PATRIoT lab

Lecture 14 – Combinational circuits 3

Dr. Aftab M. Hussain,

Assistant Professor, PATRIoT Lab, CVEST

Chapter 4

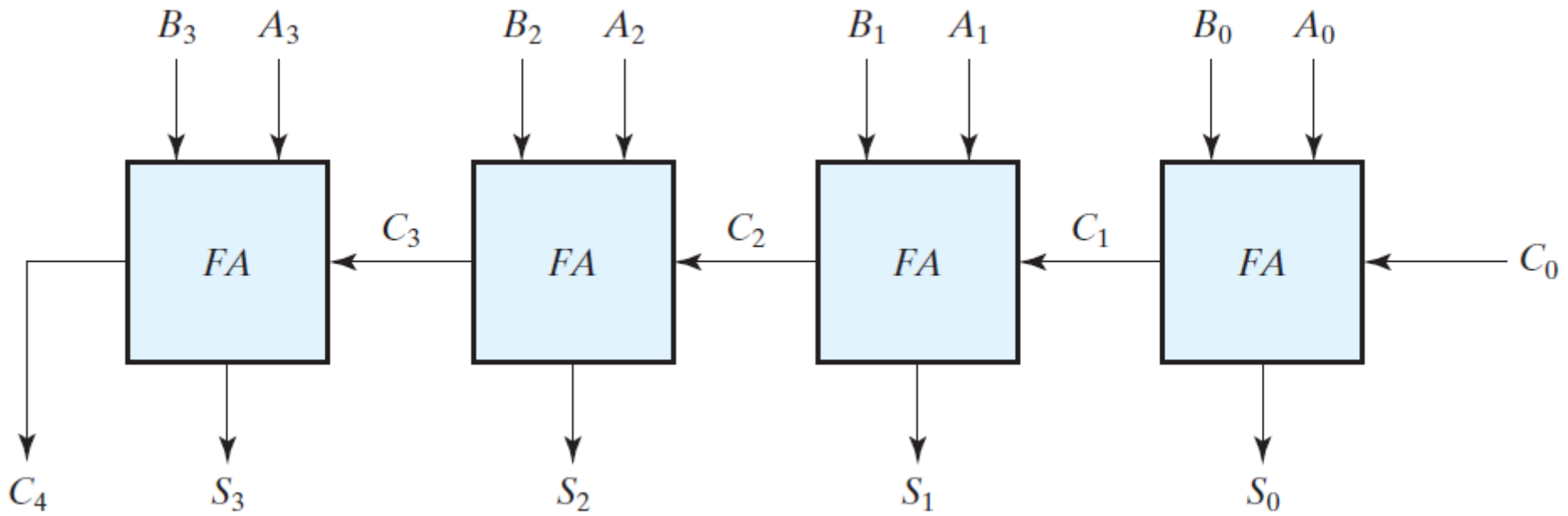
The Binary subtractor

Binary subtractor

- The subtraction of unsigned binary numbers can be done most conveniently by means of complements
- Remember that the subtraction *A* - *B* can be done by taking the 2's complement of *B* and **adding** it to *A*
- The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits
- The 1's complement can be implemented with inverters, and a 1 can be added to the sum through the input carry
- The circuit for subtracting *A* - *B* consists of an adder with inverters placed between each data input *B* and the corresponding input of the full adder
- **The input carry C_0 must be equal to 1 when subtraction is performed**
- The operation thus performed becomes *A,* plus the 1's complement of *B* , plus 1. This is equal to *A* plus the 2's complement of *B*
- That gives *A* - *B* if $A \geq B$ or the 2's complement of *B* - *A* if $A < B$

Can we combine the binary adder & subtractor

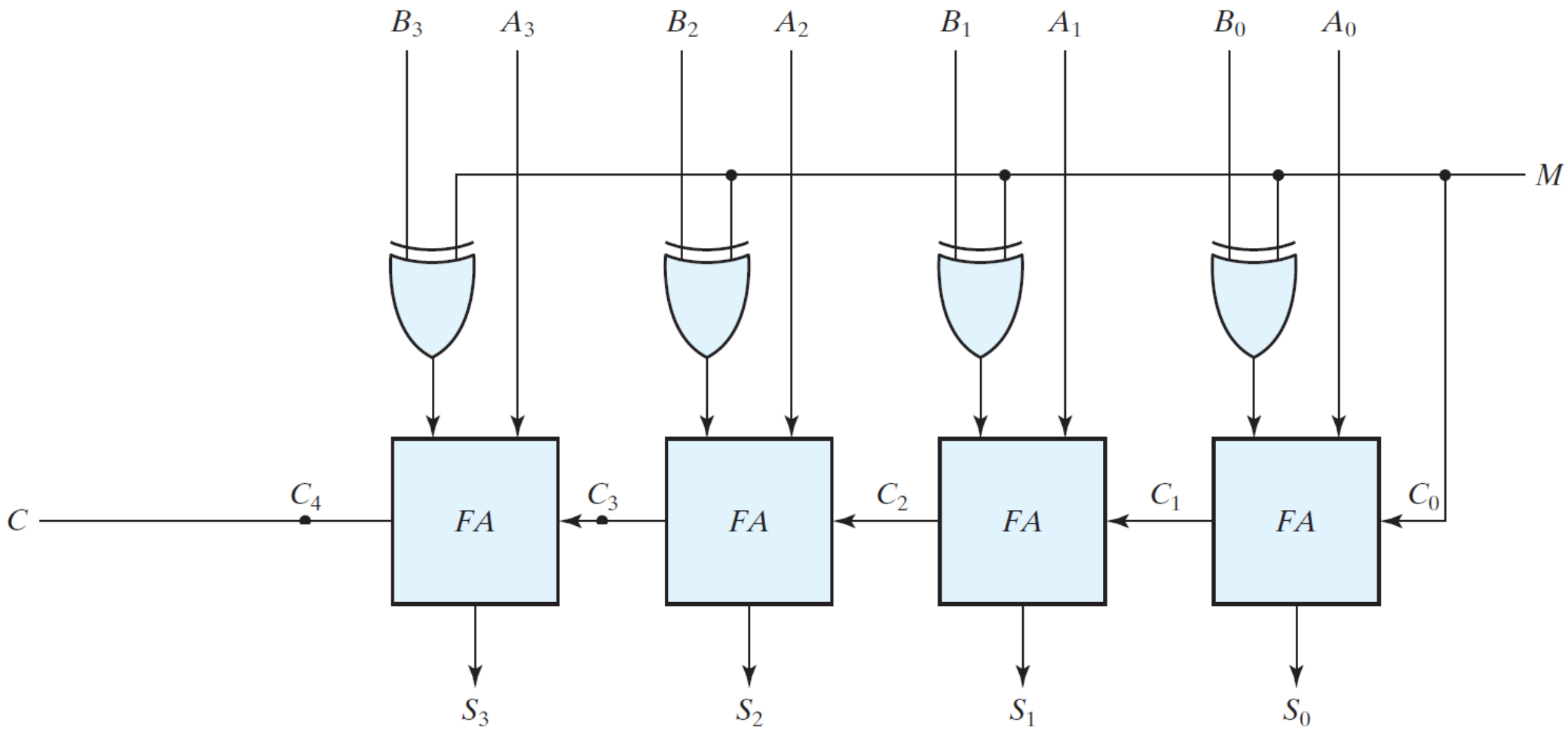
- Both use the 4-bit full adder. In one case, we use B and in another we use inverted B



Binary adder-subtractor

- Here is some magic: The addition and subtraction operations can be combined into one circuit
- The mode input *M* controls the operation
- When *M* = 0, the circuit is an adder, and when *M* = 1, the circuit becomes a subtractor
- When *M* = 0, the full adders receive the value of *B* , the input carry is 0, and the circuit performs *A* + *B*
- When *M* = 1, the full adders receive *B'* and $C_0 = 1$
- Thus, the *B* inputs are all complemented and a 1 is added through the input carry
- The circuit performs the operation *A* plus the 2's complement of *B*

Binary adder-subtractor



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

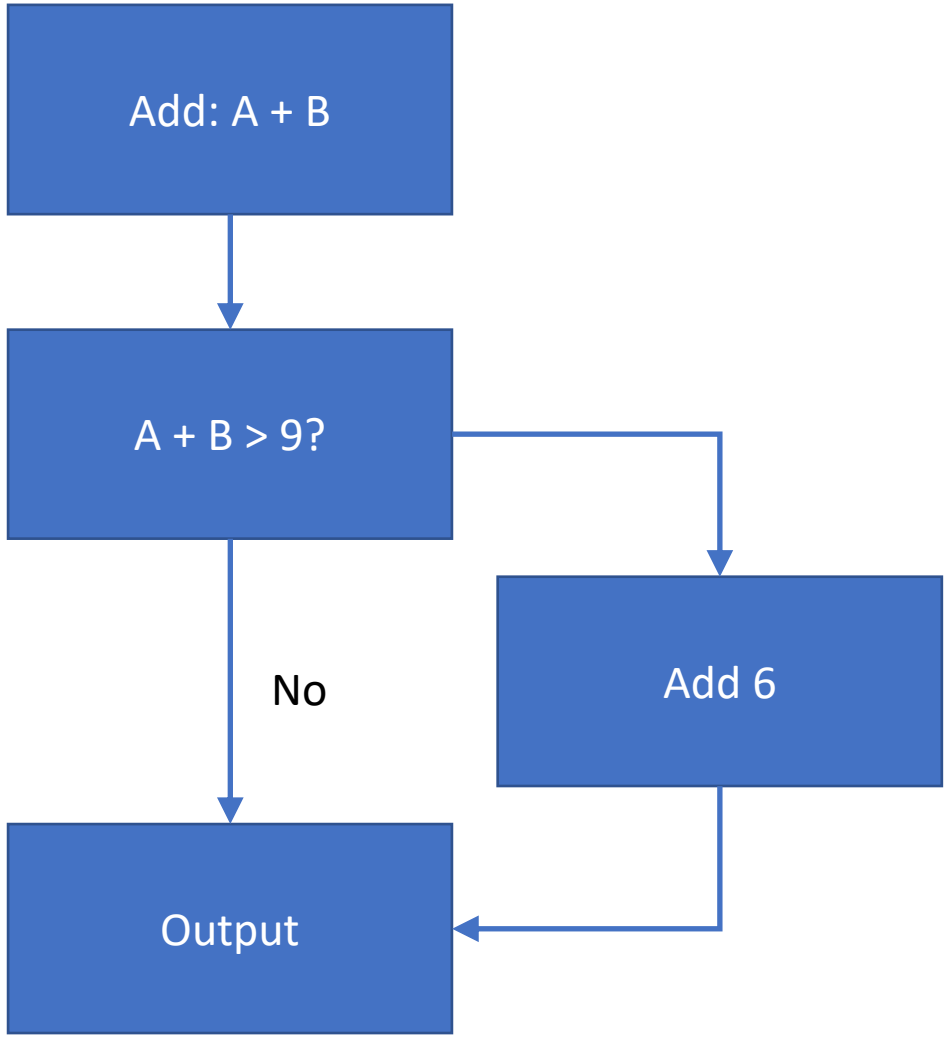
Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	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

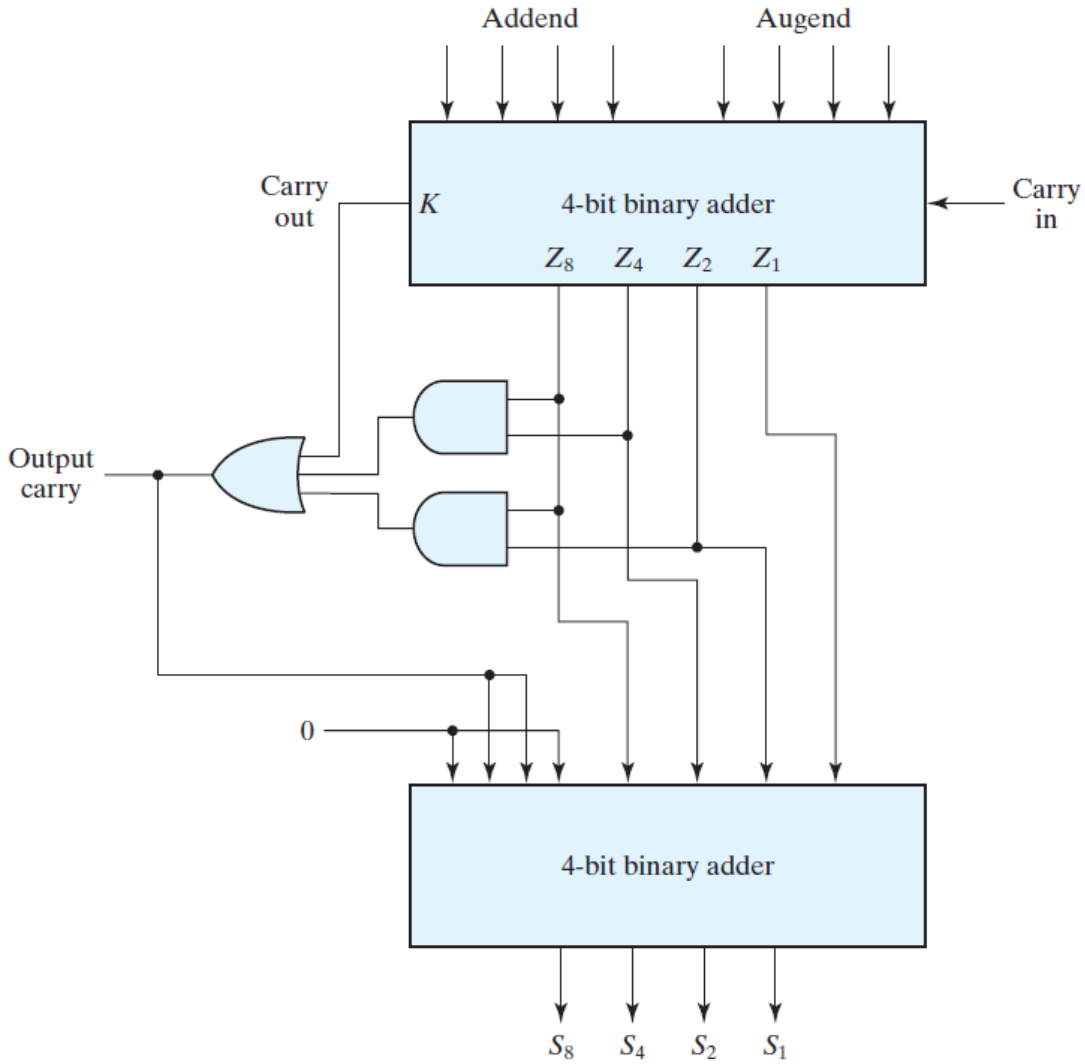
Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	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

BCD adder

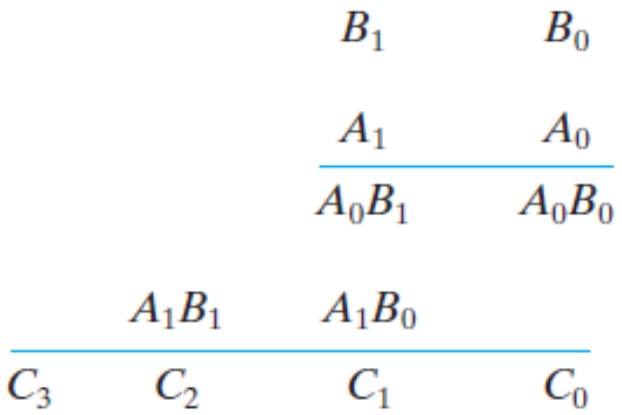
- The logic circuit that detects the necessary correction can be derived from the entries in the table
- It is obvious that a correction is needed when the binary sum has an output carry $K = 1$
- The other six combinations from 1010 through 1111 that need a correction have a 1 in position Z_8
- To distinguish 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

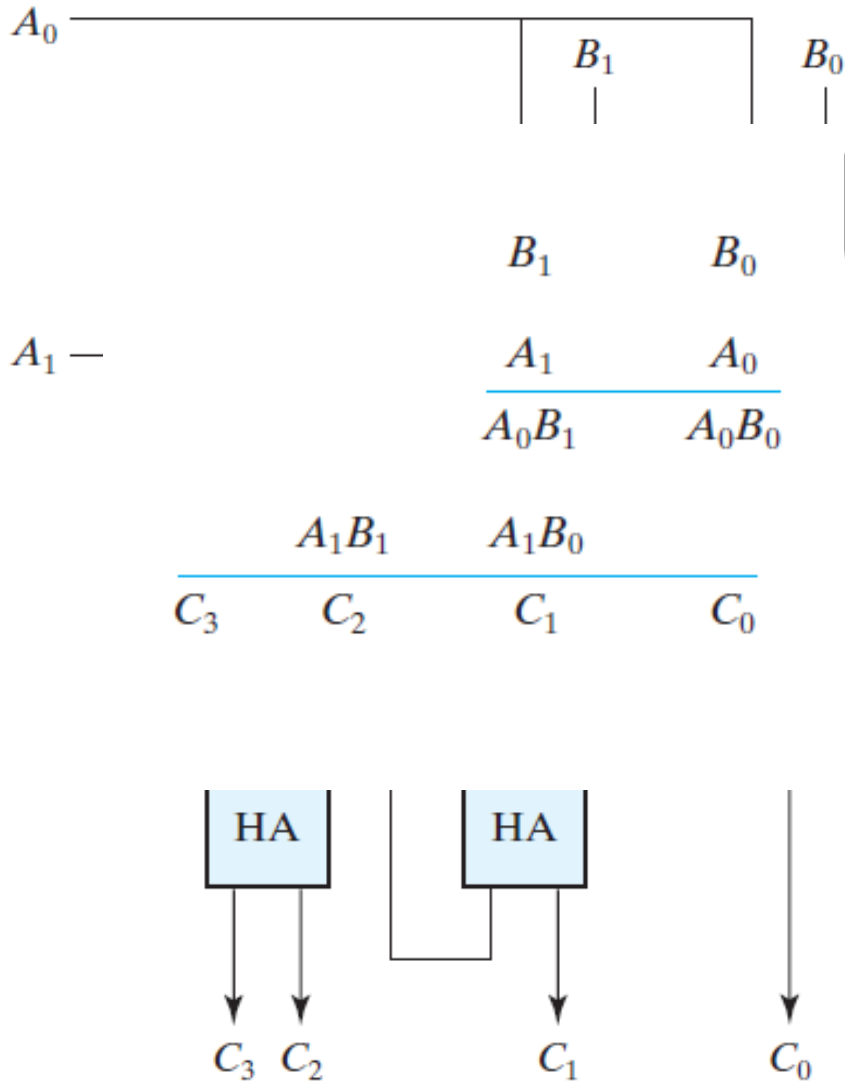
Binary multiplier

- 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?



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 $A_2A_1A_0$

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 $A_2A_1A_0$