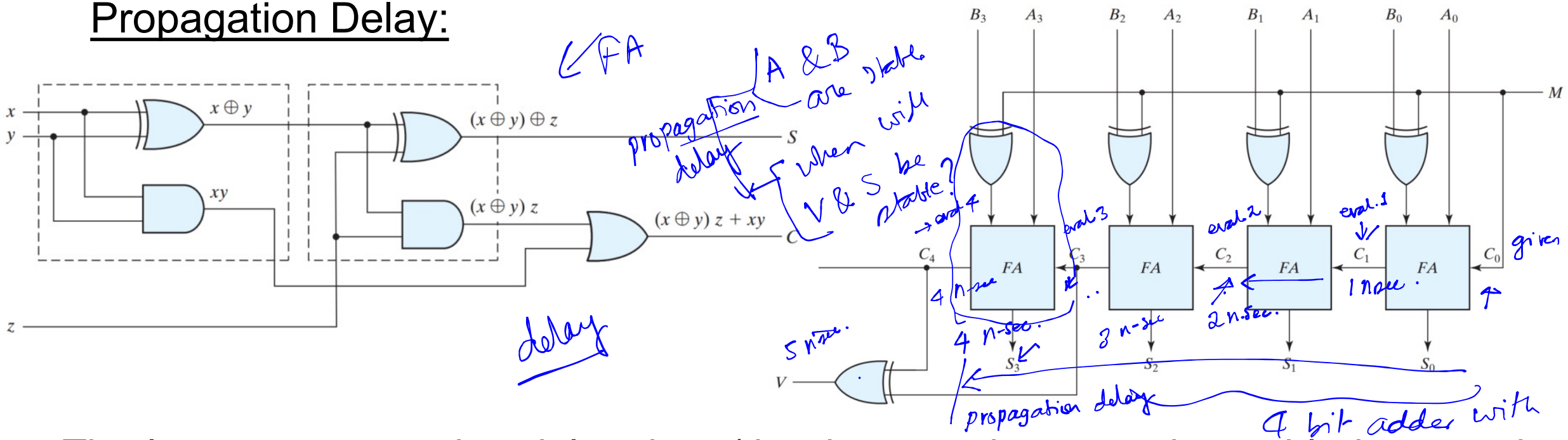
Overflow



Compile adder with validity + sign + zero check.

$$V = C_4 \oplus C_3;\ F_1 = S_3\ ;\ \ F_2 = \overline{S_0}\,\overline{S_1}\,\overline{S_2 S_3}$$

Propagation Delay:



- The longest propagation delay time (the time gap between the stable input and stable output) in an adder is the time it takes the carry to propagate through the full adders.
- Consider inputs A_3 and B_3 that are available as soon as input signals are applied to the adder. However, carry C_3 to this stage does not settle to its final value until C_2 is available from the previous stage. Similarly, C_2 has to wait for C_1 and so on down to C_0.

Carry Lookahead Logic:

P – Propagate and G – Generate.

$$B_i = B_i^* \oplus M$$

$$P_i = A_i \oplus B_i \; ; \; G_i = A_i B_i$$

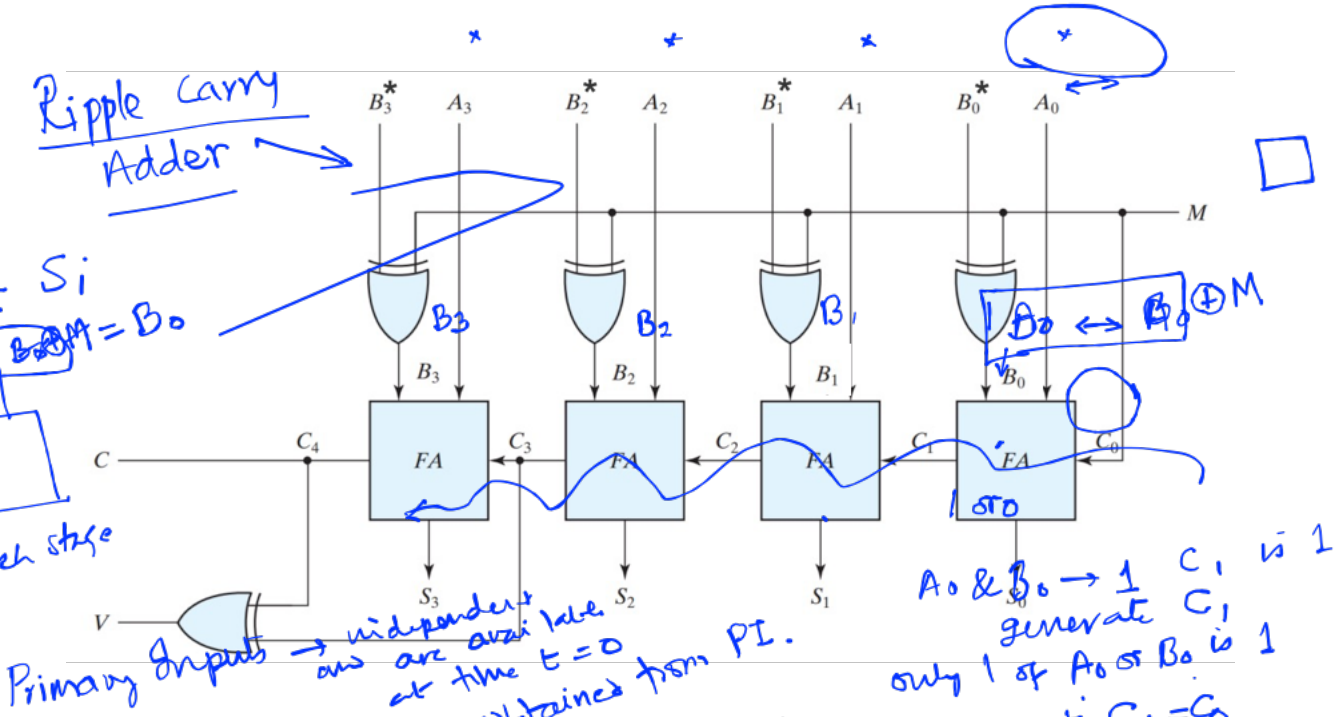
$$S_i = P_i \oplus C_i \; ; \; C_{i+1} = G_i + P_i C_i$$

C_0 = Input carry ; $C_1 = G_0 + P_0 C_0$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0)$$

$$= G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$



Carry Lookahead Logic:

P – Propagate and G – Generate.

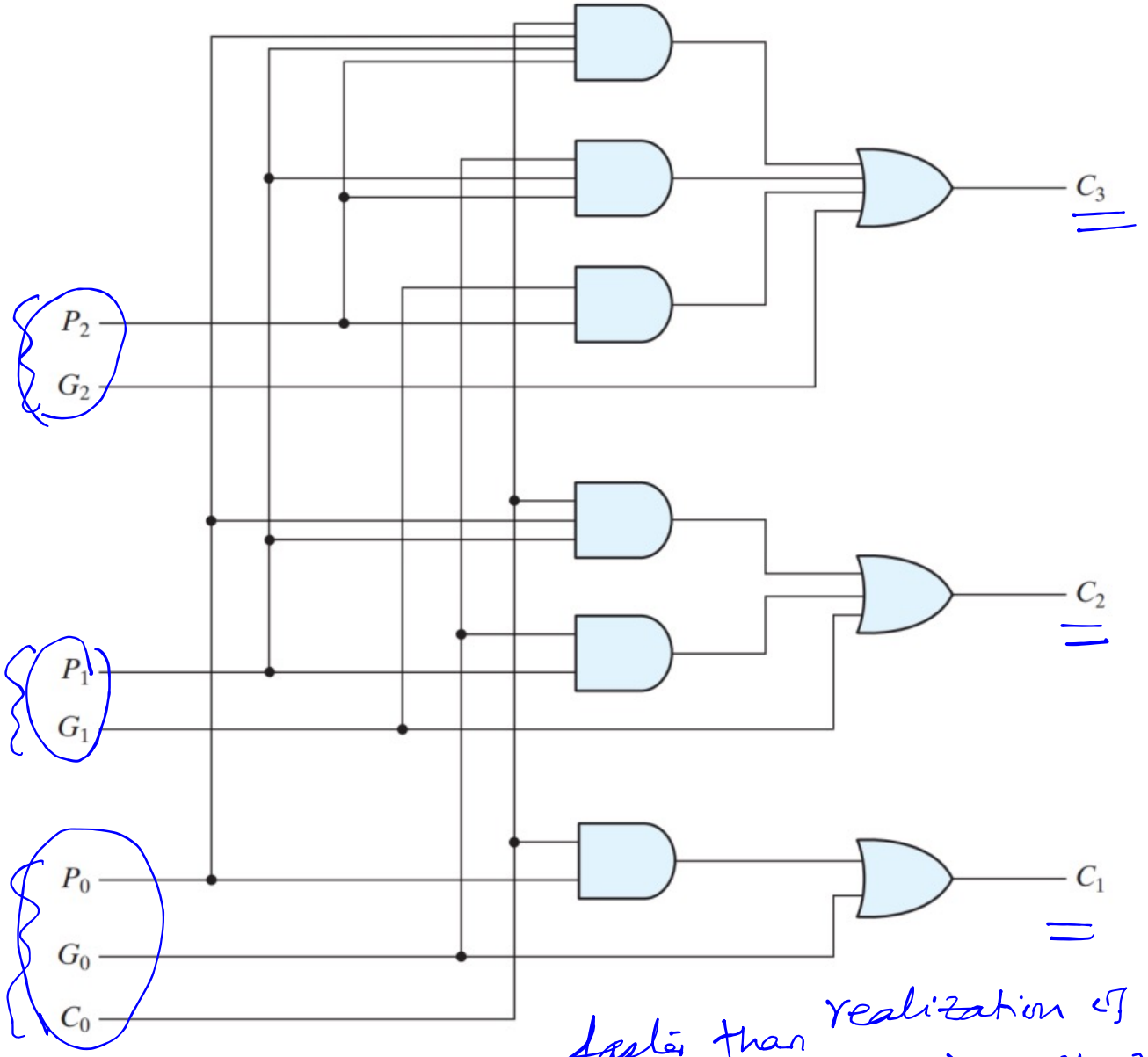
$$P_i = A_i \oplus B_i \; ; \; G_i = A_i B_i$$

$$S_i = P_i \oplus C_i \; ; \; C_{i+1} = G_i + P_i C_i$$

C_0 = Input carry ; $C_1 = G_0 + P_0 C_0$

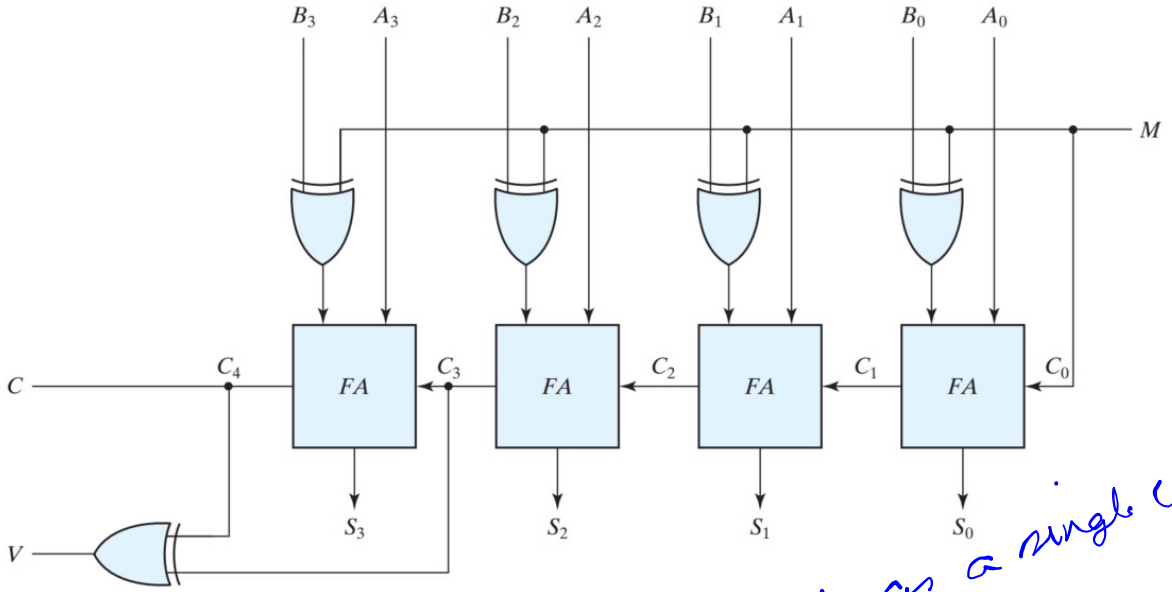
$$C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0)$$
$$= G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

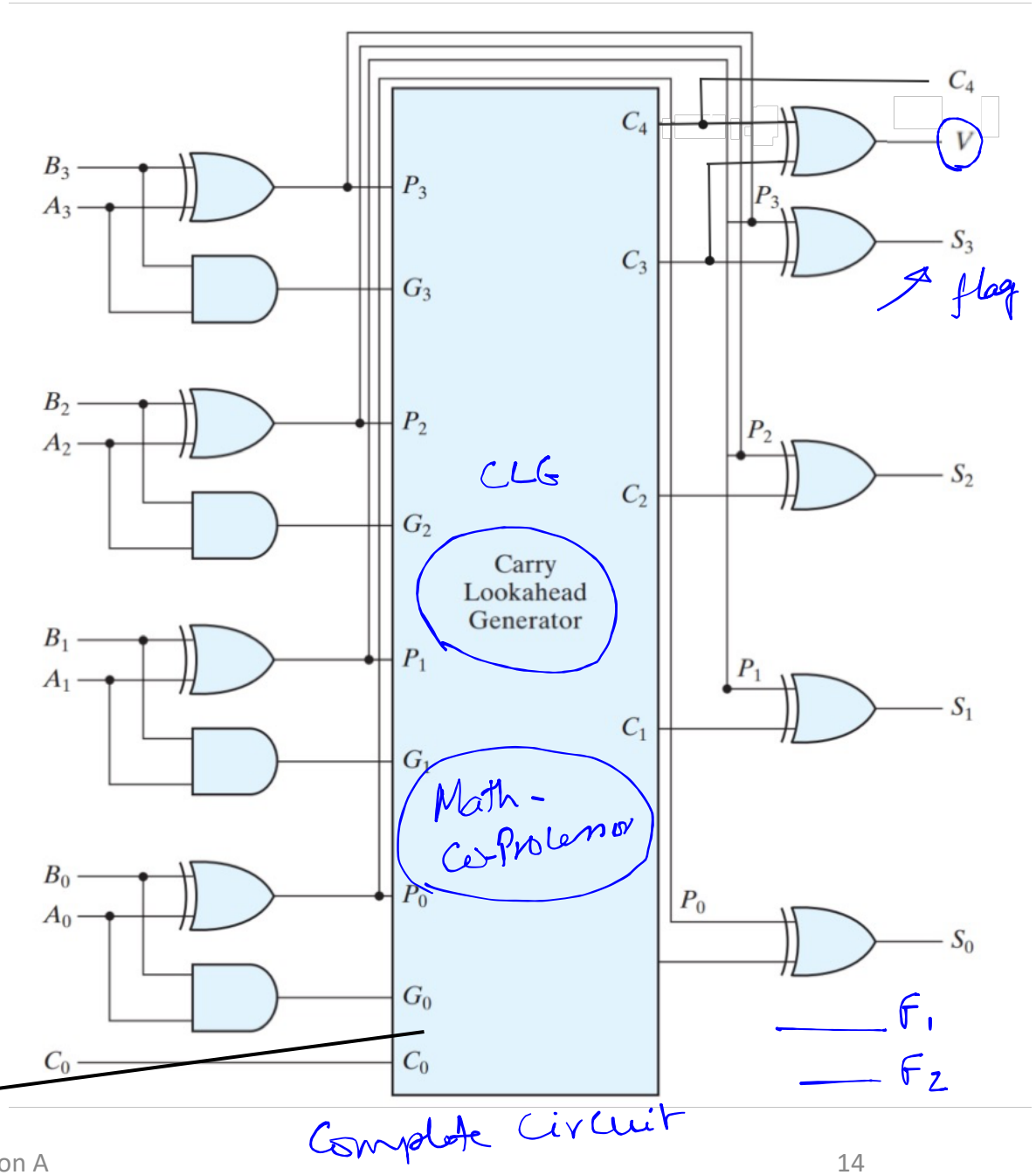


Faster than realization of Carry Look Ahead Logic

The Complete Adder:

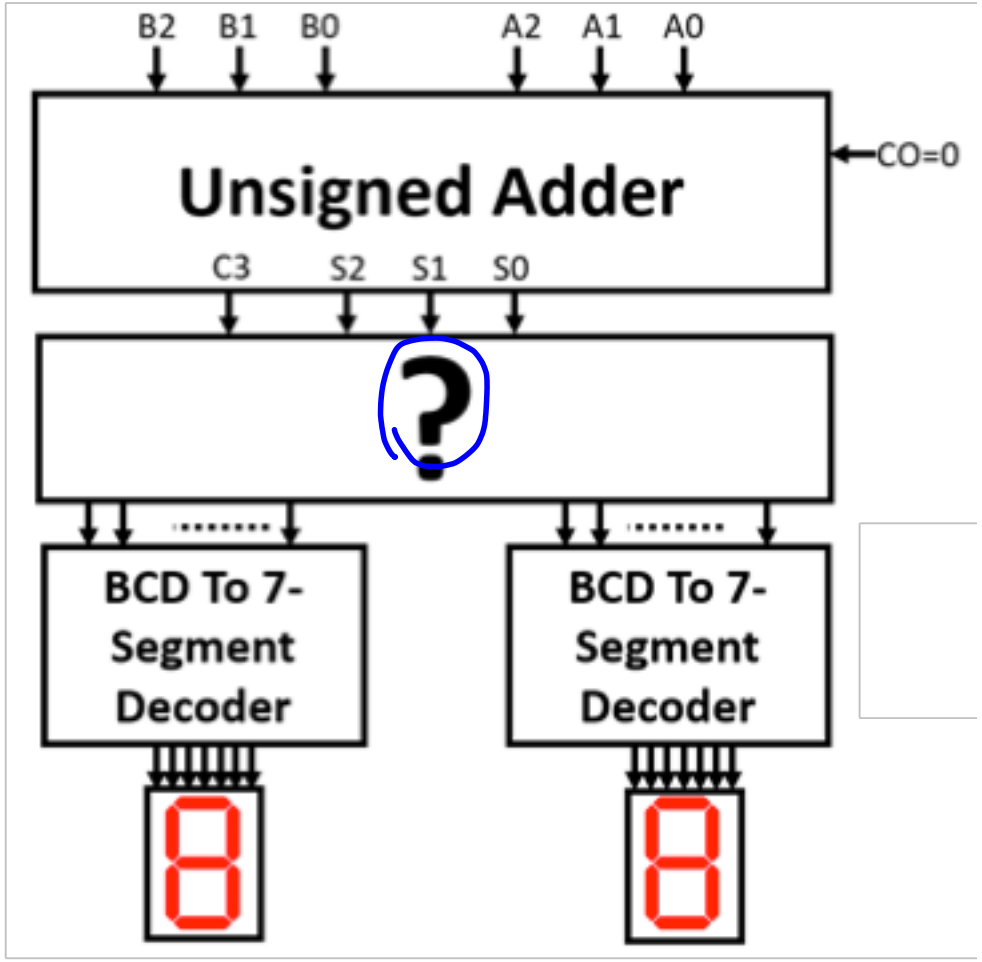


↗ available as a single chip + CLG chip Math Co-Processor.

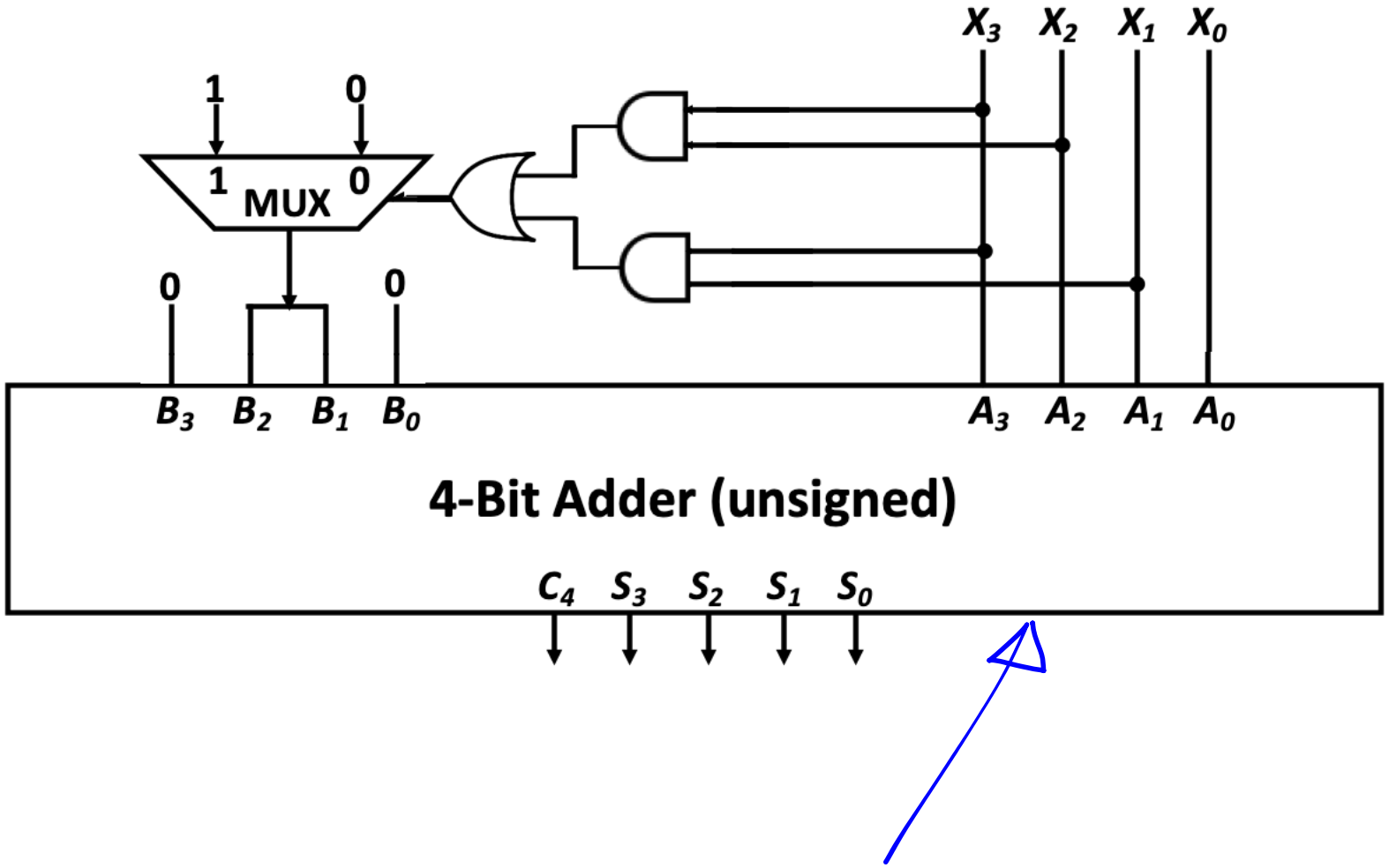


Math Co-Processor

HW --- Identify the Functional Blocke marked with ?:



H.W: Identify the functionality:



A_3	A_2	A_1	A_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

B_3	B_2	B_1	B_0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	1	1	0
0	1	1	0
0	1	1	0
0	1	1	0
0	1	1	0

We will discuss the HW problems in the class after you guys try it and raise questions in trying to understand the blocks or the function of these circuits.