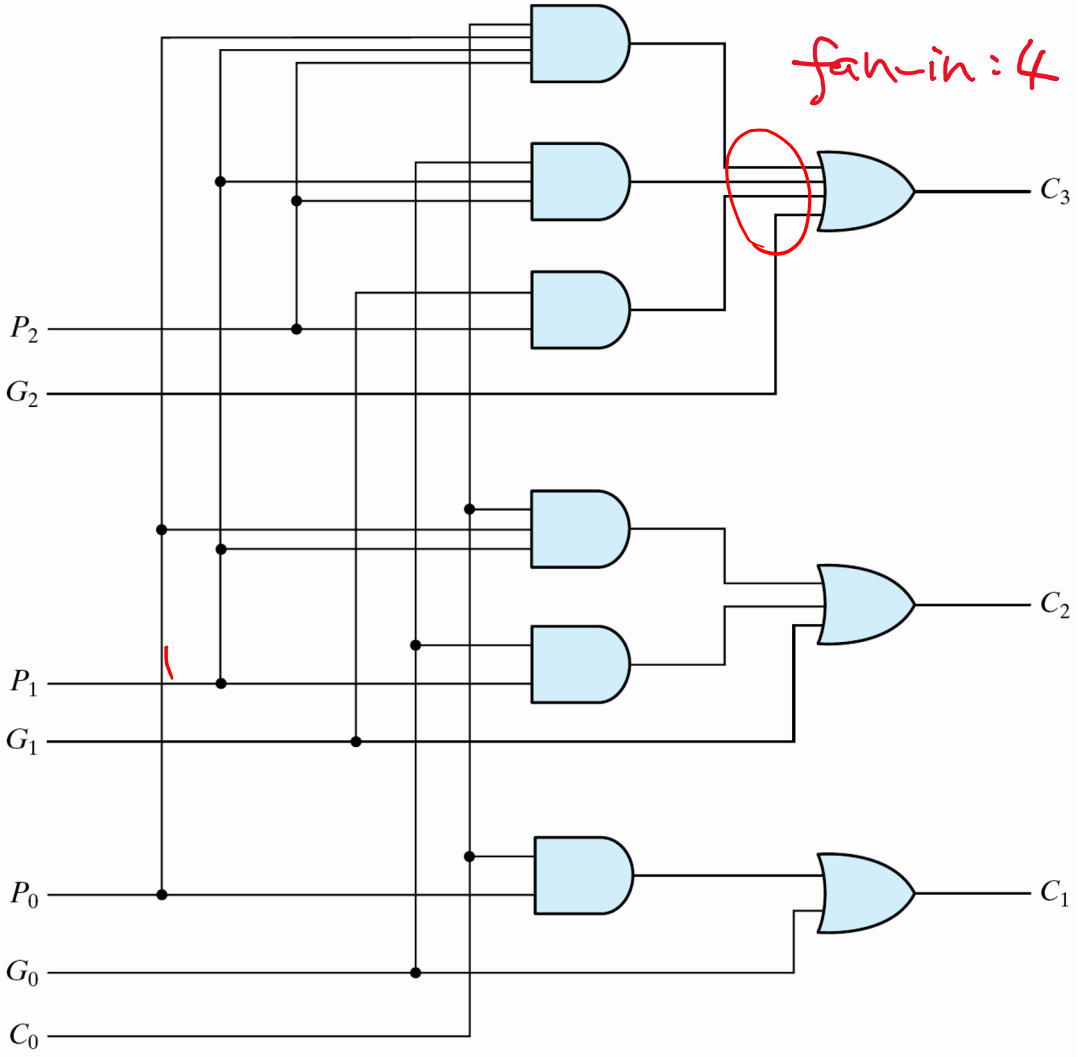
Carry Lookahead Adder (CLA) (2/5)

- Logic diagram of carry lookahead generator

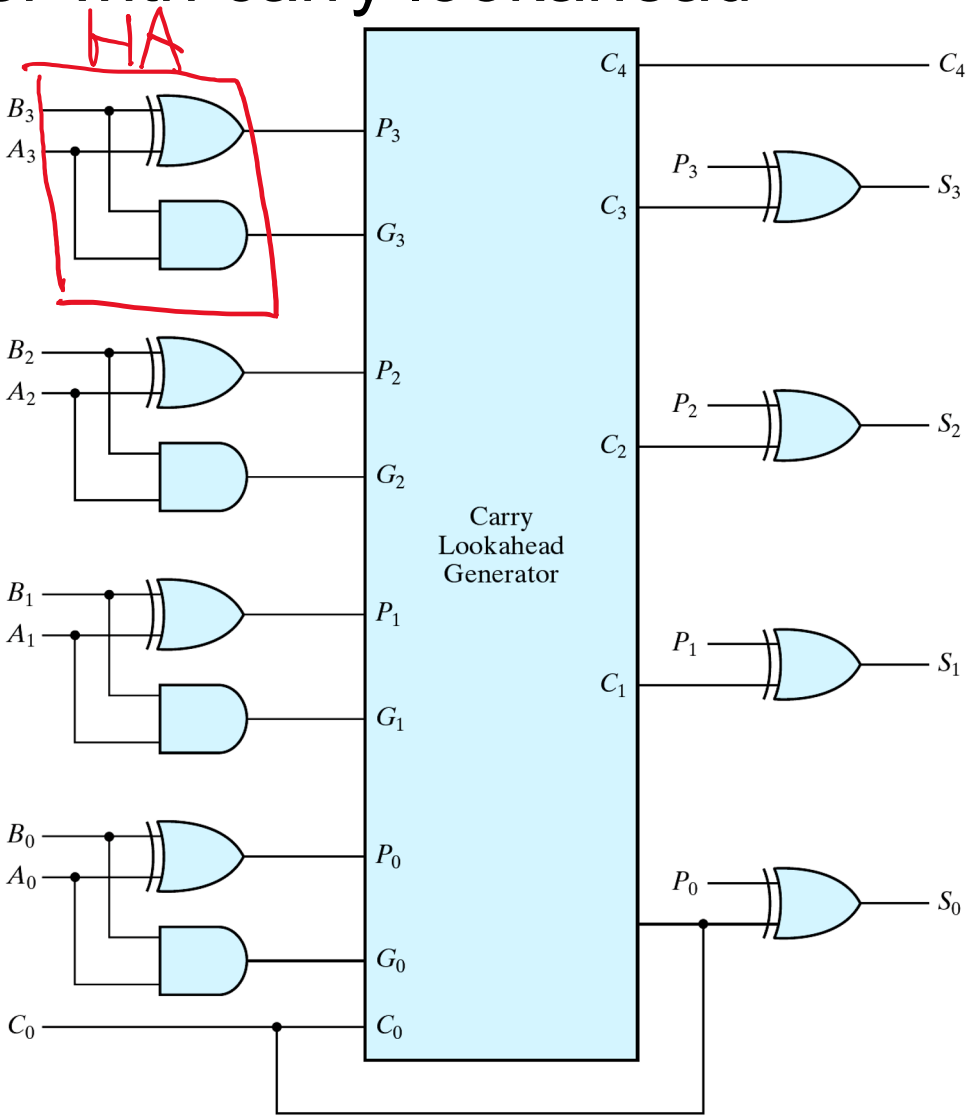


Carry Lookahead Adder (CLA) (3/5)

- No carry (propagation) chain
- 2-level logic achieved
- However, fan-in number of AND/OR gates still grows gradually
  - The overall delay is not constant
  - Fan-in number is usually limited to 4 bits

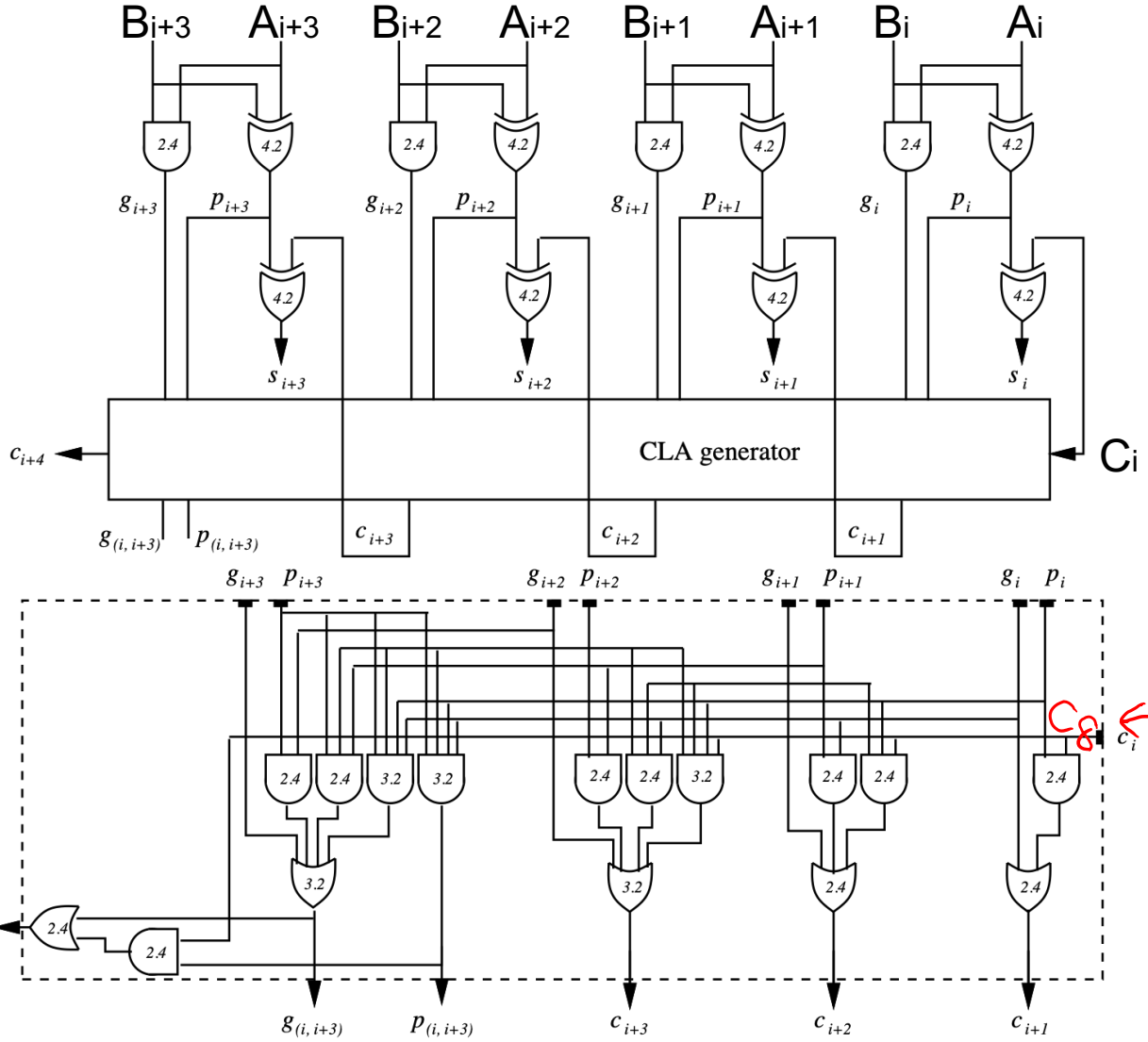
Carry Lookahead Adder (CLA) (4/5)

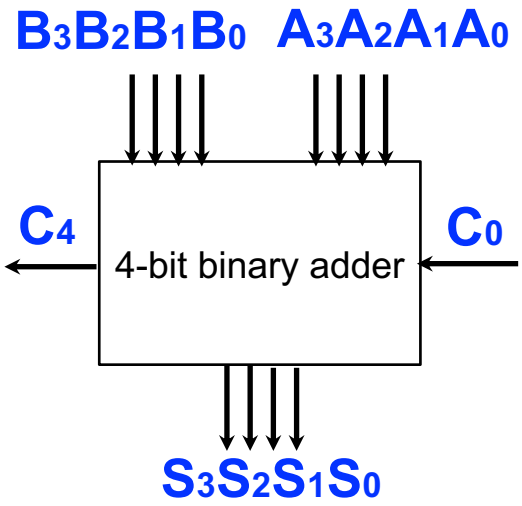
- 4-bit adder with carry lookahead

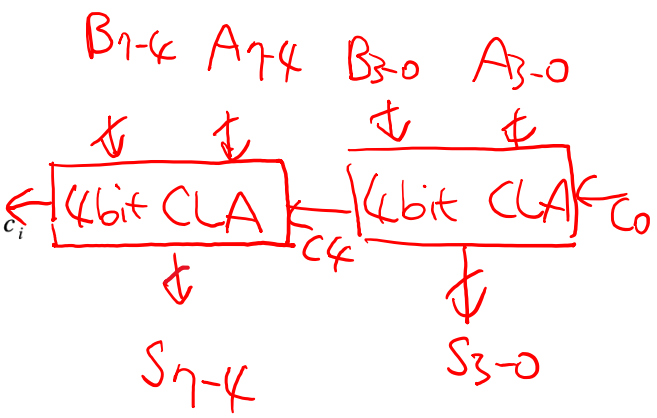


Carry Lookahead Adder (CLA) (5/5)

- Detailed logic diagram of 4-bit CLA

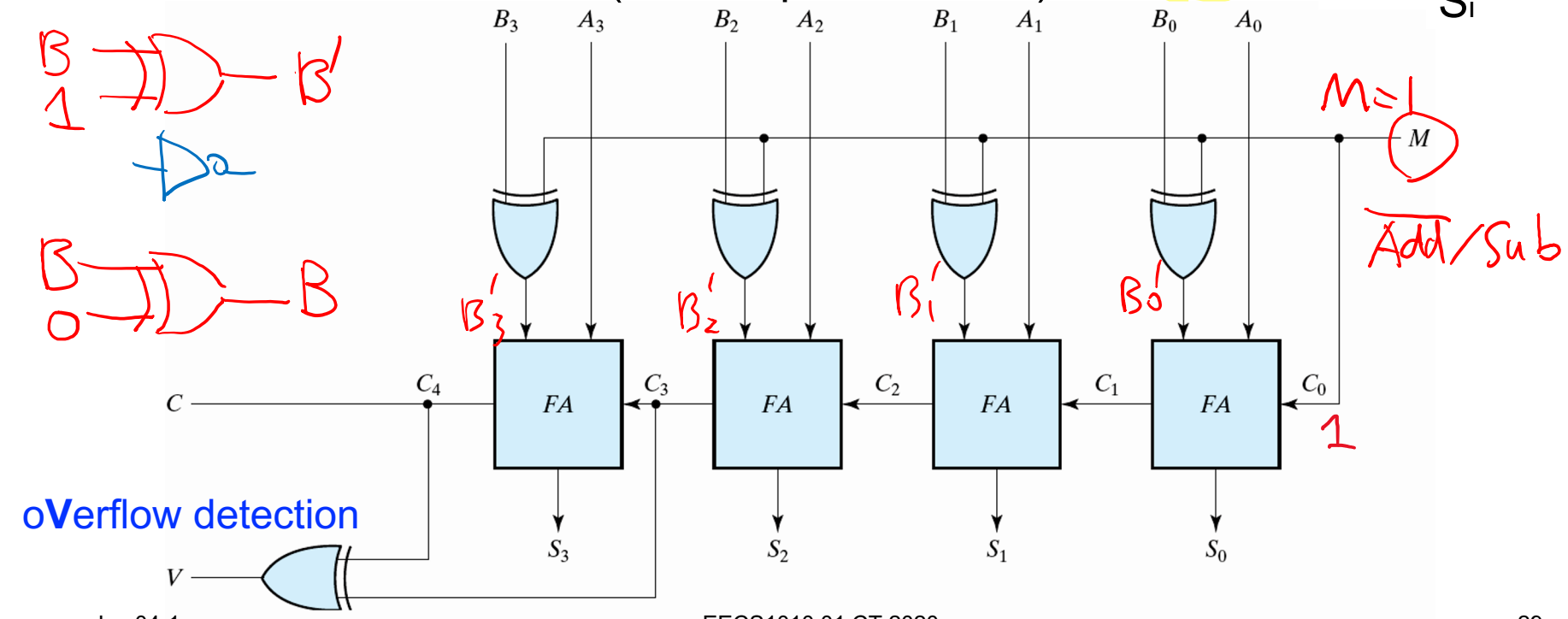


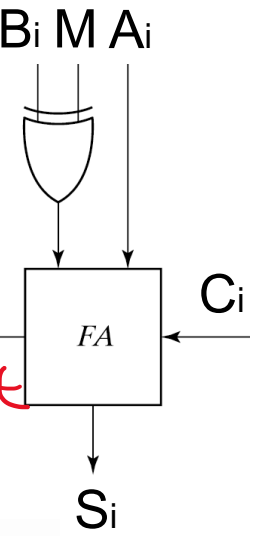




Binary Adder/Subtractor

- Binary subtraction normally is performed by adding the minuend to the 2's complement of the subtrahend
- 4-bit adder-subtractor
  - M = 0: A + B
  - M = 1: A – B = A + (2's complement of B) = A + B' + 1





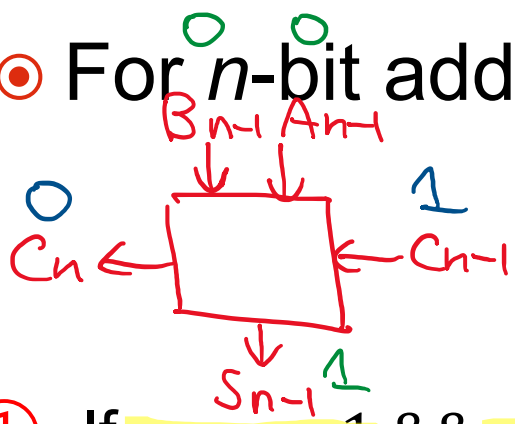
oVerflow detection

Overflow (1/2)

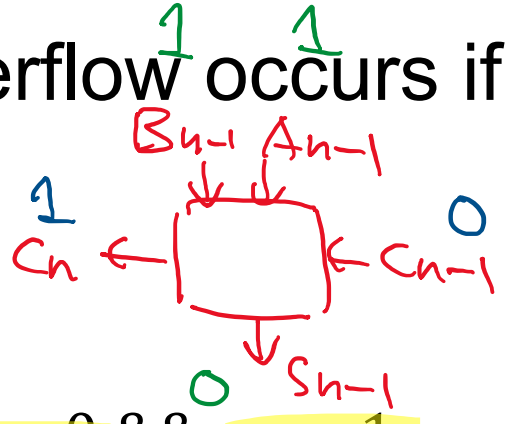
- Adding two positive numbers and obtaining a negative number; or
- Adding two negative numbers and obtaining a positive number
- Overflow can be detected
  - If the carry into the sign bit position and the carry out of the sign bit position are not equal
  - V = 1 when the overflow occurs

Overflow (2/2)

- For *n*-bit adder/subtractor, overflow occurs if



$$V = c_{n-1} \oplus c_n$$



① If $c_{n-1} = 1$ && $c_n = 0$

$\Rightarrow A_{n-1} = B_{n-1} = 0$

$\Rightarrow S_{n-1} = 1$

$\Rightarrow$ Adding two positive numbers, result is negative

② If $c_{n-1} = 0$ && $c_n = 1$

$\Rightarrow A_{n-1} = B_{n-1} = 1$

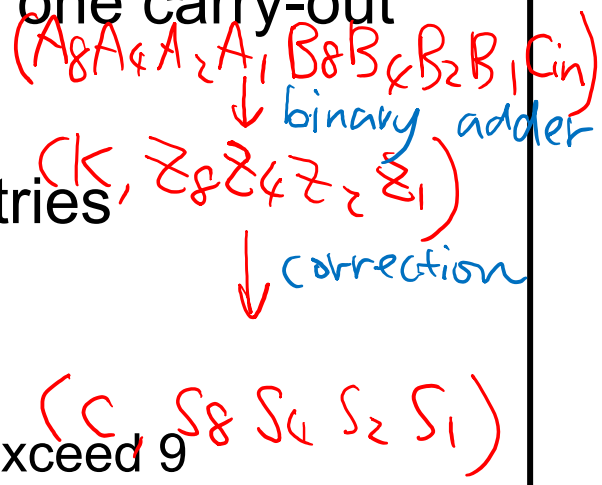
$\Rightarrow S_{n-1} = 0$

$\Rightarrow$ Adding two negative numbers, result is positive

Decimal Adder

Decimal Adder (1/3)

- Addition of 2 decimal digits in BCD
  - 9 inputs: 2 BCD's and one carry-in
  - 5 outputs: 1 BCD and one carry-out
- Design approaches
  - Truth table with 2^9 entries (seriously?!)
  - Using binary adders
    - A digit in BCD cannot exceed 9
      - The max sum 9 + 9 + 1 = 19
    - Binary to BCD (adding 6 for correction)



Decimal symbol	BCD digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Decimal Adder (2/3) $C = Z_8Z_2 + Z_8Z_4 + K$

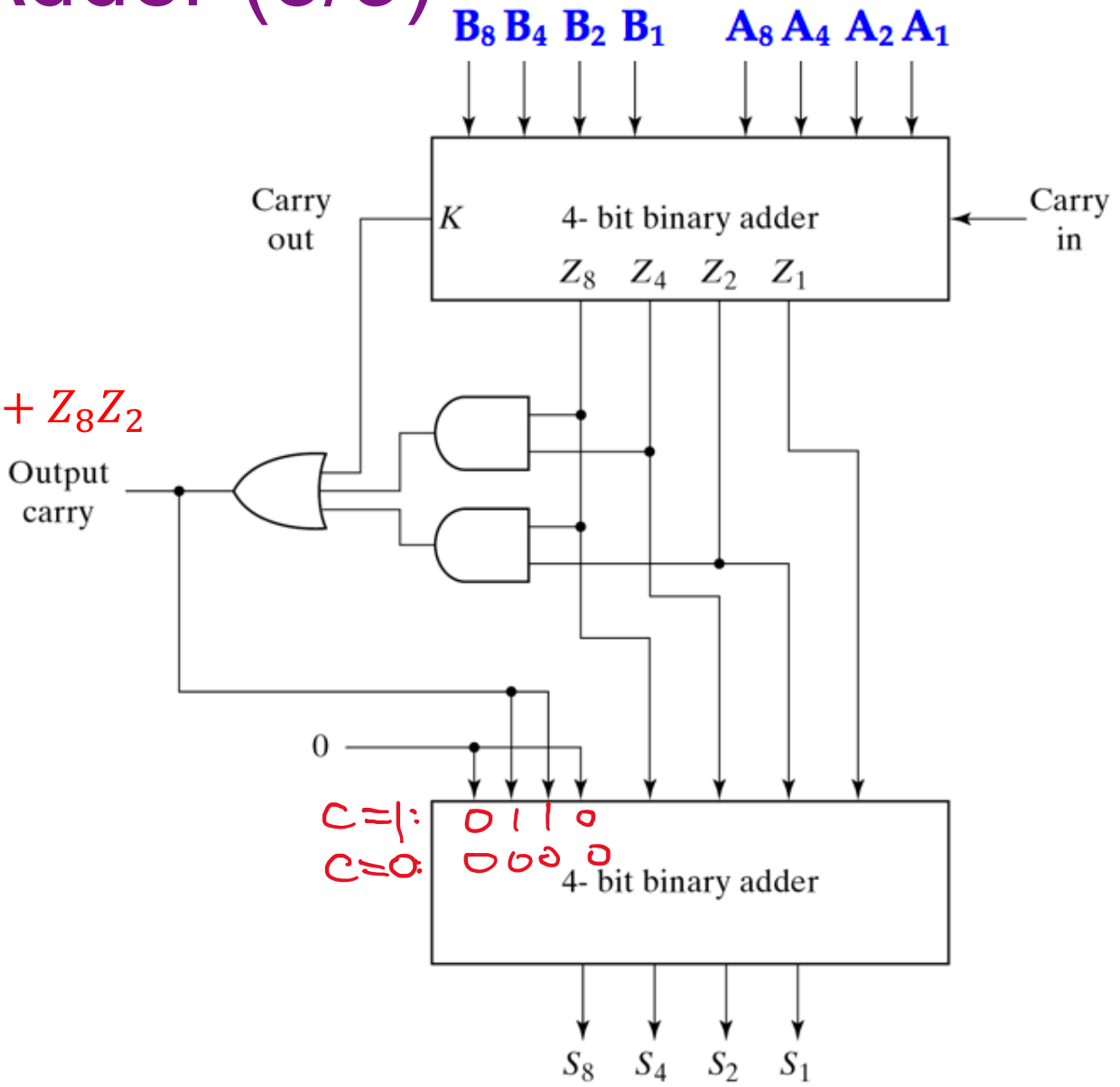
Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

0-9

10 - 19

Decimal Adder (3/3)

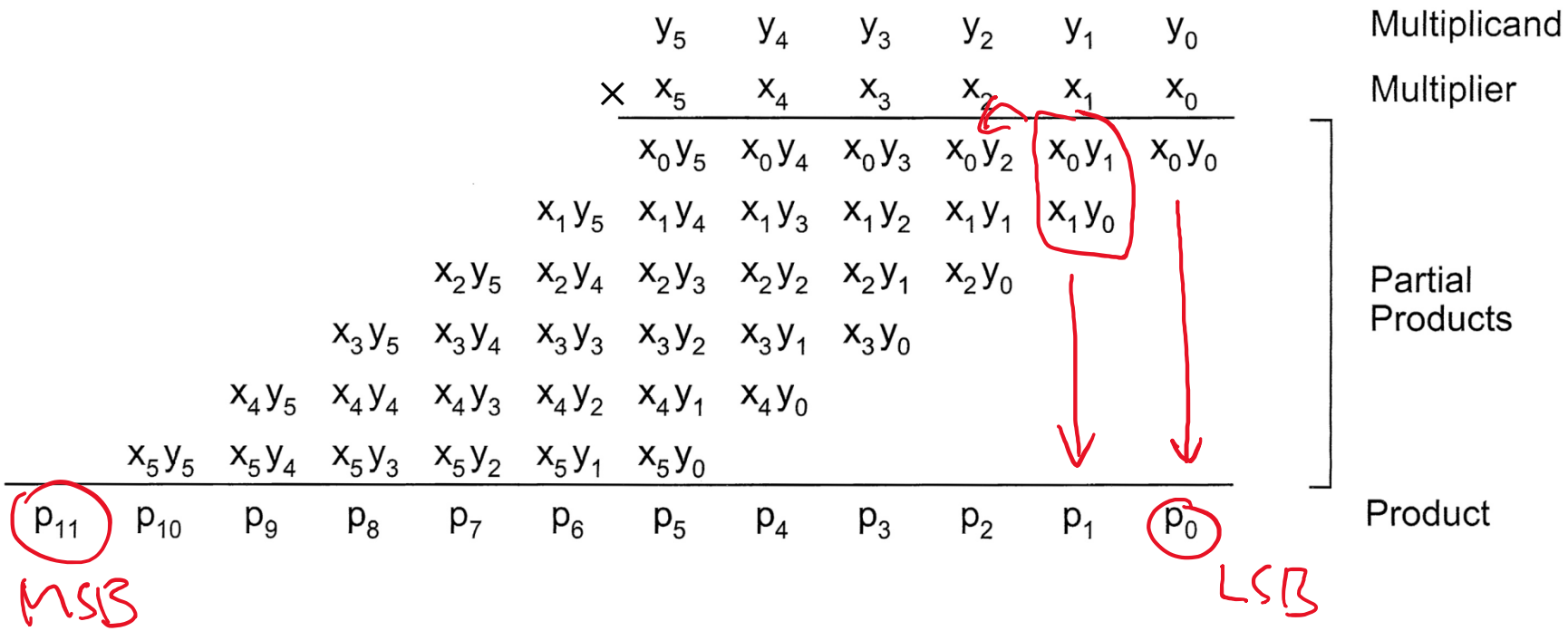
$$C = K + Z_8Z_4 + Z_8Z_2$$



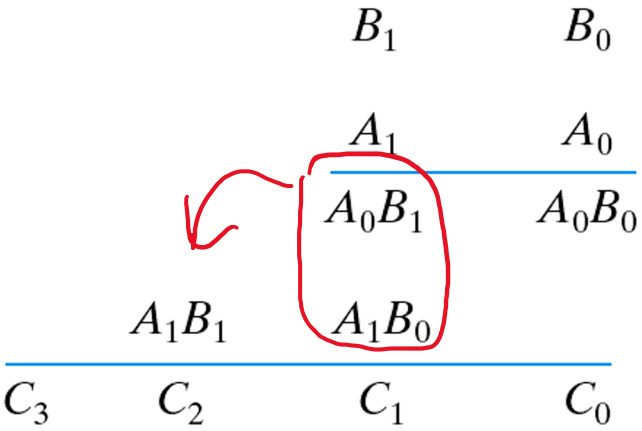
Binary Multiplier

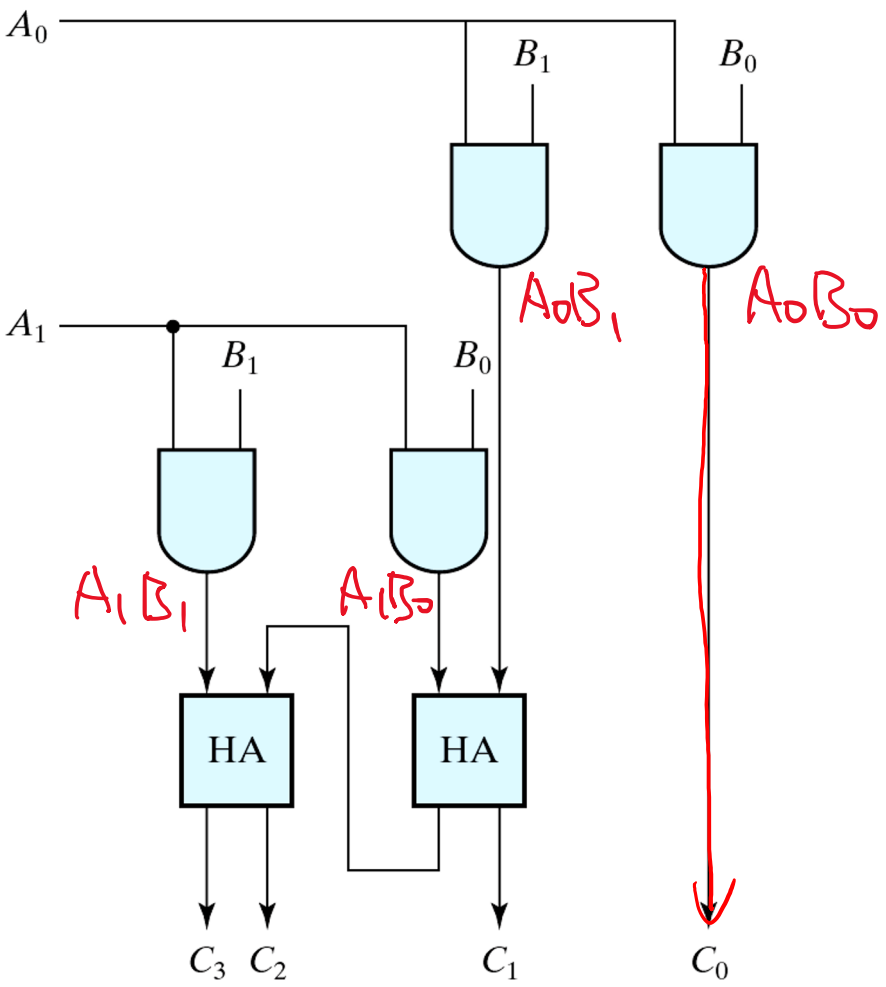
Multiplication

- Multiplication consists of
  - Generation of partial products
  - Accumulation of shifted partial products

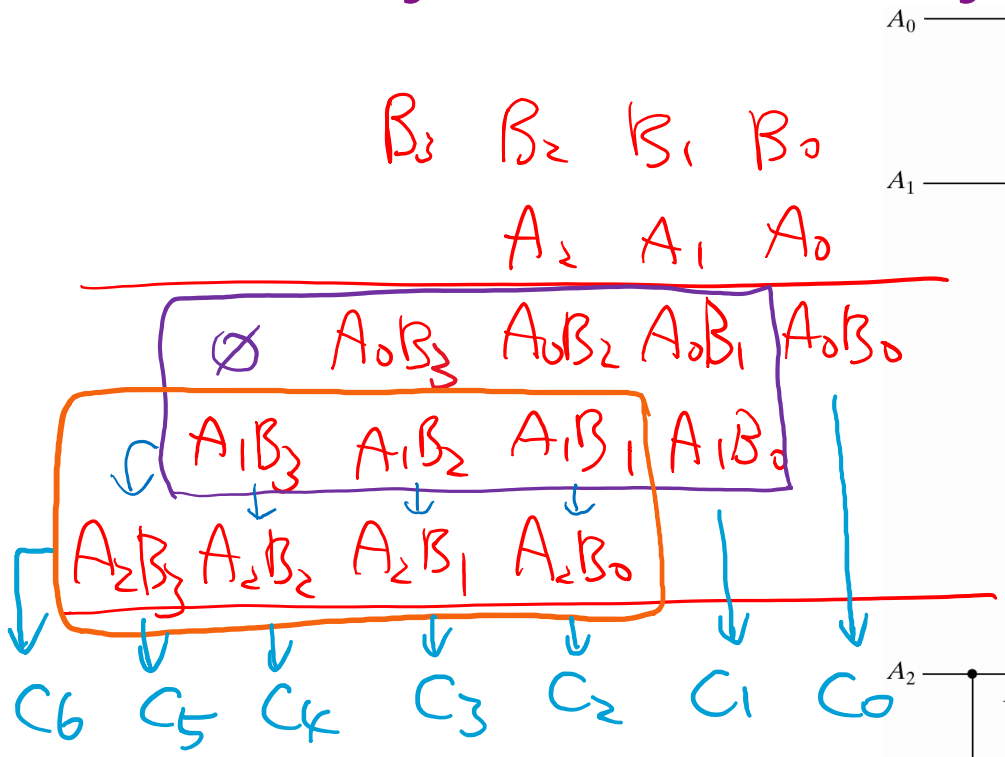


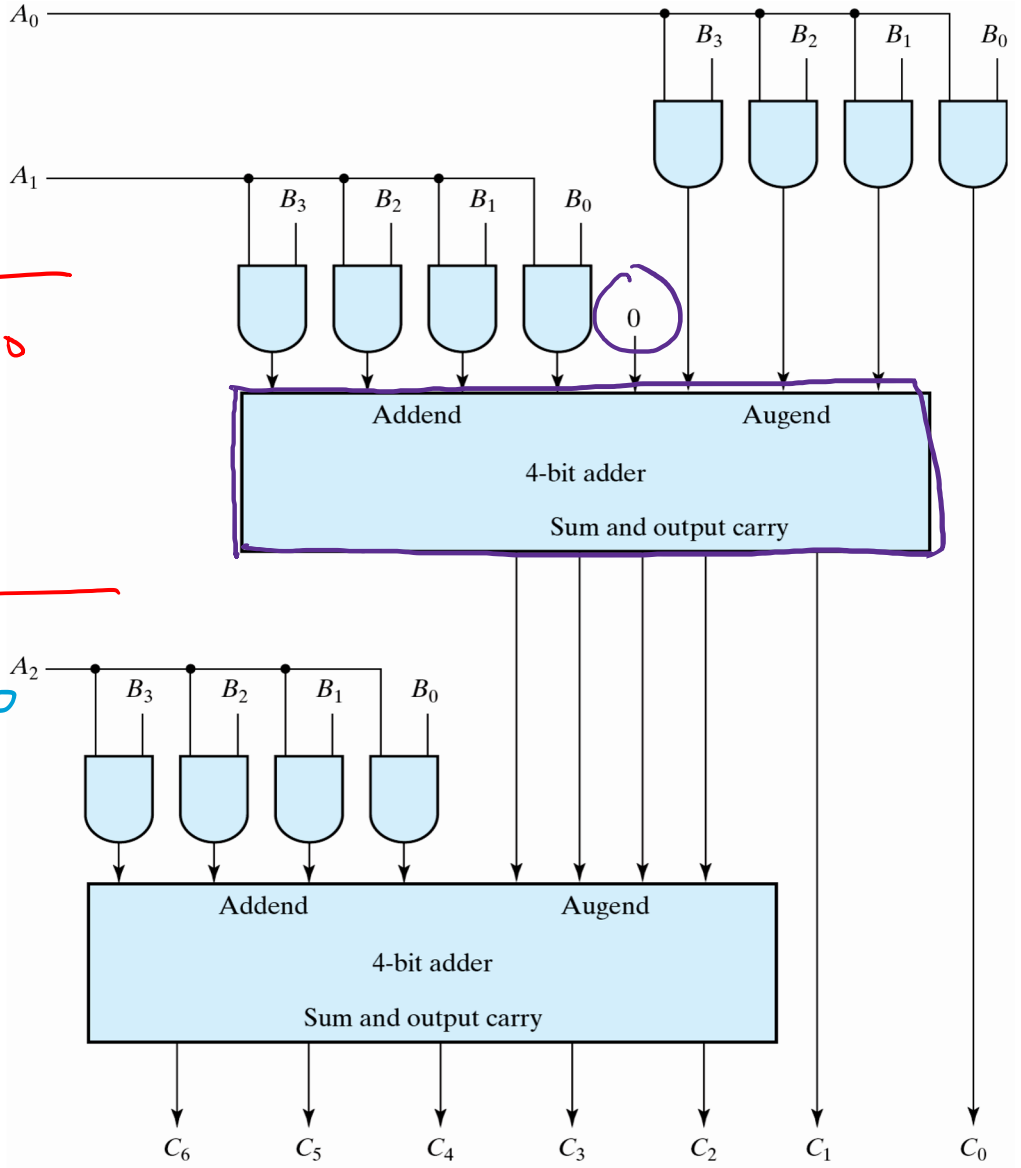
2-bit by 2-bit Binary Multiplier





4-bit by 3-bit Binary Multiplier





Magnitude Comparator

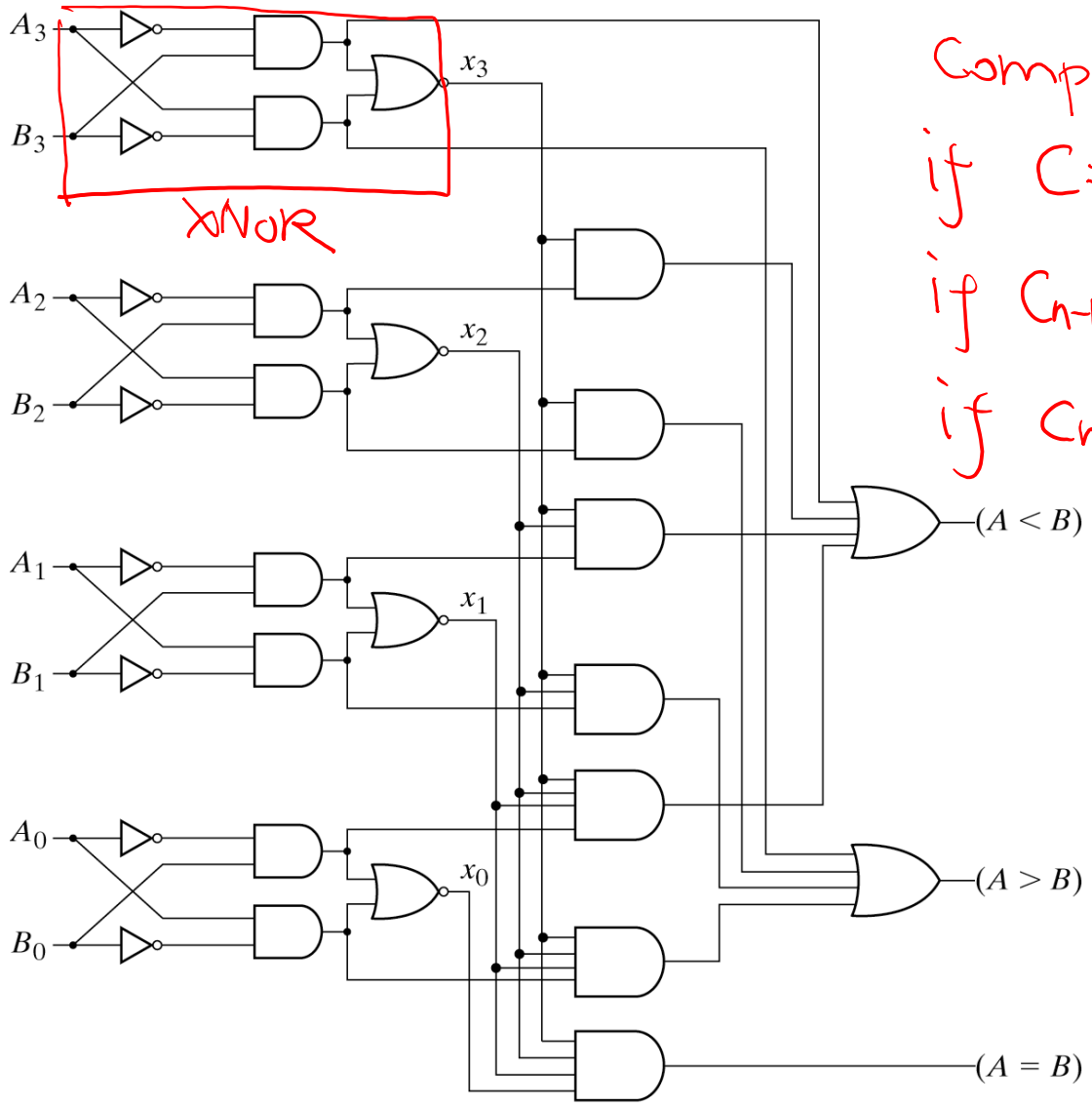
Equality Comparator

Magnitude Comparator (1/2)

- Comparison of two positive numbers, three possible results
  - $A = B$, $A > B$, $A < B$
- Design approaches (for *n*-bit numbers)
  - By the truth table: 2^{2n} rows => not practicable
  - By algorithm to build a regular circuit

    $A = A_3A_2A_1A_0$, $B = B_3B_2B_1B_0$
    - $A = B$, if $A_3 = B_3$ and $A_2 = B_2$ and $A_1 = B_1$ and $A_0 = B_0$
      - Equality: $x_i = A_iB_i + A_i'B_i' = (A \oplus B)'$ (XNOR)
      - $A = B \rightarrow x_3x_2x_1x_0$
    - $A > B$
      - $A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$
    - $A < B$
      - $A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$

Magnitude Comparator (2/2)



2's complement

Compute $B - A = C$

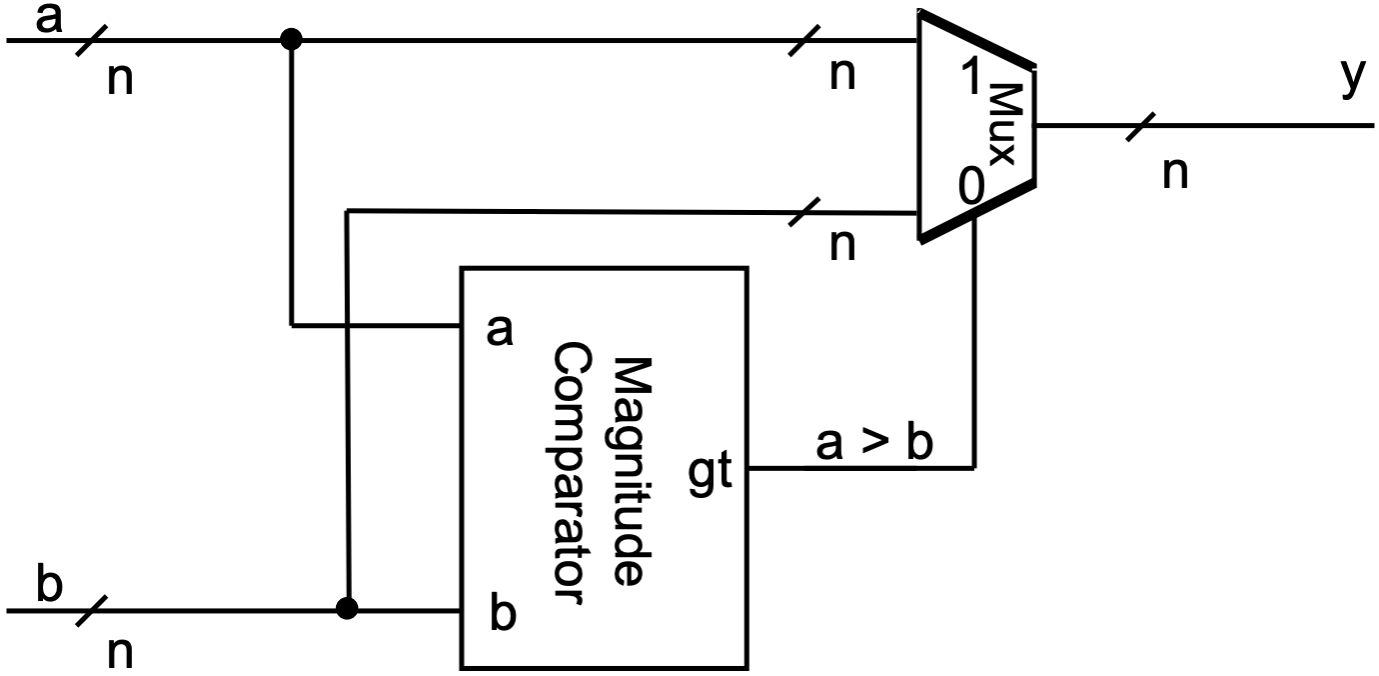
if $C=0 \Rightarrow A=B$

if $C_{n-1}=1 \Rightarrow B<A$

if $C_{n-1}=0 \Rightarrow B \geq A$

Maximum Unit

$$y = \max\{a, b\}$$



Decoders

Decoders

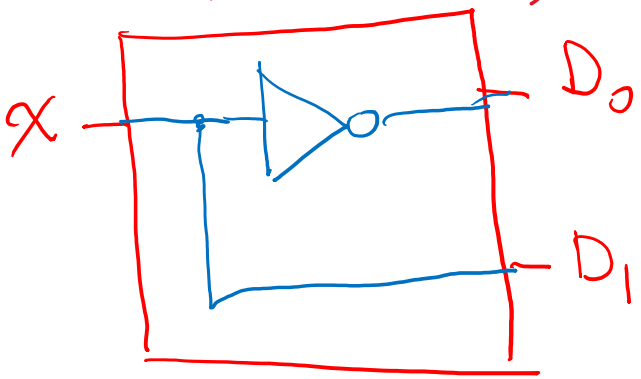
- A decoder is a combinational circuit that converts binary information from *n* input lines to an *m* (maximum of 2^n) unique output lines
  - *n*-to-*m*-line decoder (or $n \times m$ decoder)
  - Output variables are mutually exclusive because only one output can be equal to 1 at any time (the vary 1-minterm)
  - Can be implemented with AND gates

1-to-2-line decoder
(1x2 decoder)

x	D_0	D_1
0	1	0
1	0	1

$D_0 = x'$

$D_1 = x$



3-to-8-line Decoder (1/2)

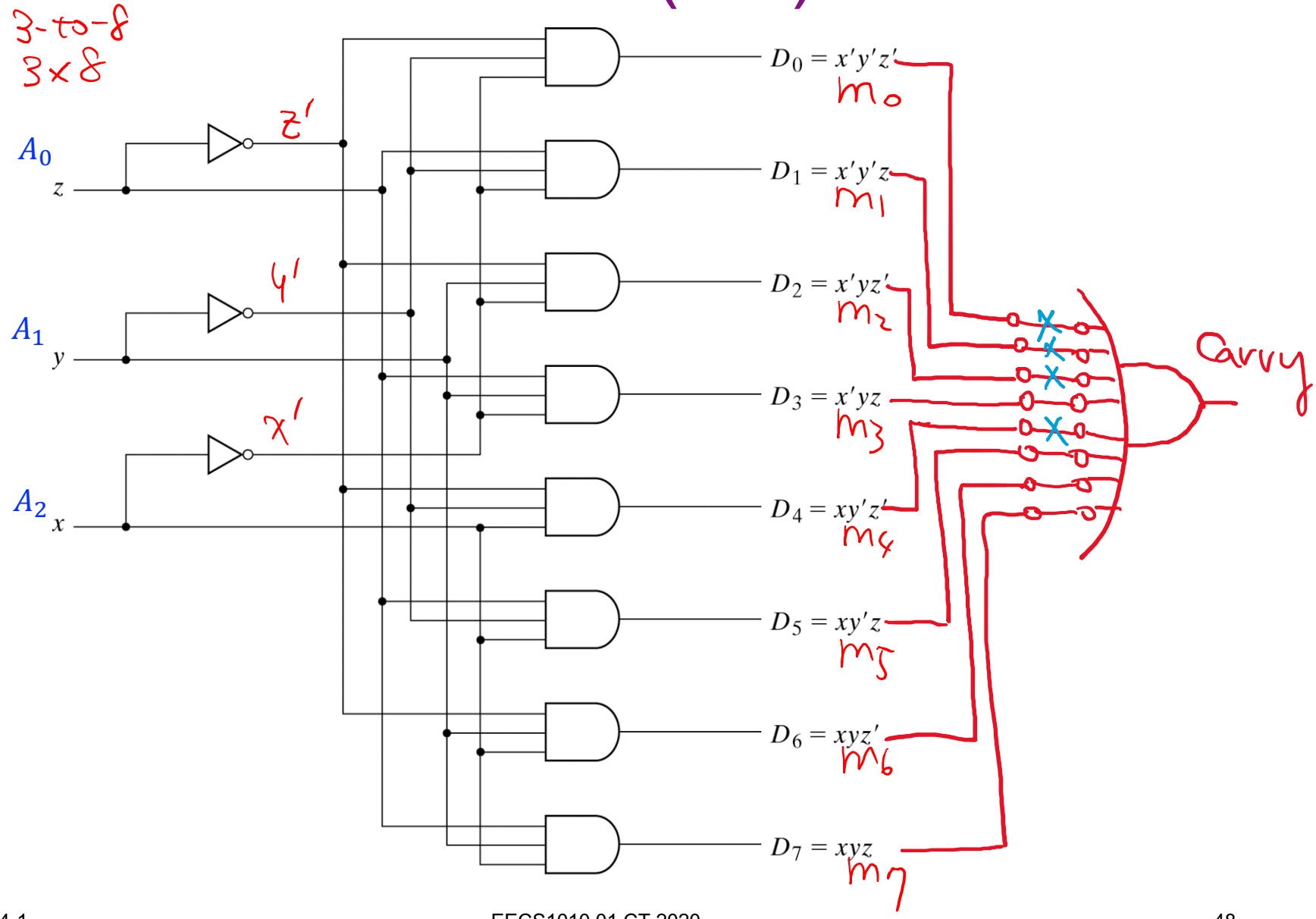
Truth Table of a Three-to-Eight-Line Decoder

Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
A_2	A_1	A_0								
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Binary Coding

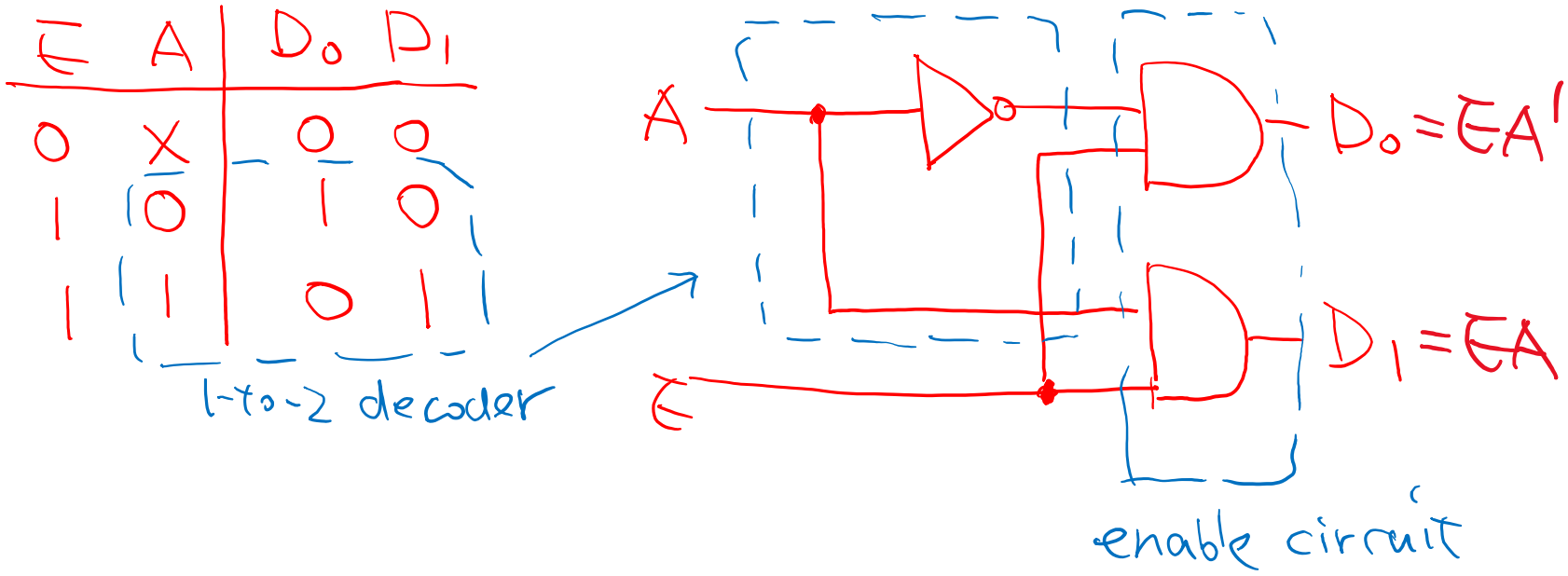
One-hot Coding

3-to-8-line Decoder (2/2)



Decoders with Enable Input (1/3)

- Line decoder with enable control (E)
- Also called demultiplexer (DMUX, DEMUX)



Decoders with Enable Input (2/3)

- Constructed with NAND gates
  - Decode minterms in their complemented form

反向邏輯

	A_1	A_0				
E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

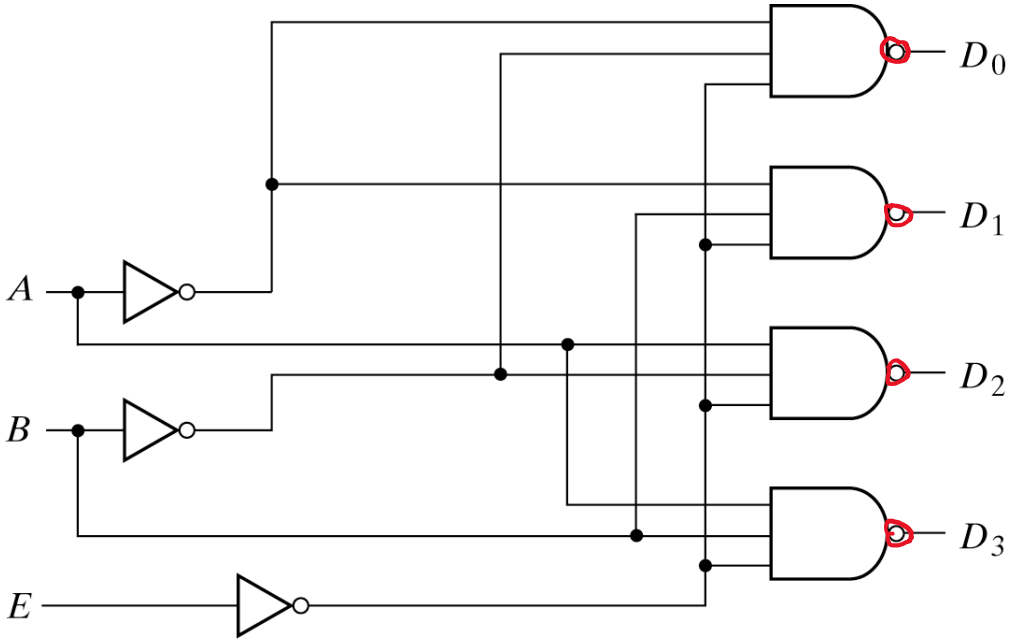
enable

$D_0 = (E'A'B')'$

$D_1 = (E'A'B)'$

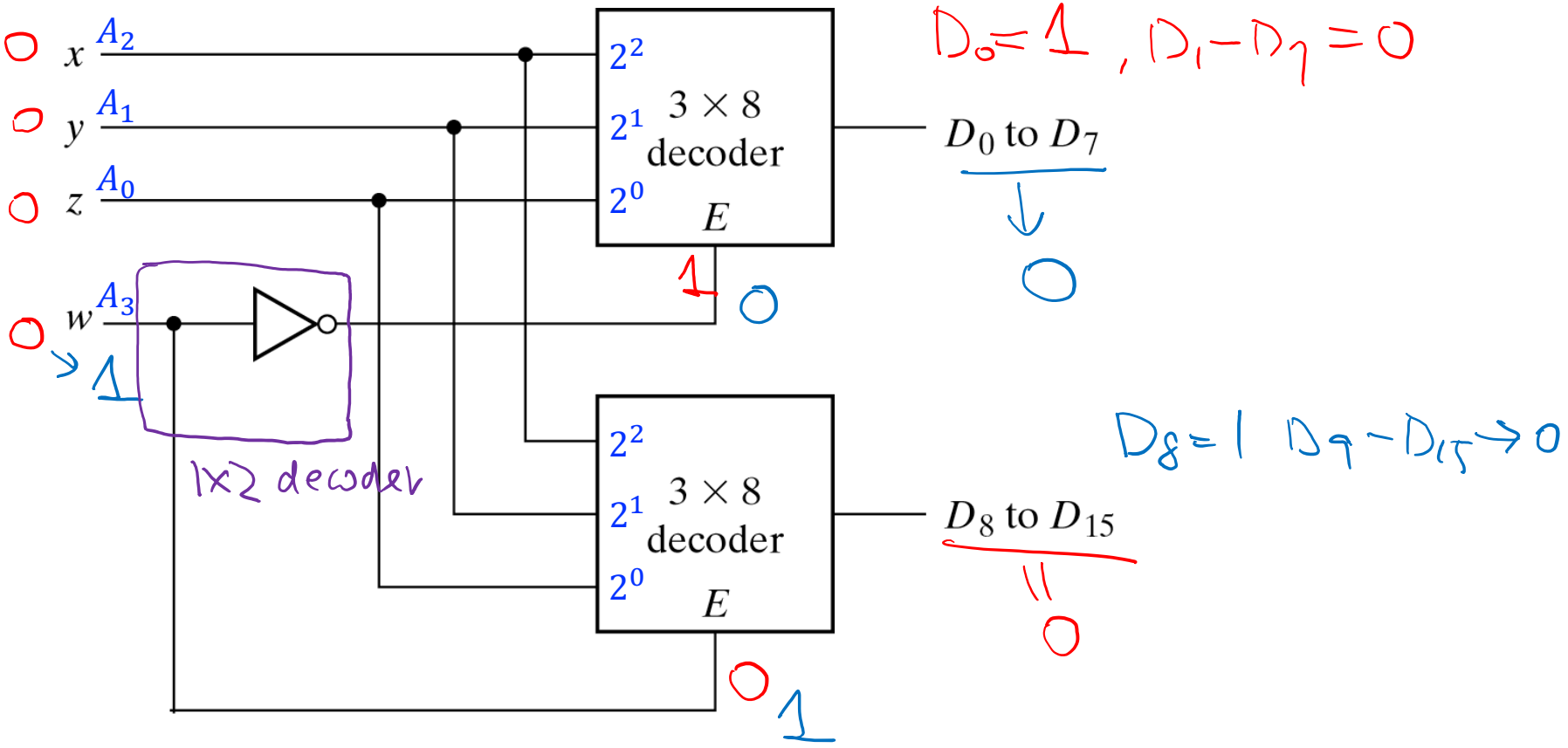
$D_2 = (E'AB')'$

$D_3 = (E'AB)'$



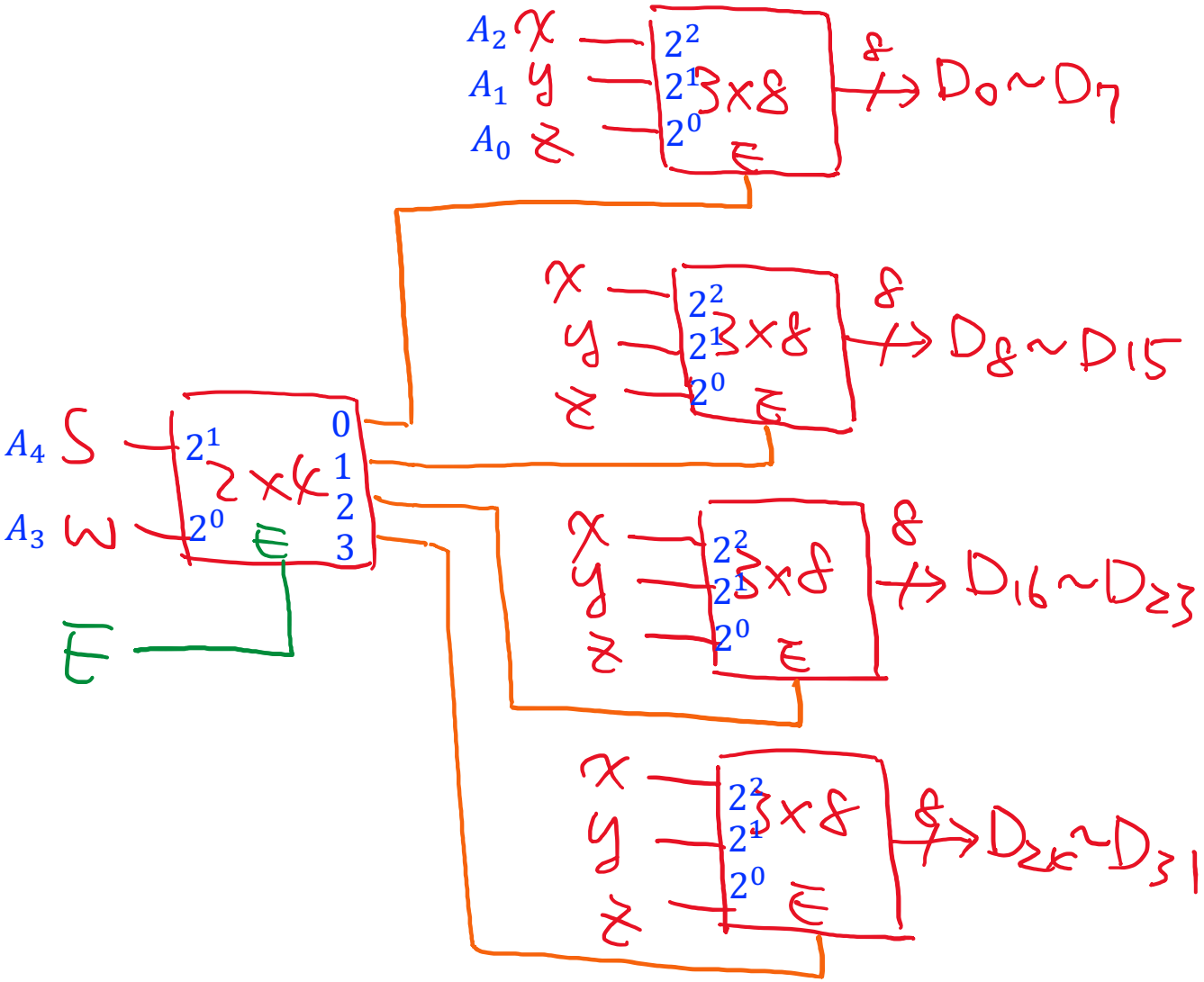
Decoder Expansion

- Larger decoders can be implemented with smaller decoders
  - 4x16 decoder with two 3x8 decoders
    - Input: $\{w, x, y, z\}$ (or $\{A_3, A_2, A_1, A_0\}$)



5x32 Decoder by Using 3x8 Decoders and 2x4 Decoder

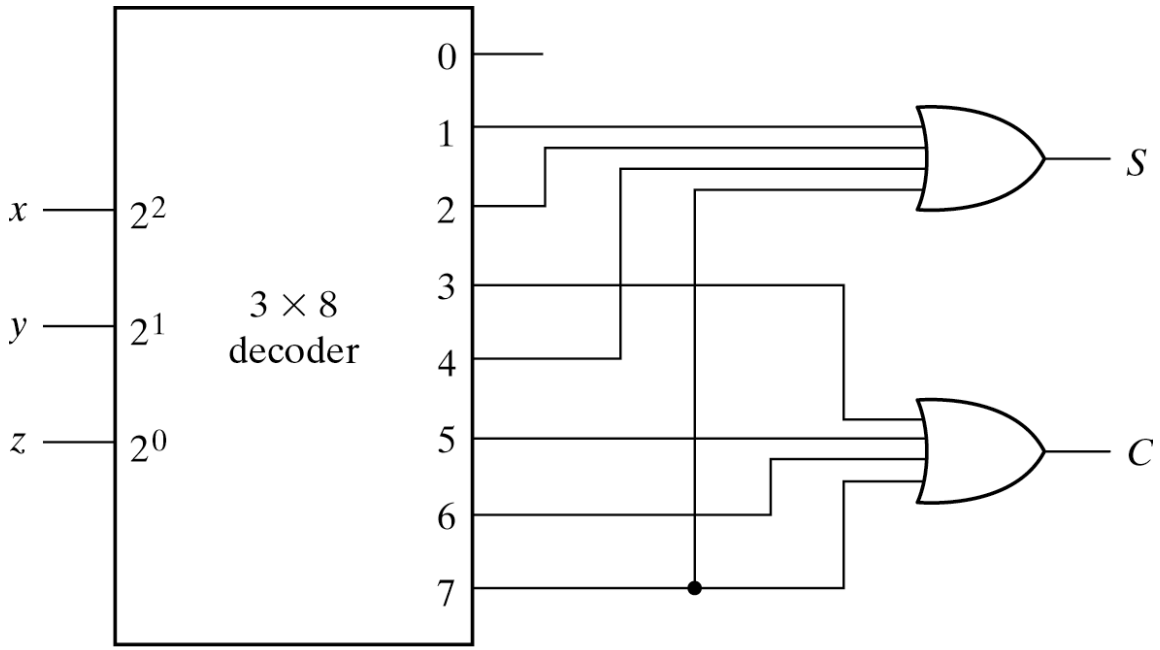
(s, w, x, y, z) (or $\{A_4, A_3, A_2, A_1, A_0\}$)



Combinational Logic Implementation (1/2)

- Any combinational circuit with *n* inputs and *m* outputs can be implemented with an *n*-to-2^n decoder in conjunction with *m* external OR gates
- A full-adder
  - $S(x, y, z) = \sum(1, 2, 4, 7)$
  - $C(x, y, z) = \sum(3, 5, 6, 7)$

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



Combinational Logic Implementation (2/2)

- A function with $k > 2^n/2$ minterms can be expressed in its complement form with $2^n - k$ minterms
  - NOR gates are used instead of OR gates
- If NAND gates are used for the decoder, the output OR gates are replaced by NAND gates
  - AND-OR => NAND-NAND

$f = \Sigma(0,1,2,3,5,6)$

(#1's > #0's)

$f' = \Sigma(4,7)$

(#0's > #1's)

取 f' 再加 INV ⇒ less fan-in