Lecture 15 – Combinational Circuits 4

Dr. Aftab M. Hussain,

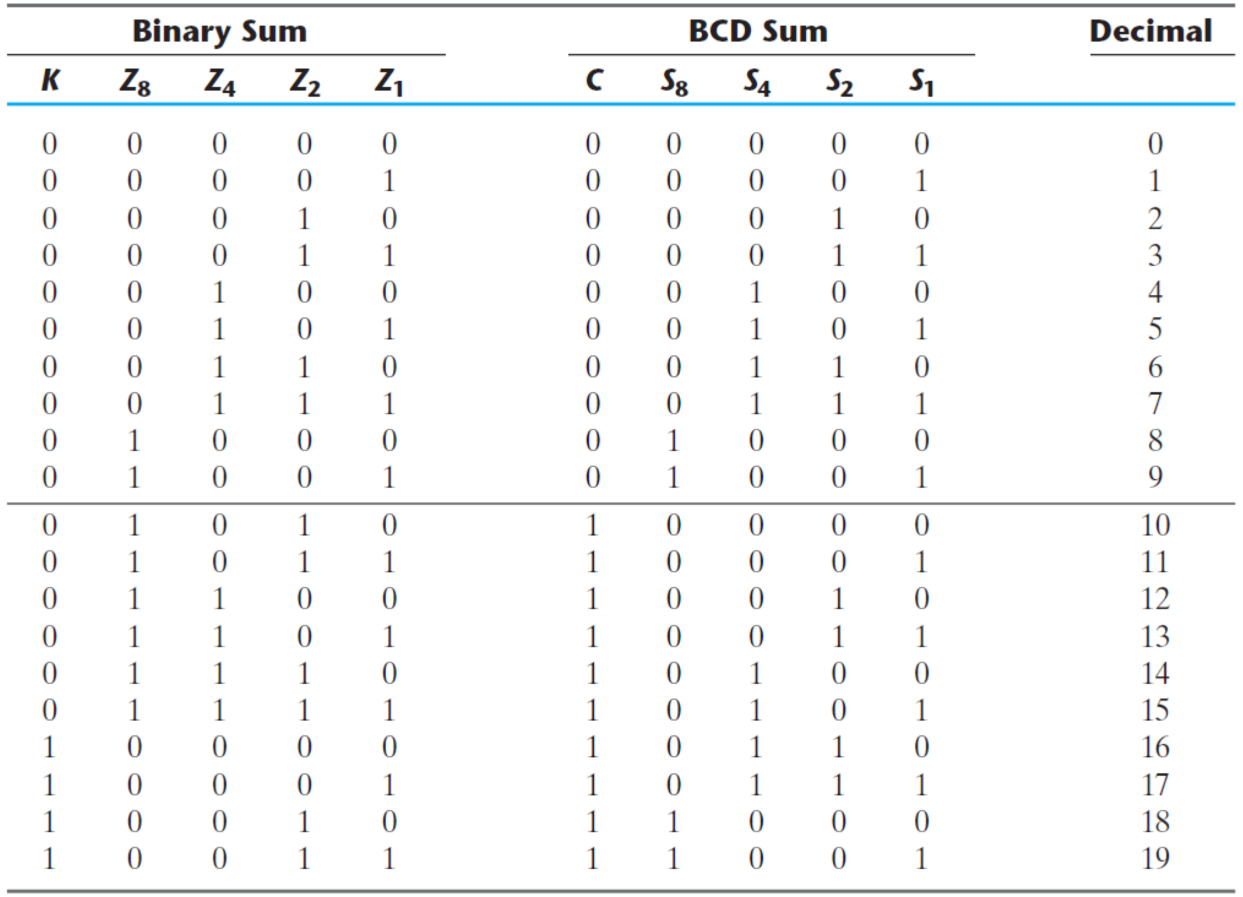
Assistant Professor, PATRIoT Lab, CVEST

Chapter 4

The BCD Adder

BCD adder

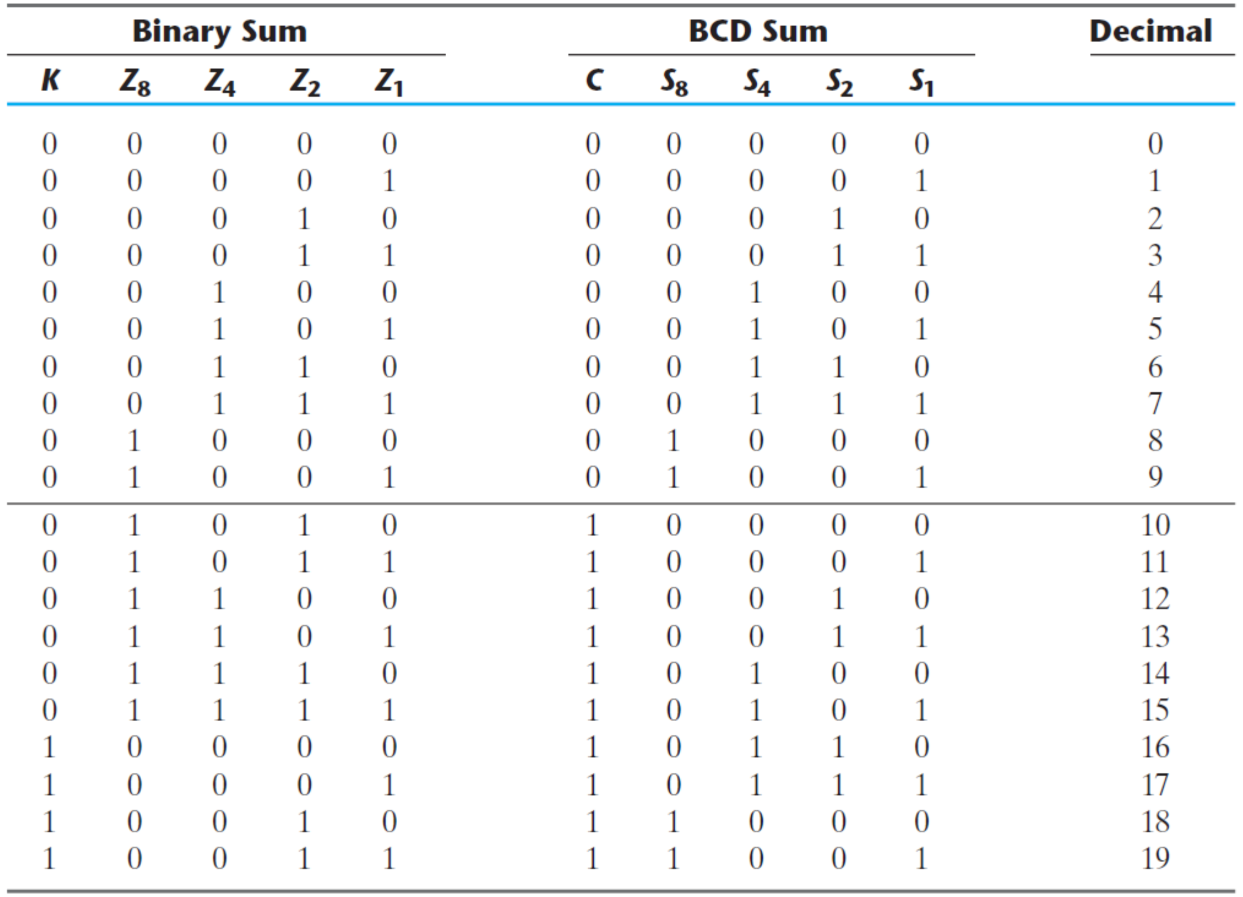
- Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage
- Since each input digit does not exceed 9, the output sum cannot be greater than 9 + 9 + 1 = 19, the 1 in the sum being an input carry
- Suppose we apply two BCD digits to a four-bit binary adder
- The adder will form the sum in *binary* and produce a result that ranges from 0 through 19



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

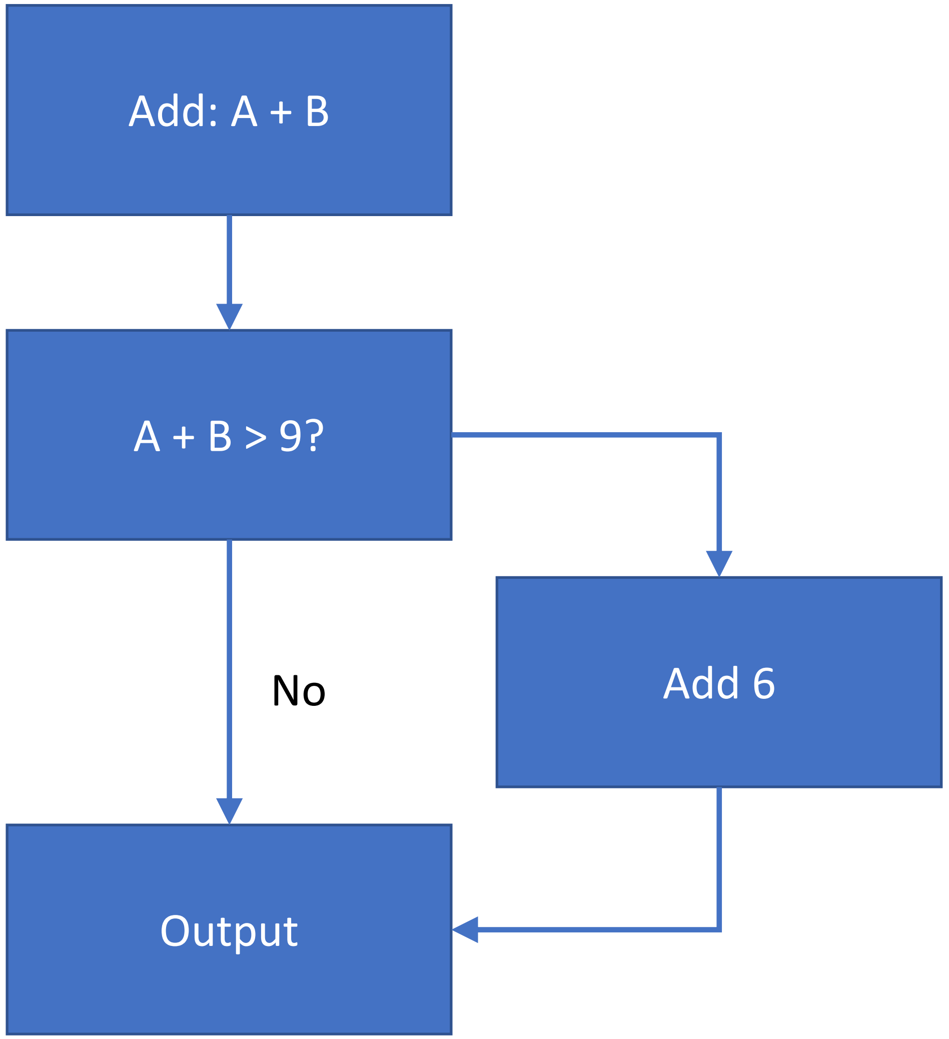
BCD adder

- In examining the contents of the table, it becomes apparent that when the binary sum is equal to or less than 1001, the corresponding BCD number is identical, and therefore no conversion is needed
- When the binary sum is greater than 1001, we obtain an invalid BCD representation
- The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required



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

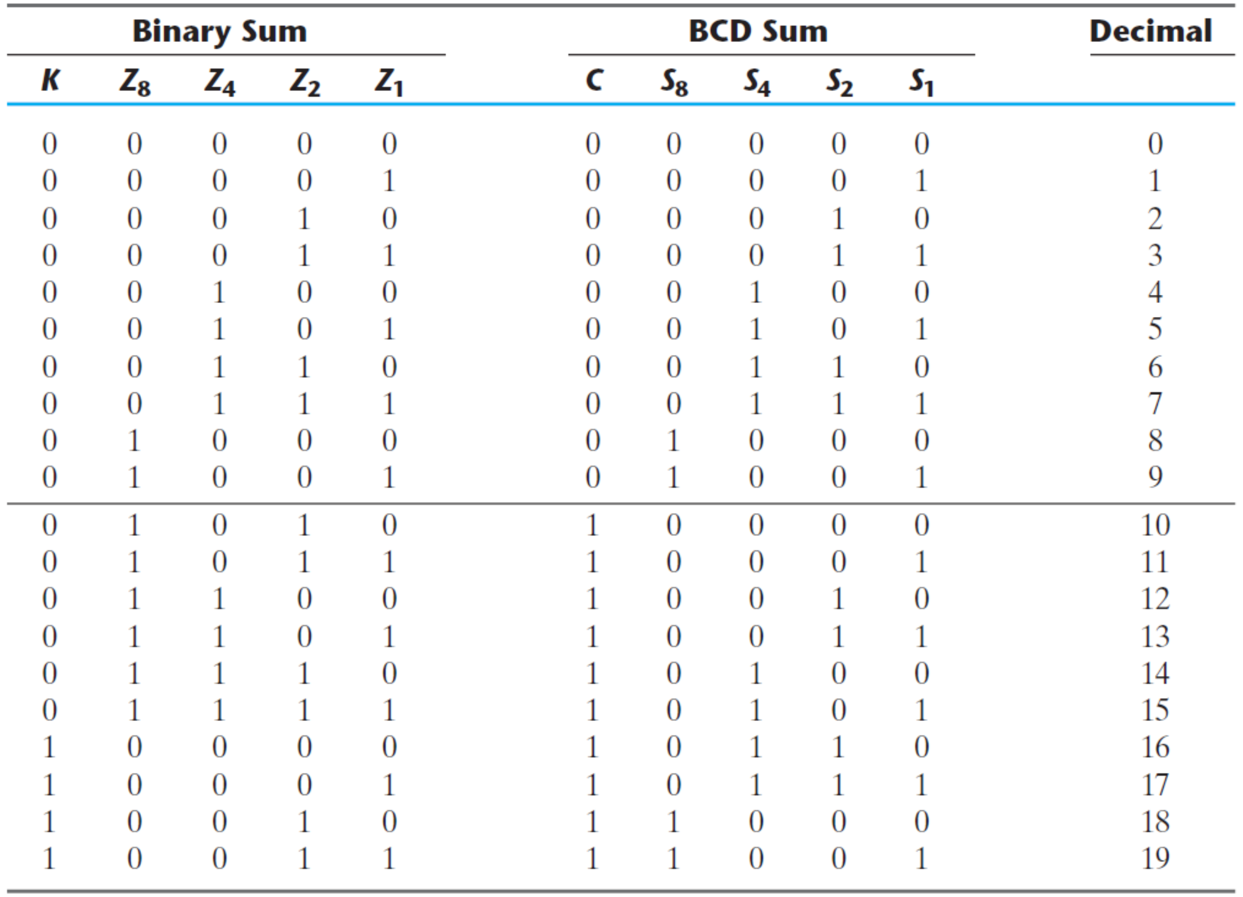
BCD adder - algorithm



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

BCD adder

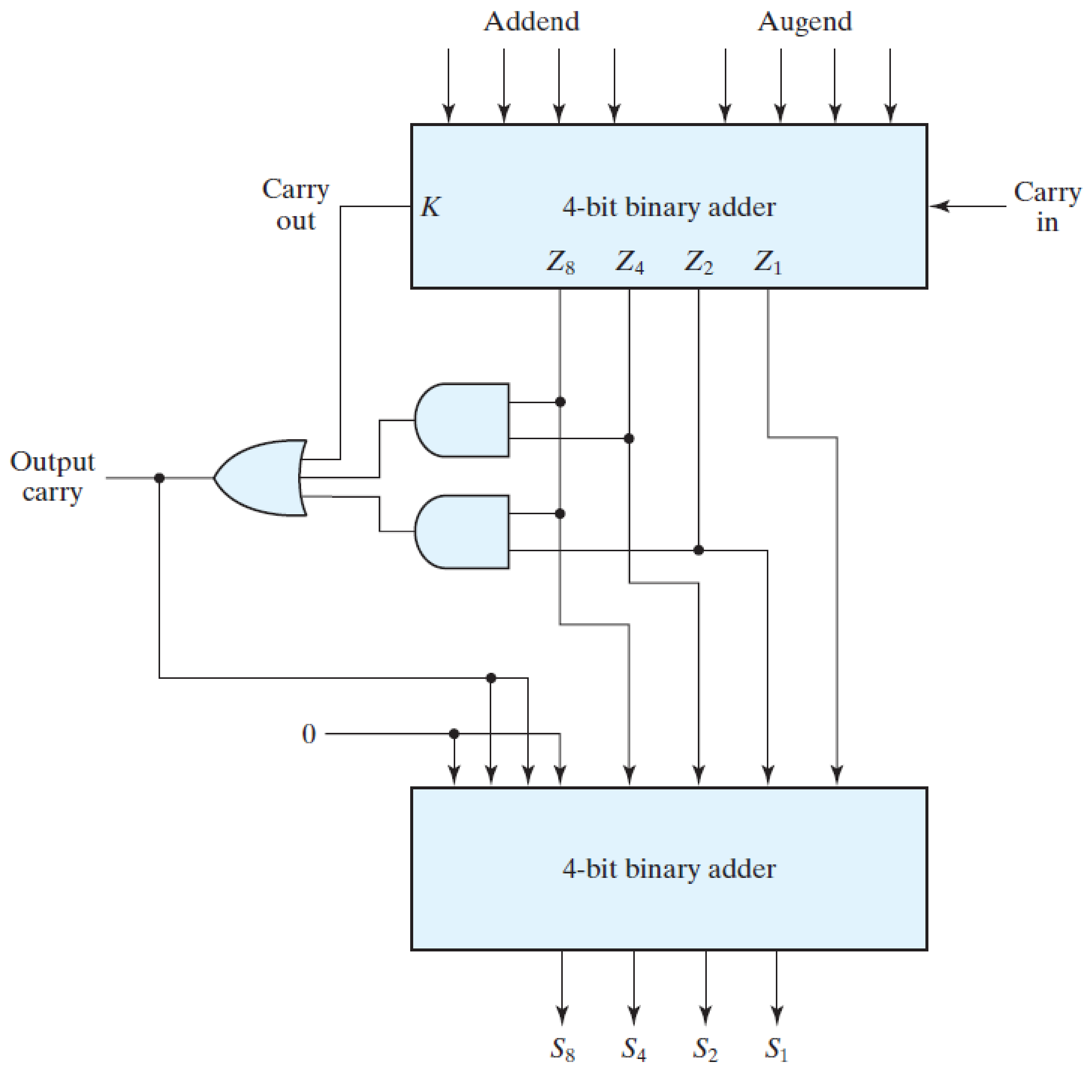
- The logic circuit that detects the necessary correction can be derived from the entries in the table
- It is obvious that a correction is needed when the binary sum has an output carry *K* = 1
- The other six combinations from 1010 through 1111 that need a correction have a 1 in position Z_8
- To distinguish them from binary 1000 and 1001, which also have a 1 in position Z_8, we specify further that either Z_4 or Z_2 must have a 1
- Thus, the condition for a correction and an output carry can be expressed by the Boolean function: $Cor = K + Z_8Z_4 + Z_8Z_2$
- When *Cor* = 1, it is necessary to add 0110 to the binary sum and provide an output carry for the next stage



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

BCD adder

- The logic circuit that detects the necessary correction can be derived from the entries in the table
- It is obvious that a correction is needed when the binary sum has an output carry *K* = 1
- The other six combinations from 1010 through 1111 that need a correction have a 1 in position Z_8
- To distinguish them from binary 1000 and 1001, which also have a 1 in position Z_8, we specify further that either Z_4 or Z_2 must have a 1
- Thus, the condition for a correction and an output carry can be expressed by the Boolean function: $Cor = K + Z_8Z_4 + Z_8Z_2$
- When *Cor* = 1, it is necessary to add 0110 to the binary sum and provide an output carry for the next stage



The Binary Multiplier

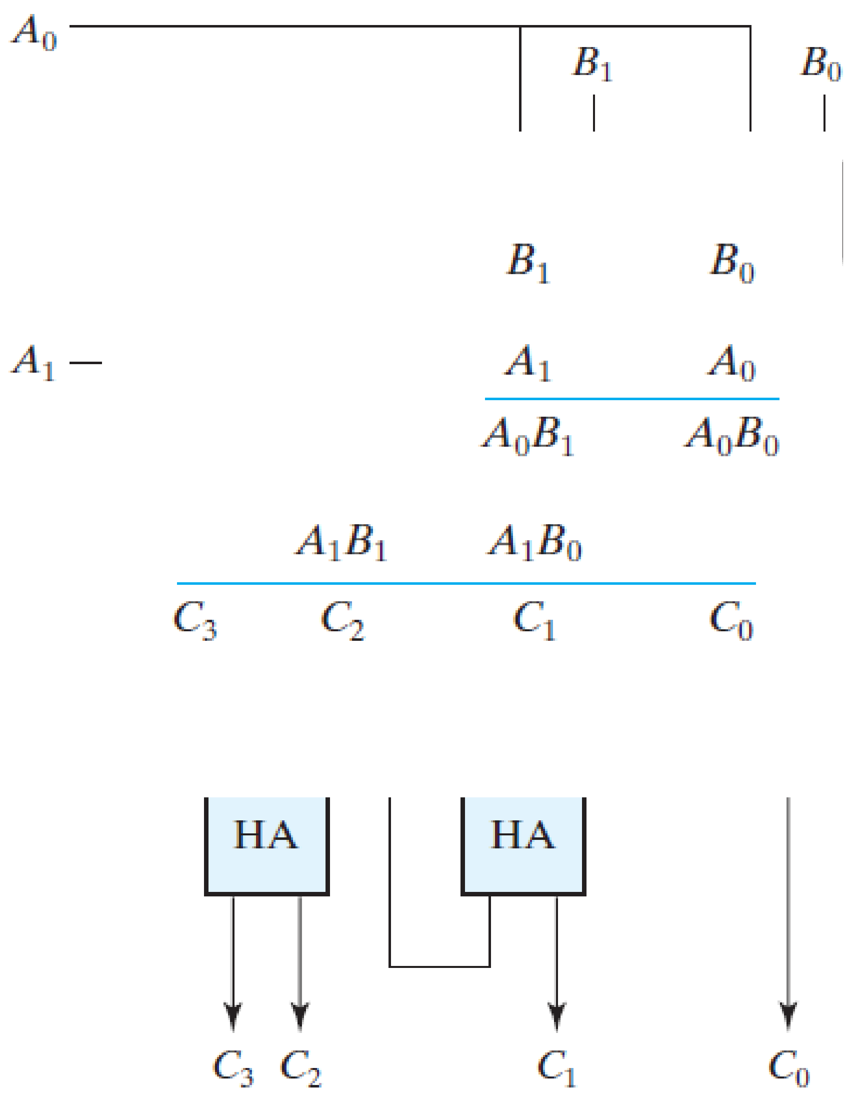
Binary multiplier

- Multiplication of binary numbers is performed in the same way as multiplication of decimal numbers
- The multiplicand is multiplied by each bit of the multiplier, starting from the least significant bit
- Each such multiplication forms a partial product
- Successive partial products are shifted one position to the left
- The final product is obtained from the sum of the partial products
- Consider a 2-bit x 2-bit multiplier. Number of inputs/outputs?

$$\begin{array}{cccc}
 & & B_1 & B_0 \\
 & & A_1 & A_0 \\
\cline{3-4}
 & & A_0B_1 & A_0B_0 \\
 & A_1B_1 & A_1B_0 & \\
\hline
C_3 & C_2 & C_1 & C_0
\end{array}$$

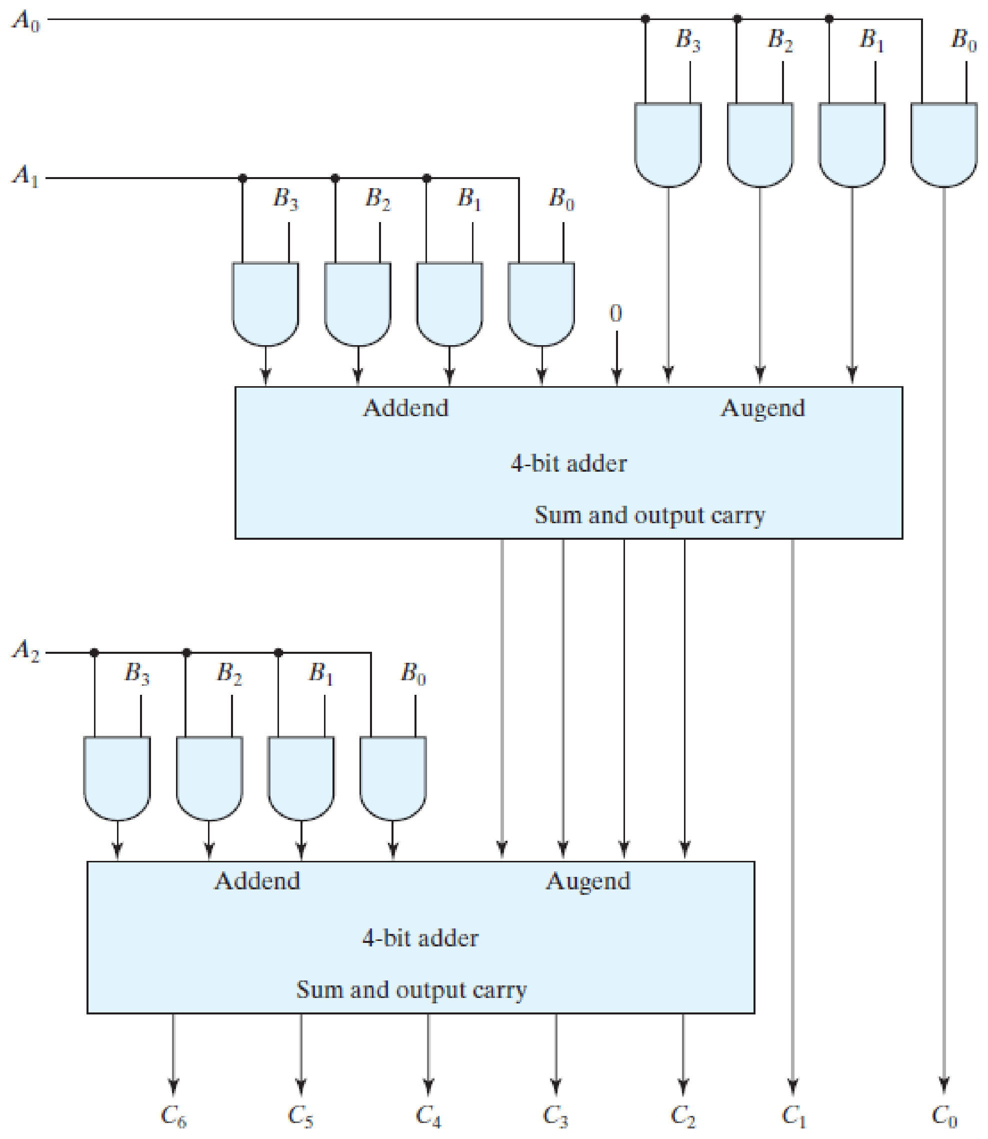
Binary multiplier

- The multiplicand bits are B_1 and B_0, the multiplier bits are A_1 and A_0, and the product is $C_3C_2C_1C_0$
- The first partial product is formed by multiplying B_1B_0 by A_0
- The multiplication of two bits such as A_0 and B_0 produces a 1 if both bits are 1; otherwise, it produces a 0
- This is identical to an AND operation
- Therefore, the partial products can be implemented with simple AND gates
- The two partial products are added with two half-adder (HA) circuits



Binary multiplier

- A combinational circuit binary multiplier with more bits can be constructed in a similar fashion
- A bit of the multiplier is ANDed with each bit of the multiplicand in as many levels as there are bits in the multiplier
- The binary output in each level of AND gates is added with the partial product of the previous level to form a new partial product
- The last level produces the product
- For *J* multiplier bits and *K* multiplicand bits, we need *J* * *K* AND gates and (*J* – 1) *K* -bit adders to produce a product of (*J* + *K*) bits
- Consider a multiplier circuit that multiplies a binary number represented by four bits by a number represented by three bits
- Let the multiplicand be represented by $B_3B_2B_1B_0$ and the multiplier by $A_2A_1A_0$



The Binary Comparator

Binary comparator

- The comparison of two numbers is an operation that determines whether one number is greater than, less than, or equal to the other number
- A *magnitude comparator* is a combinational circuit that compares two numbers A and B and determines their relative magnitudes
- The outcome of the comparison is specified by three binary variables that indicate whether $A > B$, $A = B$, or $A < B$
- On the one hand, the circuit for comparing two n -bit numbers has 2^{2n} entries in the truth table and becomes too cumbersome, even with $n = 3$
- On the other hand, as one may suspect, a comparator circuit possesses a certain amount of regularity

Binary comparator

- Consider two numbers, *A* and *B* , with four digits each $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$
- The two numbers are equal if all pairs of significant digits are equal: $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$, **and** $A_0 = B_0$
- To check bit-wise equality, we can use the XNOR gate

$$x_i = A_iB_i + A_i'B_i' \text{ for } i = 0, 1, 2, 3$$

- For equality to exist, all x_i variables must be equal to 1, a condition that dictates an AND operation of all variables:

$$(A = B) = x_3x_2x_1x_0$$

Binary comparator

- To determine whether *A* is greater or less than *B*, we inspect the relative magnitudes of pairs of significant digits, starting from the most significant position
- If the two digits of a pair are equal, we compare the next lower significant pair of digits
- The comparison continues until a pair of unequal digits is reached
- If the corresponding digit of *A* is 1 and that of *B* is 0, we conclude that *A* > *B*. If the corresponding digit of *A* is 0 and that of *B* is 1, we have *A* < *B*
- The comparison can be expressed logically by the two Boolean functions:

$$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$
$$(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$$

Binary comparator

$x_i = A_iB_i + A_i'B_i'$ for *i* = 0, 1, 2, 3

$(A = B) = x_3x_2x_1x_0$

$(A > B)$

$= A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1'$

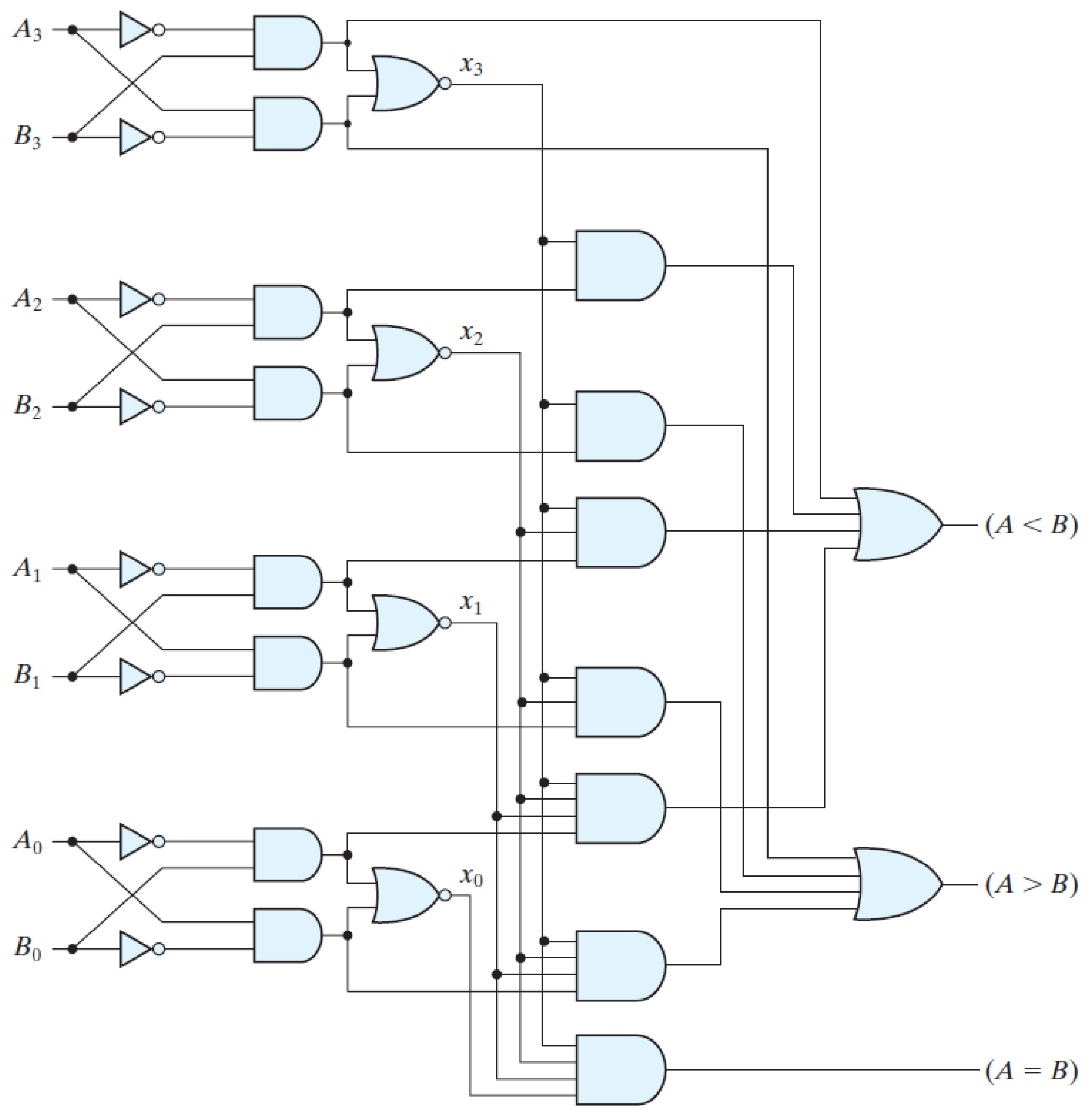
$+ x_3x_2x_1A_0B_0'$

$(A < B)$

$= A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1$

$+ x_3x_2x_1A_0'B_0$

Interesting: Can we prove that only one of (A=B), (A>B) and (A<B) will be “1” at any given time?


A_3
B_3
x_3
A_2
B_2
x_2
A_1
B_1
x_1
A_0
B_0
x_0
$(A < B)$
$(A > B)$
$(A = B)$