Standard representation for logic functions

1. ***Sum-of-products (SOP) form:***
   - This form is also called the Disjunctive Normal Form (DNF).
   - For example, $f(A,B,C) = \bar{A}B + \bar{B}C$

2. ***Product-of-sums (POS) form:***
   - This form is also called the Conjunctive Normal Form (CNF).
   - The function of above equation may also be written in the form shown in equation below.
   - By using multiplying it out and using the consensus theorem, we can see that it is the same as

$$f(A,B,C) = (\bar{A} + \bar{B})(B + C)$$

3. ***Standard sum-of-products form:***
   - This form is also called Disjunctive Canonical Form (DCF).
   - It is also called the Expanded Sum of Products Form or Canonical Sum-of-Products Form.
   - In this form, the function is the sum of a number of product terms where each product term contains all the variables of the function either in complemented or uncomplemented form.
   - This can be derived from the truth table by finding the sum of all the terms that correspond to those combinations (rows) for which 'f ' assumes the value 1.
   - It can also be obtained from the SOP form algebraically as shown below.

$$\begin{aligned} f(A,B,C) = \bar{A}B + \bar{B}C &= \bar{A}B\,(C + \bar{C}) + (A + \bar{A})\bar{B} \\ &= \bar{A}BC + \bar{A}B\bar{C} + A\bar{B}C + \bar{A}\bar{B}C \end{aligned}$$

   - A product term which contains all the variables of the function either in complemented or uncomplemented form is called a minterm.
   - A minterm assumes the value 1 only for one combination of the variables. An n variable function can have in all 2^n minterms.
   - The sum of the minterms whose value is equal to 1 is the standard sum of products form of the function.
   - The minterms are often denoted as m_0, m_1, m_2, ..., where the suffixes are the decimal codes of the combinations.
   - For a 3-variable function $m_0 = \bar{A}\bar{B}\bar{C}$, $m_1 = \bar{A}\bar{B}C$, $m_2 = \bar{A}B\bar{C}$, $m_3 = \bar{A}BC$, $m_4 = A\bar{B}\bar{C}$, $m_5 = A\bar{B}C$, $m_6 = AB\bar{C}$, $m_7 = ABC$.
   - Another way of representing the function in canonical SOP form is by showing the sum of minterms for which the function equals 1.
   - Thus $f(A,B,C) = m_1 + m_2 + m_3 + m_5$
   - Yet another way of representing the function in DCF is by listing the decimal codes of the minterms for which f = 1.
   - Thus $f(A,B,C) = \sum_m (1,2,3,5)$

4. ***Standard product-of-sums form:***
   - This form is also called Conjunctive Canonical Form (CCF).
   - It is also called Expanded Product-of-Sums Form or Canonical Product-of-Sums Form.
   - This is derived by considering the combinations for which f = 0. Each term is a sum of all the variables.
   - A variable appears in uncomplemented form if it has a value of 0 in the combination and appears in complemented form if it has a value of l in the combination.

$$f(A,B,C) = (\bar{A} + \bar{B})(B + C) = (\bar{A} + \bar{B} + C\bar{C})\,(A\bar{A} + B + C)$$

$$= (\bar{A} + \bar{B} + C)(\bar{A} + \bar{B} + \bar{C})(A + B + C)(\bar{A} + B + C)$$

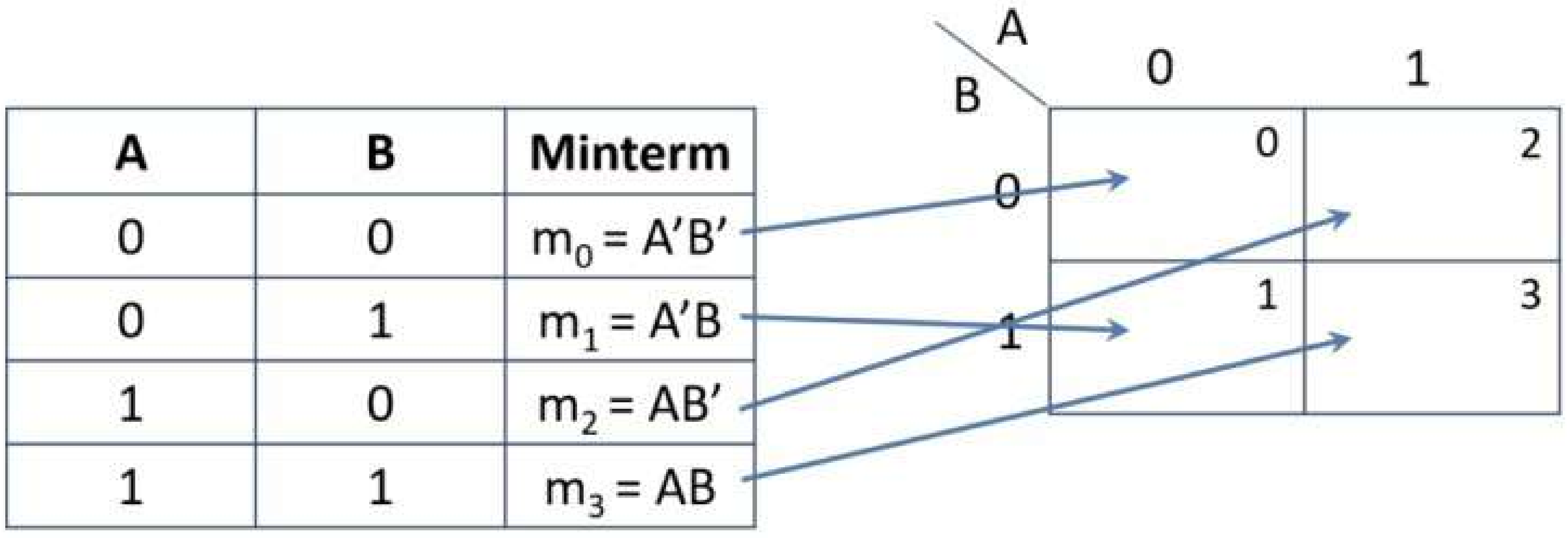
- A sum term which contains each of the n variables in either complemented or uncomplemented form is called a maxterm.
- A maxterm assumes the value 0 only for one combination of the variables. For all other combinations it will be 1.
- There will be at the most 2^n maxterms. The product of maxterms corresponding to the rows for which f = 0, is the standard or canonical product of sums form of the function.
- Maxterms are often represented as M_0, M_1, M_2, ..., where the suffixes denote their decimal code.
- Thus, the CCF of function f may be written as $f(A,B,C) = M_0 \cdot M_4 \cdot M_6 \cdot M_7$
- Expression can also be expressed as $f(A,B,C) = \prod_M(0,4,6,7)$
- Where ∏ represents the product of all maxterms whose decimal code is given within the parenthesis.

Karnaugh Map (K-Map)

- The Karnaugh map (K-map) method is a systematic method of simplifying the Boolean expression.
- The K-map is a chart or a graph, composed of an arrangement of adjacent cells, each representing a particular combination of variables in sum or product form.
- The output values placed in each cell are derived from the minterms of a Boolean function.
- A *minterm* is a product term that contains all of the function's variables exactly once, either complemented or not complemented.

Two-variable K-Map

- The two variable expression can have 2^2 = 4 possible combinations of the input variables A and B.
- Each of these combinations, A'B', A'B, AB', and AB (in SOP form) is called minterm.
- These possible combinations can be represented in table and by the map as follows:

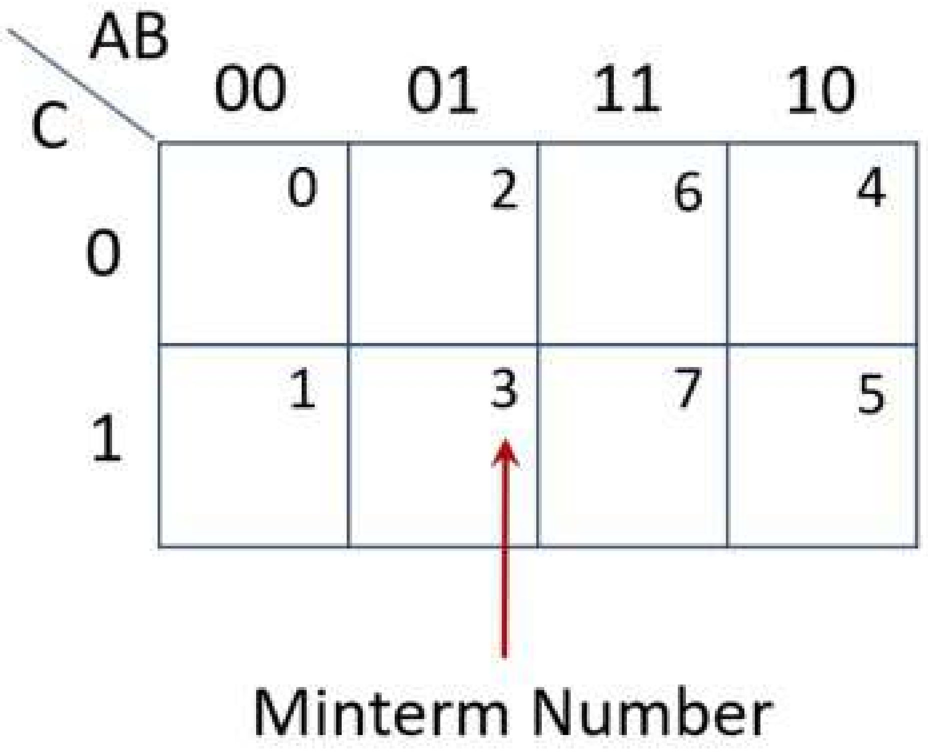


A	B	Minterm
0	0	m_0 = A'B'
0	1	m_1 = A'B
1	0	m_2 = AB'
1	1	m_3 = AB

Three-variable K-Map

- A function in three variable (A, B, C) expressed in the standard SOP form can have eight possible combinations: A'B'C', A'B'C, A'BC', A'BC, AB'C', AB'C, ABC' and ABC.
- Each one of these combinations designated by m_0, m_1, m_2, m_3, m_4, m_5, m_6, and m_7, respectively, is called minterm.
- In the standard POS form, the eight possible combinations are: A+B+C, A+B+C', A+B'+C, A+B'+C', A'+B+C, A'+B+C', A'+B'+C, and A'+B'+C'.
- Each one of these combinations designated by M_0, M_1, M_2, M_3, M_4, M_5, M_6, and M_7, respectively, is called maxterm.

A	B	C	Minterm
0	0	0	m_0 = A'B'C'
0	0	1	m_1 = A'B'C
0	1	0	m_2 = A'BC'
0	1	1	m_3 = A'BC
1	0	0	m_4 = AB'C'
1	0	1	m_5 = AB'C
1	1	0	m_6 = ABC'
1	1	1	m_7 = ABC



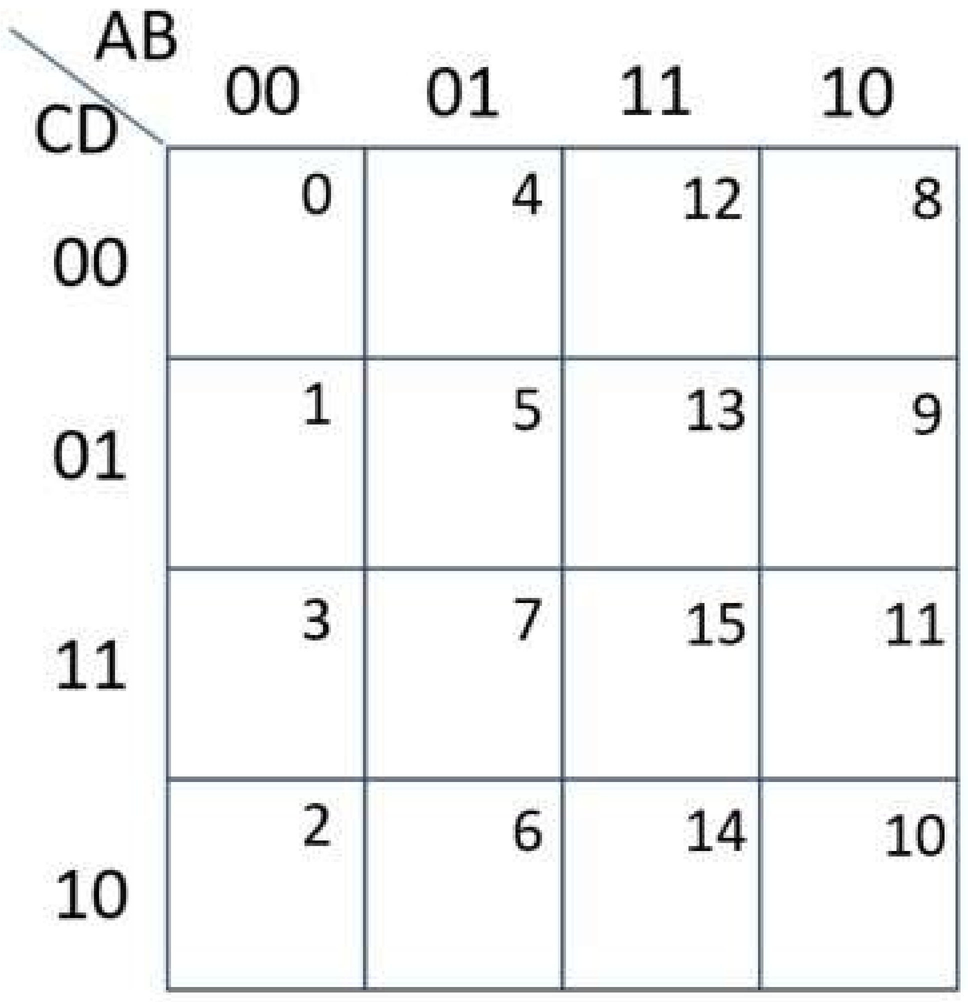
C \ AB	00	01	11	10
0	0	2	6	4
1	1	3	7	5

Four-variable K-Map

- The four variables A, B, C and D have sixteen possible combinations that can be represented by the map as follows

A	B	C	D	Minterm
0	0	0	0	m_0 = A'B'C'D'
0	0	0	1	m_1 = A'B'C'D
0	0	1	0	m_2 = A'B'CD'
0	0	1	1	m_3 = A'B'CD
0	1	0	0	m_4 = A'BC'D'
0	1	0	1	m_5 = A'BC'D
0	1	1	0	m_6 = A'BCD'
0	1	1	1	m_7 = A'BCD

A	B	C	D	Minterm
1	0	0	0	m_8 = AB'C'D'
1	0	0	1	m_9 = AB'C'D
1	0	1	0	m_{10} = AB'CD'
1	0	1	1	m_{11} = AB'CD
1	1	0	0	m_{12} = ABC'D'
1	1	0	1	m_{13} = ABC'D
1	1	1	0	m_{14} = ABCD'
1	1	1	1	m_{15} = ABCD

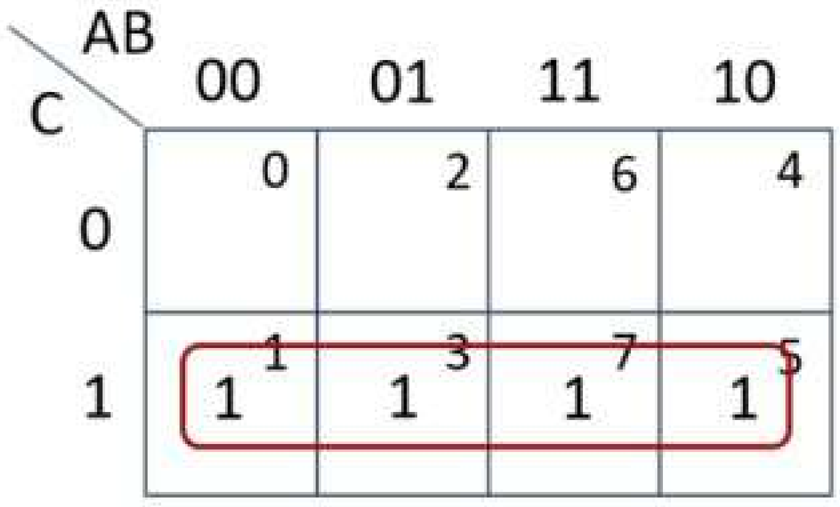


CD \ AB	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

Reduction using K-Map

- Squares which are physically adjacent to each other or which can be made adjacent by wrapping the map around from left to right or top to bottom can be combined to form bigger squares.
- The bigger squares (2 squares, 4 squares, 8 squares, etc.) must form either a geometric square or rectangle.
- For the minterms or maxterms to be combinable into bigger squares, it is necessary but not sufficient that their binary designations differ by a power of 2.

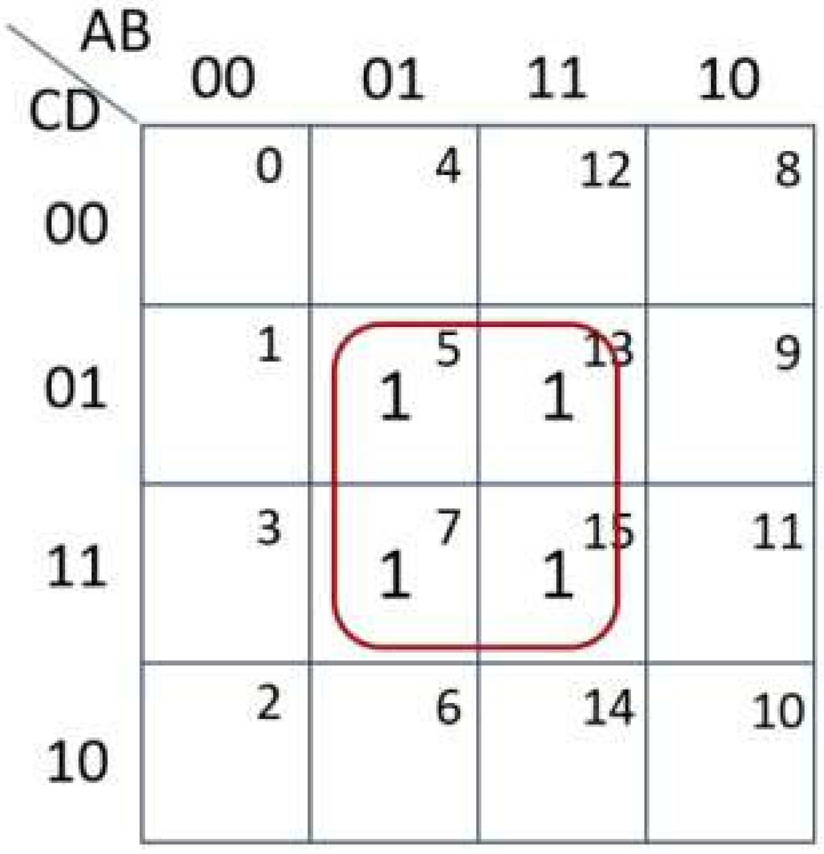
Example - 1 Reduce f(A, B, C) = A'B'C + A'BC + AB'C + ABC



C \ AB	00	01	11	10
0	0	2	6	4
1	1 1	3 1	7 1	5 1

Answer: f = C

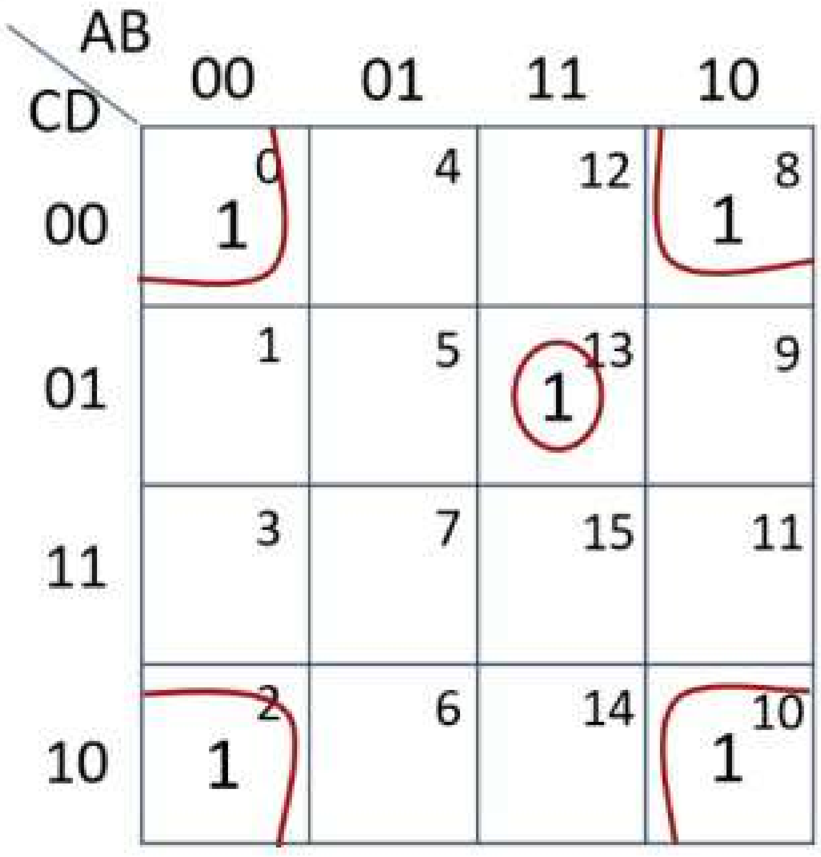
Example - 2 Reduce f(A, B, C, D) = ABC'D + ABCD + A'BC'D + A'BCD



CD \ AB	00	01	11	10
00	0	4	12	8
01	1	5 1	13 1	9
11	3	7 1	15 1	11
10	2	6	14	10

Answer: f = BD

Example - 3 Reduce f(A, B, C, D) = $\sum_m (0, 2, 8, 10, 13)$

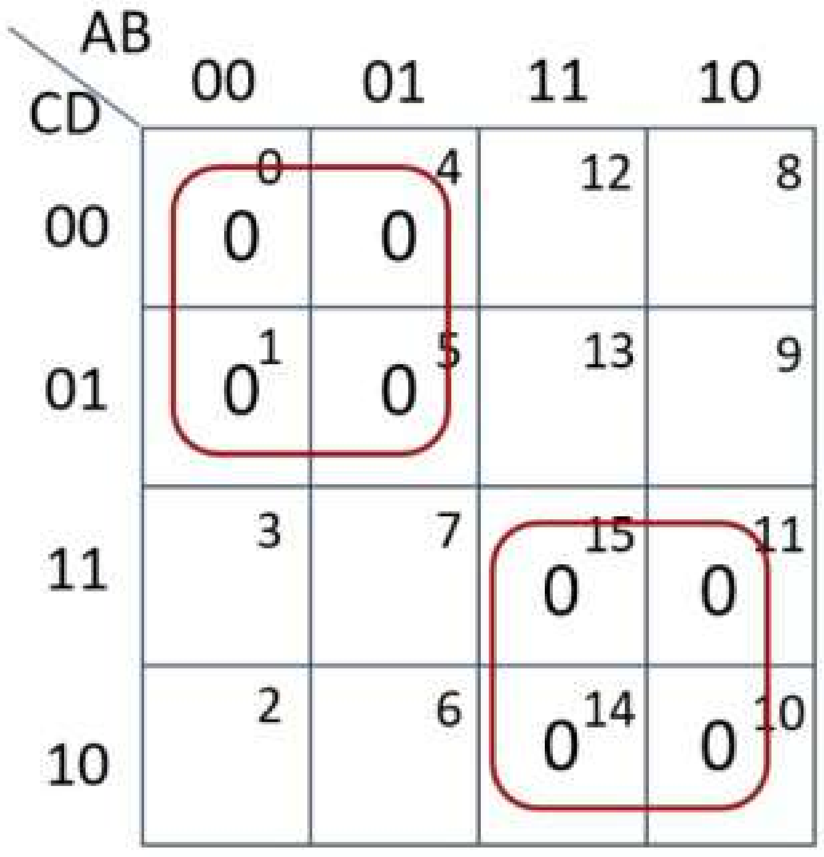


CD \ AB	00	01	11	10
00	0 1	4	12	8 1
01	1	5	13 1	9
11	3	7	15	11
10	2 1	6	14	10 1

Answer: f = B'D' + ABC'D

Example – 4 Reduce f(A, B, C, D) = $\prod_M(0, 1, 4, 5, 10, 11, 14, 15)$

- In POS form, we have to put "0" in the given number boxes in K-Map.
- Make group of Os same as we make group of 1s in SOP form.
- For common "1" write complemented form and for common "0" write uncomplemented form of variable, e.g. 1 - A' and 0 - A.



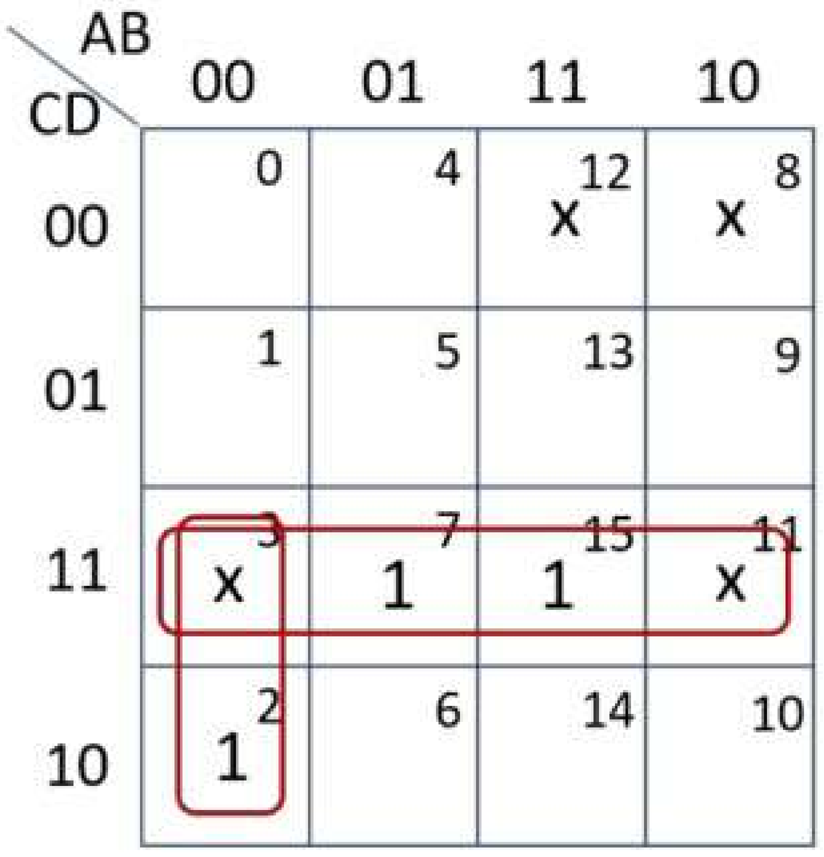
CD \ AB	00	01	11	10
00	0^{0}	0^{4}	12	8
01	0^{1}	0^{5}	13	9
11	3	7	0^{15}	0^{11}
10	2	6	0^{14}	0^{10}

Answer: f = (A+C) (A'+C')

K-Map with don't care condition

- Suppose we are given a problem of implementing a circuit to generate a logical 1 when a 2, 7, or 15 appears on a four-variable input.
- A logical 0 should be generated when 0, 1, 4, 5, 6, 9, 10, 13 or 14 appears.
- The input conditions for the numbers 3, 8, 11 and 12 never occur in the system. This means we don't care whether inputs generate logical 1 or logical 0.
- Don't care combinations are denoted by 'x' in K-Map which can be used for the making groups.
- The above example can be represented as

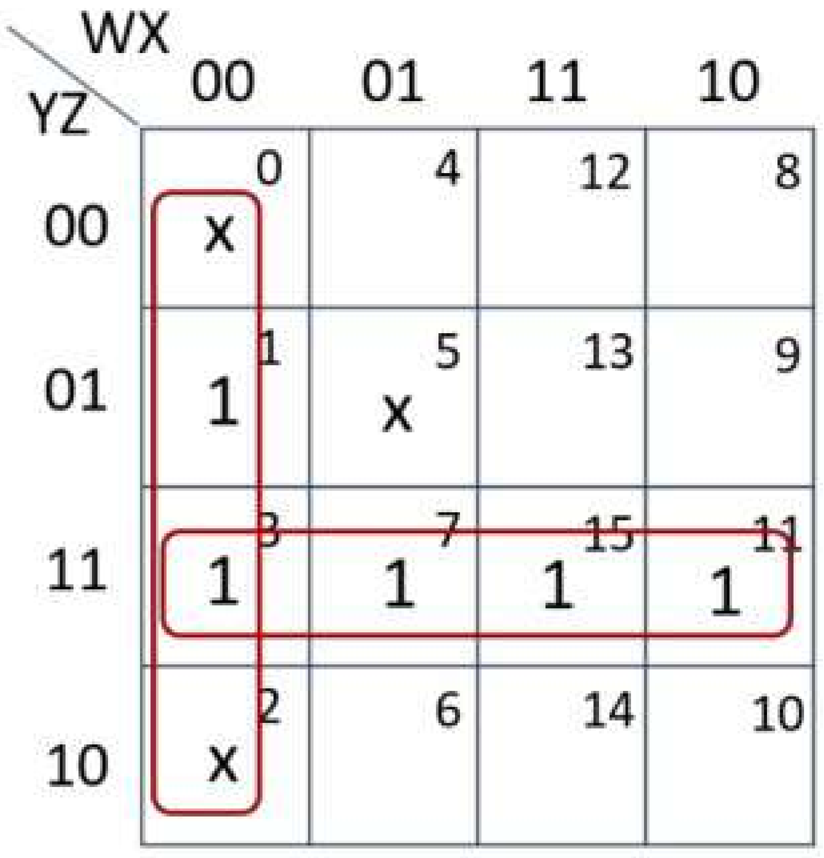
Example - 1 Reduce f(A, B, C, D) = $\sum_m(2, 7, 15) + d(3, 8, 11, 12)$



CD \ AB	00	01	11	10
00	0	4	x^{12}	x^{8}
01	1	5	13	9
11	x^{3}	1^{7}	1^{15}	x^{11}
10	1^{2}	6	14	10

Answer: f = CD + A'B'C

Example - 2 Reduce f(W, X, Y, Z) = $\sum_m(1,3,7,11,15) + d(0,2,5)$

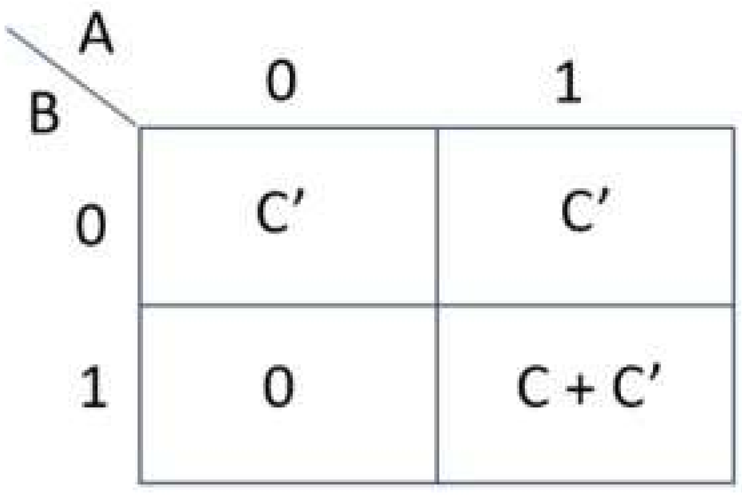


YZ \ WX	00	01	11	10
00	0 x	4	12	8
01	1 1	5 x	13	9
11	3 1	7 1	15 1	11 1
10	2 x	6	14	10

Answer: f = W’X’ + YZ

Variable-Entered Map (VEM)

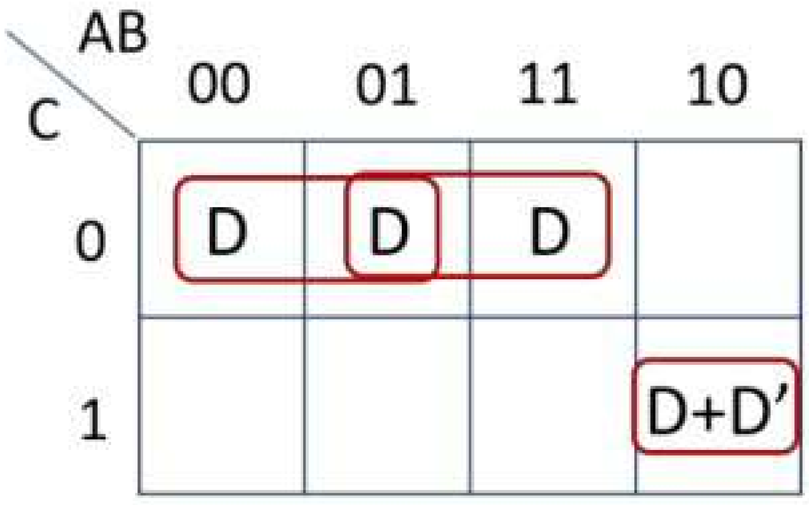
- Variable-entered map can be used to plot an n-variable problem on n - 1 variable map.
- Possible to reduce the map dimension by two or three in some cases.
- Advantage of using VEM occurs in design problems involving multiplexers.
- For example, Map-entered variable for 3 variable function
- Consider the function $X = A'B'C' + ABC' + AB'C' + ABC$
- Considering value of X to be function of map location and the variable C and plotting 2 variable K-Map.



B \ A	0	1
0	C’	C’
1	0	C + C’

Example - 1 Reduce f = AB’CD + A’BC’D + AB’CD’ + ABC’D + A’B’C’D

$$= \underset{5}{\underline{AB'C}}D + \underset{2}{\underline{A'BC'}}D + \underset{5}{\underline{AB'C}}D' + \underset{6}{\underline{ABC'}}D + \underset{0}{\underline{A'B'C'}}D$$

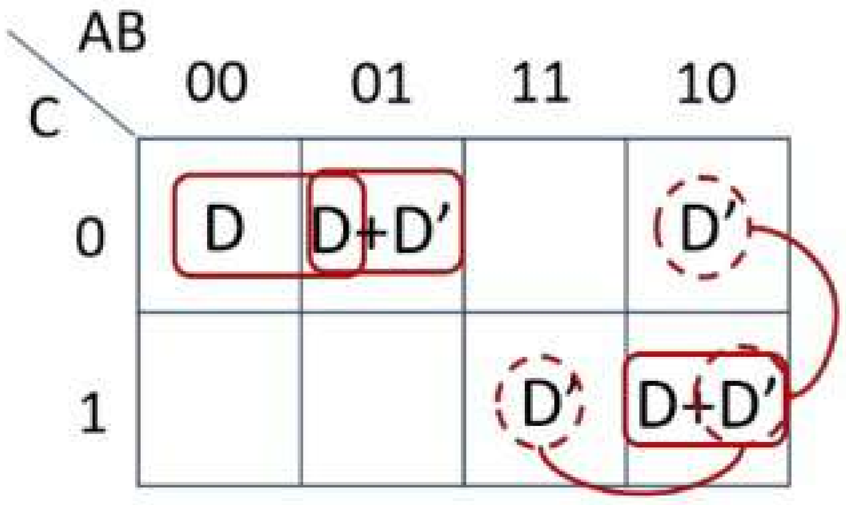


C \ AB	00	01	11	10
0	D	D	D	
1				D+D’

Answer: f = A’C’D + BC’D + AB’C

Example - 2 Reduce f = A'B'C'D + A'BC'D' + A'BC'D + AB'C'D' + AB'CD' + AB'CD + ABCD' using VEM method.

$$= \underset{0}{\underline{A'B'C'}}D + \underset{2}{\underline{A'BC'}}D' + \underset{2}{\underline{A'BC'}}D + \underset{4}{\underline{AB'C'}}D' + \underset{5}{\underline{AB'C}}D' + \underset{5}{\underline{AB'C}}D + \underset{7}{\underline{ABC}}D'$$

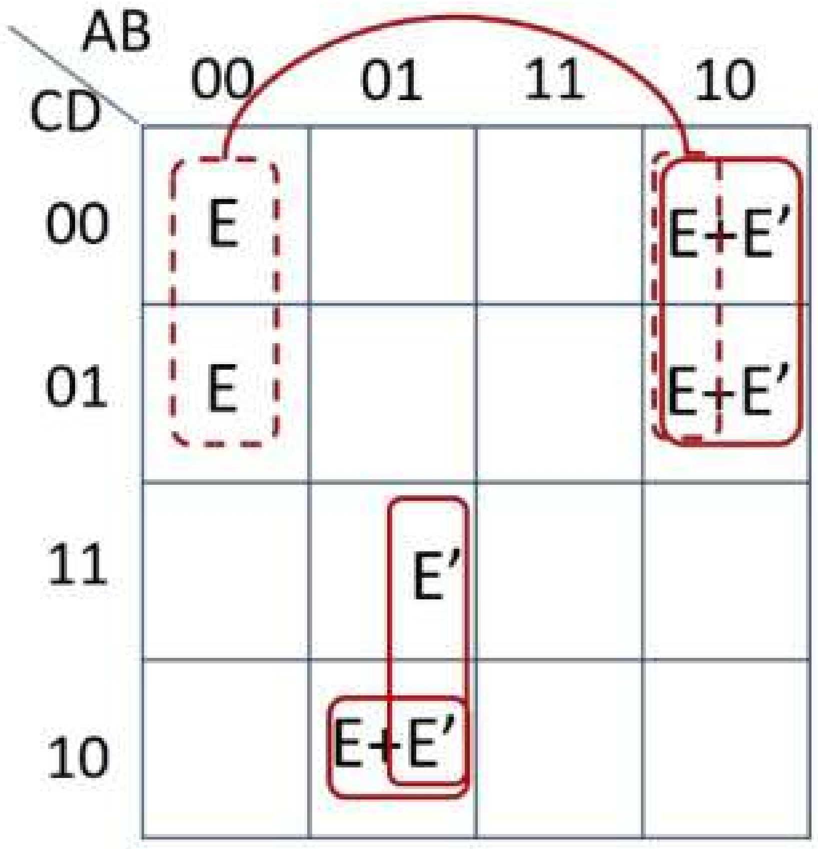


C \ AB	00	01	11	10
0	D	D+D'		D'
1			D'	D+D'

Answer: f = A'C'D + A'BC' + ACD' + AB'D' + AB'C

Example - 3 Reduce f = A'B'C'D'E + A'B'C'DE + A'BCD'E' + A'BCD'E + AB'C'D'E' + AB'C'D'E + AB'C'D + A'BCDE'

$$= \underset{0}{\underline{A'B'C'D'}}E + \underset{1}{\underline{A'B'C'D}}E + \underset{6}{\underline{A'BCD'}}E' + \underset{6}{\underline{A'BCD'}}E + \underset{8}{\underline{AB'C'D'}}E' + \underset{8}{\underline{AB'C'D'}}E + \underset{9}{\underline{AB'C'D}} + \underset{7}{\underline{A'BCD}}E'$$



CD \ AB	00	01	11	10
00	E			E+E'
01	E			E+E'
11		E'		
10		E+E'		

Answer: f = B'C'E + AB'C' + A'BCD' + A'BCE'

Quine McCluskey Method (Tabulation Method)

Procedure for minimization using tabulation method

1. List all the minterms.

2. Arrange all minterms in groups of the same number of 1s in their binary representation in column 1. Start with the least number of 1s group and continue with groups of increasing number of 1s.

3. Compare each term of the lowest index group with every term in the succeeding group. Whenever possible, combine the two terms being compared by means of the combining theorem. Two terms from adjacent groups are combinable, if their binary representations differ by just a single digit in the same position; the combined terms consist of the original fixed representation with the differing one replaced by a dash (-). Place a check mark (√) next to every term, which has been combined with at least one term and write the combined terms in column 2. Repeat this by comparing each term in a group of index i with every term in the group of index i + 1, until all possible applications of the combining theorem have been exhausted.

4. Compare the terms generated in step 3 in the same fashion; combine two terms which differ by only a single 1 and whose dashes are in the same position to generate a new term. Two terms with dashes in different positions cannot be combined. Write the new terms in column 3 and put a check mark next to each term which has been combined in column 2. Continue the process with terms in columns 3, 4 etc. until no further combinations are possible. The remaining *unchecked terms* constitute the *set of prime implicants* of the expression.

5. List all the prime implicants and draw the *prime implicant chart*. (The don't cares if any should not appear in the prime implicant chart).

6. Obtain the *essential prime implicants* and write the minimal expression.

Example - 1 Simplify f(A, B, C, D) = $\sum_m(1, 2, 3, 5, 6, 7, 8, 9, 12, 13, 15)$ using tabulation method.

Step:-1 List all minterm

Minterms	Binary Designation
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
12	1 1 0 0
13	1 1 0 1
15	1 1 1 1

Step:-2 Arrange all minterms in groups of same number of 1s

Column-1		
Index	Minterms	Binary Designation
Index 1	1	$\underline{0}\ \underline{0}\ \underline{0}\ 1$✓
	2	$0\ \underline{0}\ 1\ \underline{0}$✓
	8	$1\ \underline{0}\ 0\ \underline{0}$✓
Index 2	3	$0\ \underline{0}\ \underline{1}\ \underline{1}$✓
	5	$\underline{0}\ \underline{1}\ \underline{0}\ 1$✓
	6	$0\ \underline{1}\ 1\ \underline{0}$✓
	9	$\underline{1}\ \underline{0}\ 0\ \underline{1}$✓
	12	$1\ \underline{1}\ 0\ \underline{0}$✓
Index 3	7	$\underline{0}\ \underline{1}\ \underline{1}\ \underline{1}$✓
	13	$\underline{1}\ \underline{1}\ \underline{0}\ \underline{1}$✓
Index 4	15	$\underline{1}\ 1\ \underline{1}\ 1$✓

Step:-3 Compare each term of the lowest index group with every term in the succeeding group till no change.

Column-2	
Pairs	A B C D
1,3	0 0 – 1 ✓
1,5	0 – 0 1 ✓
1,9	– 0 0 1 ✓
2,3	0 0 1 – ✓
2,6	0 – 1 0 ✓
8,9	1 0 0 – ✓
8,12	1 – 0 0 ✓
3,7	0 – 1 1 ✓
5,7	0 1 – 1 ✓
5,13	– 1 0 1 ✓
6,7	0 1 1 – ✓
9,13	1 – 0 1 ✓
12,13	1 1 0 – ✓
7,15	– 1 1 1 ✓
13,15	1 1 – 1 ✓

Step:-4 Compare the terms generated in step 3 in the same fashion until no further combinations are possible.

Column-3	
Quads	A B C D
1,3,5,7	0 – – 1 **T**
1,5,9,13	– – 0 1 **S**
2,3,6,7	0 – 1 – **R**
8,9,12,13	1 – 0 – **Q**
5,7,13,15	– 1 – 1 **P**

Step:-5 List all prime implicants and draw prime implicants chart.
Prime Implicants: P(BD), Q(AC'), R(A'C), S(C'D) , T(A'D)

Minterms	1	2 ✓	3 ✓	5 ✓	6 ✓	7 ✓	8 ✓	9 ✓	12 ✓	13 ✓	15 ✓
T(A'D)	X		X	X		X					
S(C'D)	X			X				X		X	
R(A'C) ✓		$\boxed{\text{X}}$	X		X	X					
Q(AC') ✓							$\boxed{\text{X}}$	X	X	X	
P(BD) ✓				X		X				X	$\boxed{\text{X}}$

Step:-6 Obtain essential prime implicants and minimal expression.

Essential Prime Implicants: P(BD), Q(AC'), R(A'C)

Minimal Expression: P+Q+R+S = BD + A'C + AC' + C'D
P+Q+R+T = BD + A'C + AC' + A'D

As minterm 1 is covered by S and T.

Example - 2 Simplify f(A, B, C, D) = $\sum_m(0, 1, 3, 7, 8, 9, 11, 15)$ using tabulation method.

Step:-1 List all minterm

Minterms	Binary Designation
0	0 0 0 1
1	0 0 1 0
3	0 0 1 1
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
11	1 0 1 1
15	1 1 1 1

Step:-2 Arrange all minterms in groups of same number of 1s

Column-1		
Index	**Minterms**	**Binary Designation**
Index 0	0	$\underline{0}\ 0\ 0\ \underline{0}$ ✓
Index 1	1 8	$\underline{0}\ 0\ \underline{0}\ \underline{1}$ ✓ $\underline{1}\ 0\ 0\ \underline{0}$ ✓
Index 2	3 9	$\underline{0}\ \underline{0}\ \underline{1}\ 1$ ✓ $\underline{1}\ 0\ \underline{0}\ \underline{1}$ ✓
Index 3	7 11	$\underline{0}\ \underline{1}\ 1\ 1$ ✓ $\underline{1}\ \underline{0}\ \underline{1}\ 1$ ✓
Index 4	15	$\underline{1}\ \underline{1}\ 1\ 1$ ✓

Step:-3 Compare each term of the lowest index group with every term in the succeeding group till no change.

Column-2	
Pairs	**A B C D**
0,1 0,8	0 0 0 – ✓ – 0 0 0 ✓
1,3 1,9 8,9	0 0 – 1 ✓ – 0 0 1 ✓ 1 0 0 – ✓
3,7 3,11 9,11	0 – 1 1 ✓ – 0 1 1✓ 1 0 – 1 ✓
7,15 11,15	– 1 1 1 ✓ 1 – 1 1 ✓

Step:-4 Compare the terms generated in step 3 in the same fashion until no further combinations are possible.

Column-3	
Quads	**A B C D**
0,1,8,9 1,3,9,11	– 0 0 – **R** – 0 – 1 **Q**
3,7,11,15	– – 1 1 **P**

Step:-5 List all prime implicants and draw prime implicants chart.
Prime Implicants: P(CD), Q(B'D), R(B'C')

Minterms	0 ✓	1 ✓	3 ✓	7 ✓	8 ✓	9 ✓	11 ✓	15 ✓
P(CD)✓			X	X			X	X
Q(B'D)		X	X			X	X	
R(B'C')✓	X	X			X	X		

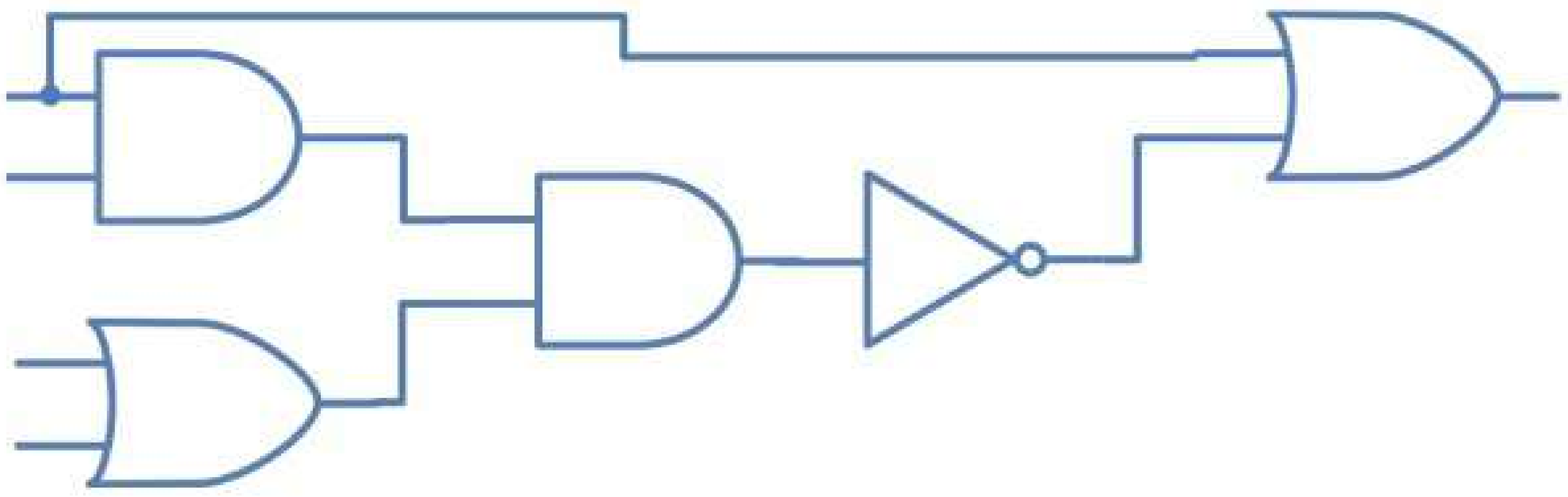
Step:-6 Obtain essential prime implicants and minimal expression.
Essential Prime Implicants: P(CD), R(B'C')

Minimal Expression: P+R = CD + B'C'

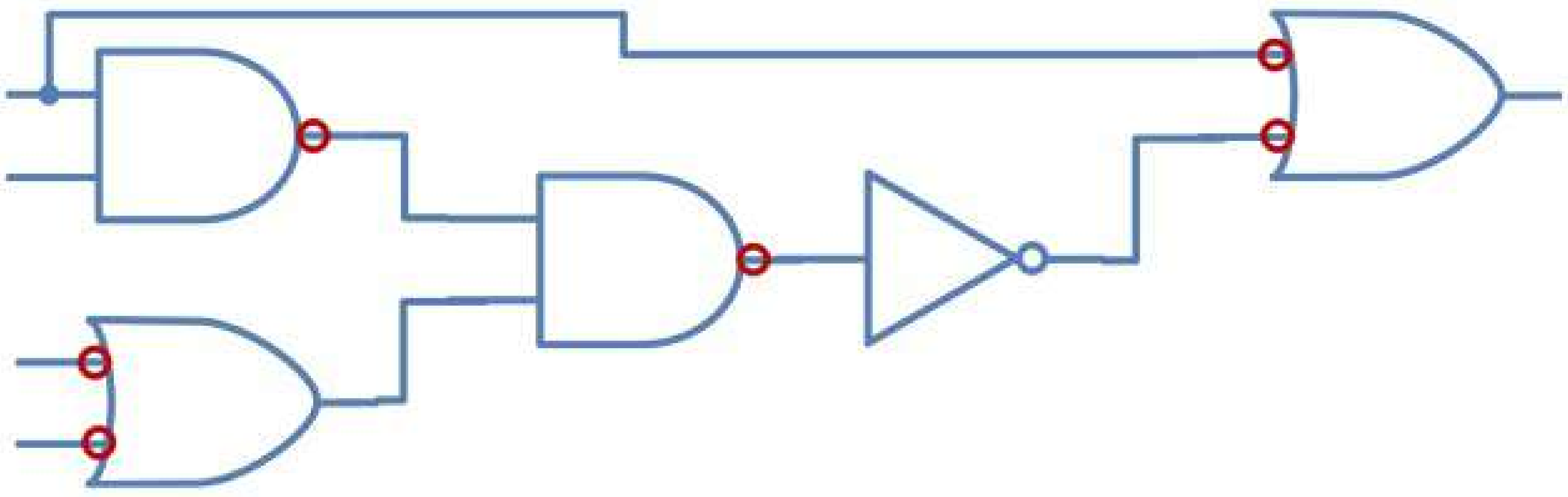
Realization using universal gates

1. Draw the circuit in AOI logic.
2. If NAND hardware is chosen, add a circle at the output of each AND gate and at the inputs to all the OR gates.
3. If NOR hardware is chosen, add a circle at the output of each OR gate and at the inputs to all the AND gates.
4. Add or subtract an inverter on each line that received a circle in steps 2 or 3 so that the polarity of signals on those lines remains unchanged from that of the original diagram.
5. Replace bubbled OR by NAND and bubbled AND by NOR.
6. Eliminate double inversions.

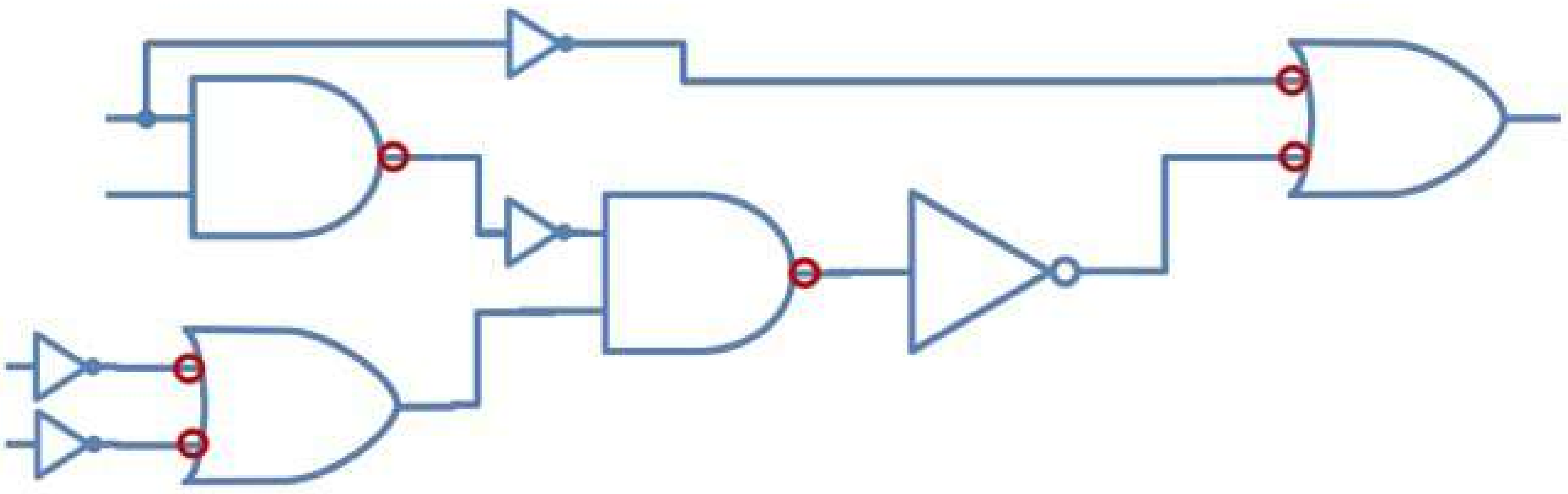
Example - 1 Implement the following AOI logic using NAND.



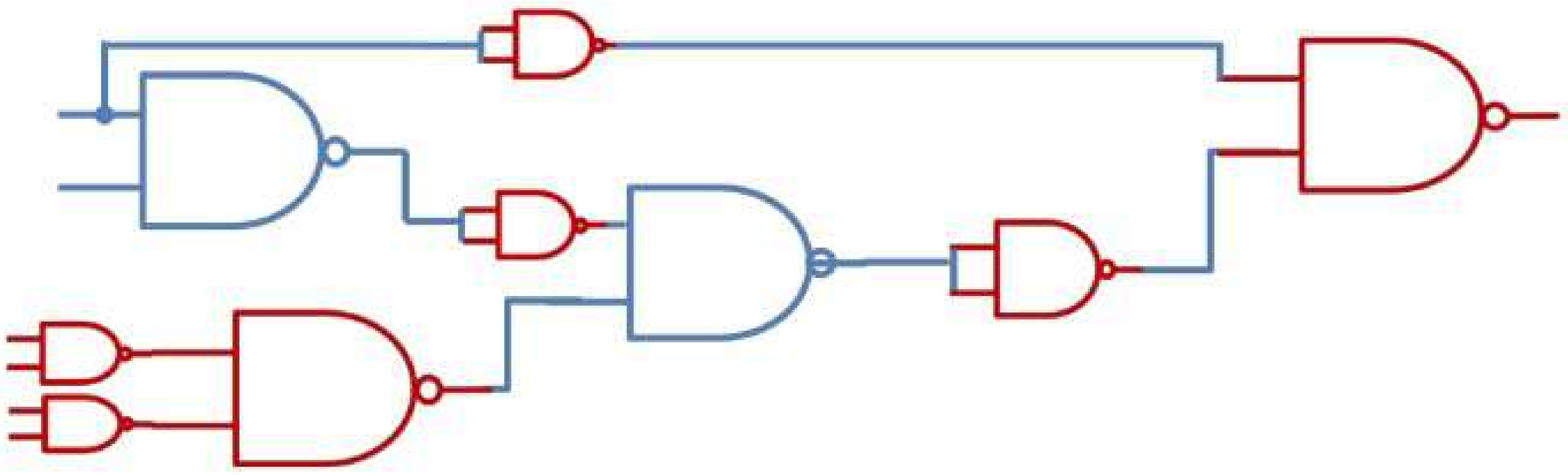
- Put a circle at the output of each AND gate and at the inputs to all OR gates



- Add an inverter to each of the lines that received only one circle at input so that polarity remains unchanged.



- Replace bubbled OR gates and NOT gates by NAND gates.

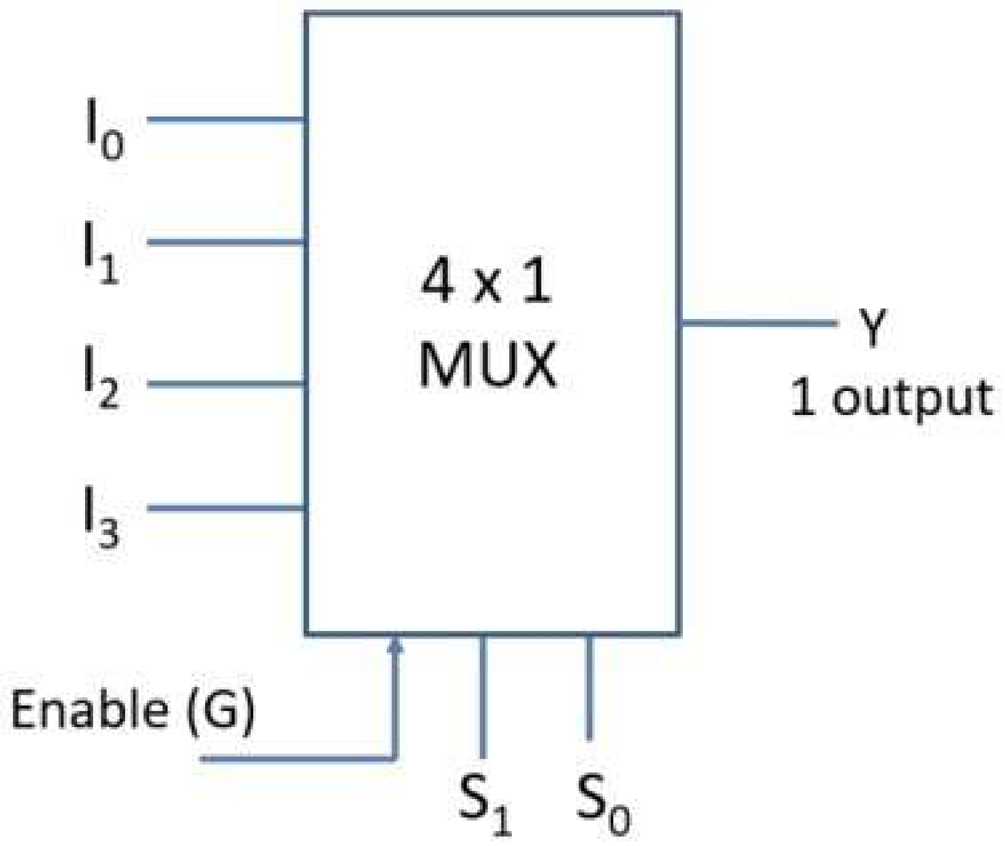


Multiplexer

- A multiplexer (MUX) is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination.
- Consider an integer 'm', which is constrained by the following relation:

  m = 2^n, where m and n are both integers.
- A **m-to-1** Multiplexer has
  - m Inputs: I_0, I_1, I_2, ... $I_{(m-1)}$
  - One Output: Y
  - n Control inputs: S_0, S_1, S_2, ... $S_{(n-1)}$
  - One (or more) Enable input(s)

  such that Y may be equal to one of the inputs, depending upon the control inputs.
- The block diagram of 4 x 1 multiplexer is as follows.



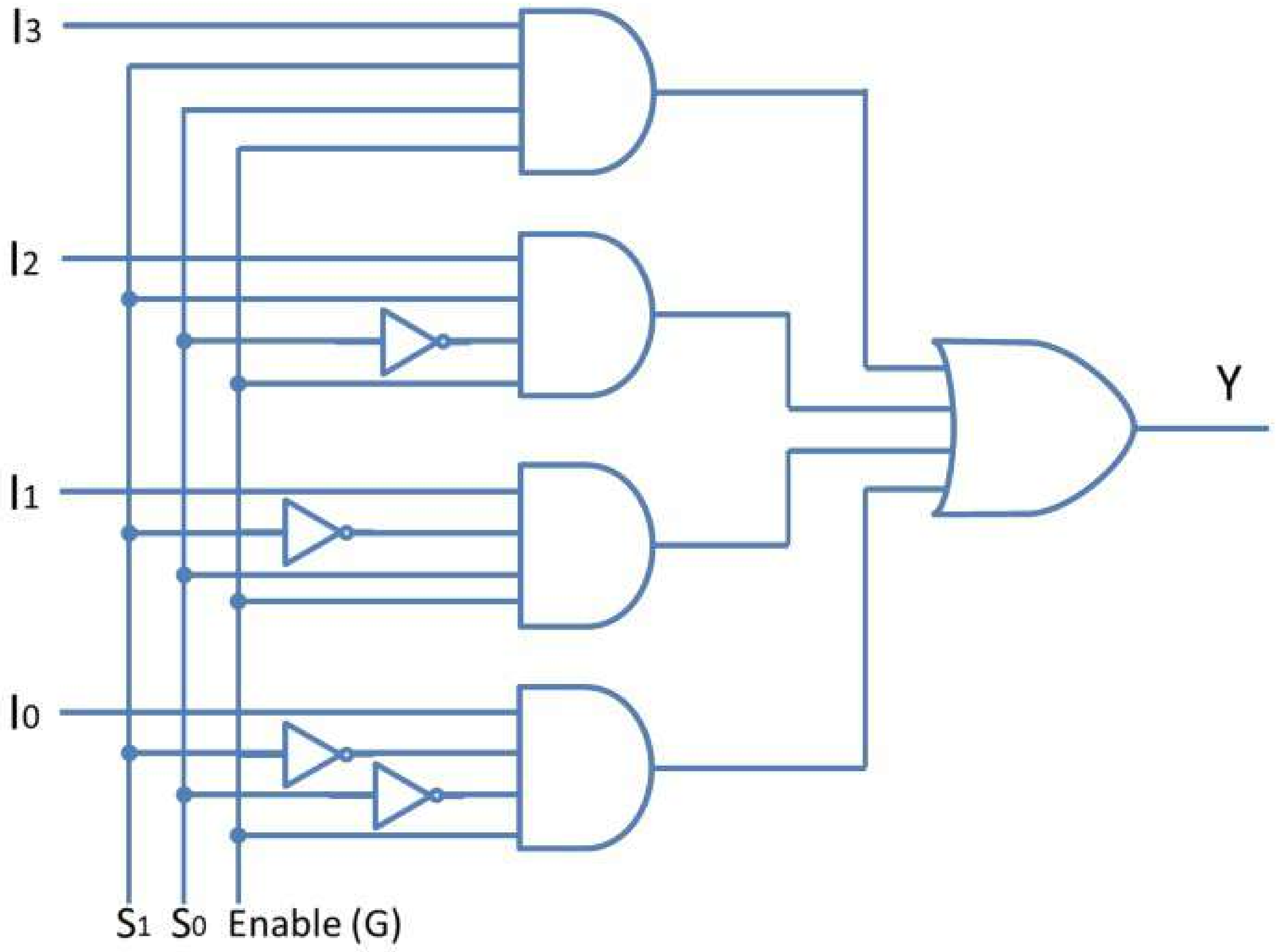
- The function table for the 4 x 1 multiplexer can be stated as below.

Select Inputs		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

- The following logic function describes the above function table.

$$\boxed{Y = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3}$$

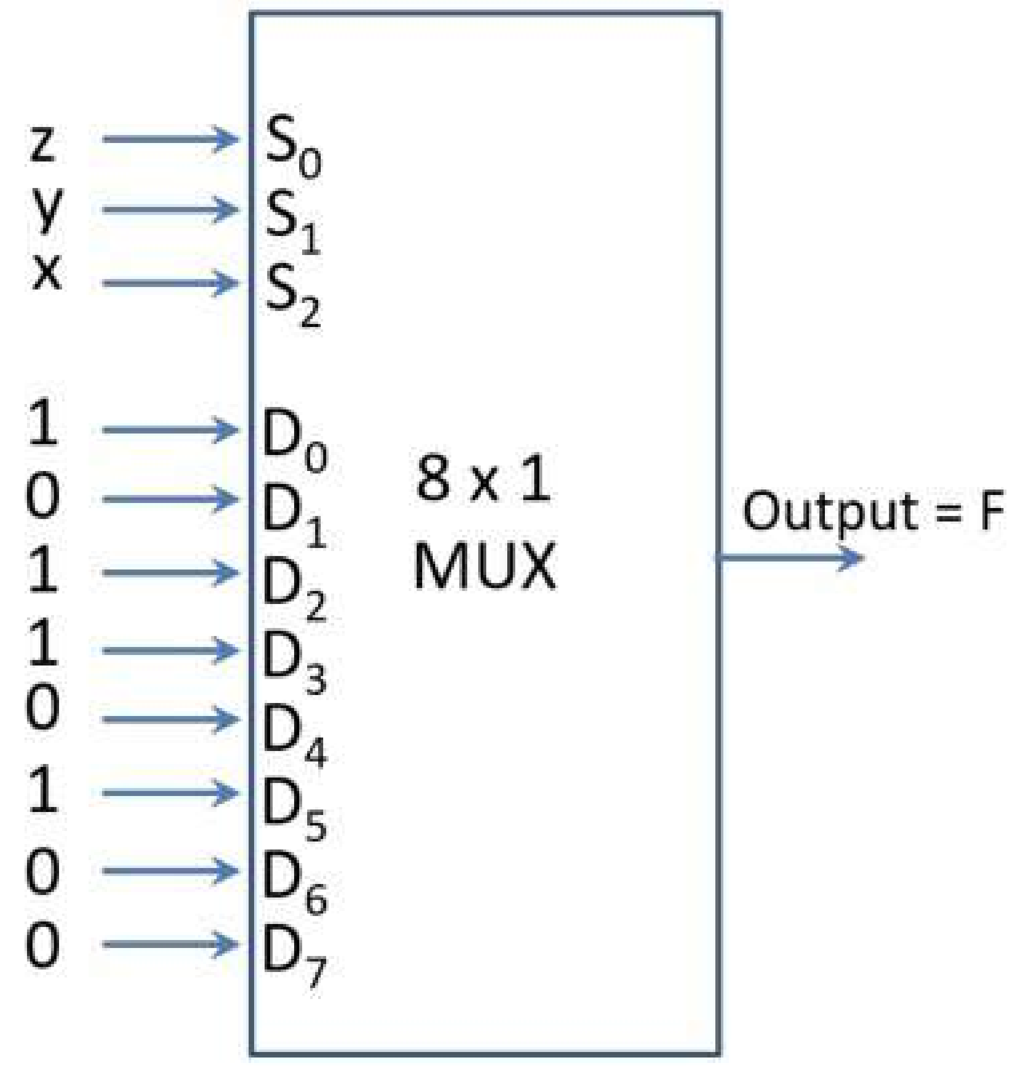
- The following figure describes the logic circuit for 4 x 1 multiplexer.



- Applications of Multiplexer is as follows:
    1. Logic function generation
    2. Data selection
    3. Data routing
    4. Operation sequencing
    5. Parallel-to-serial conversion
    6. Waveform generation

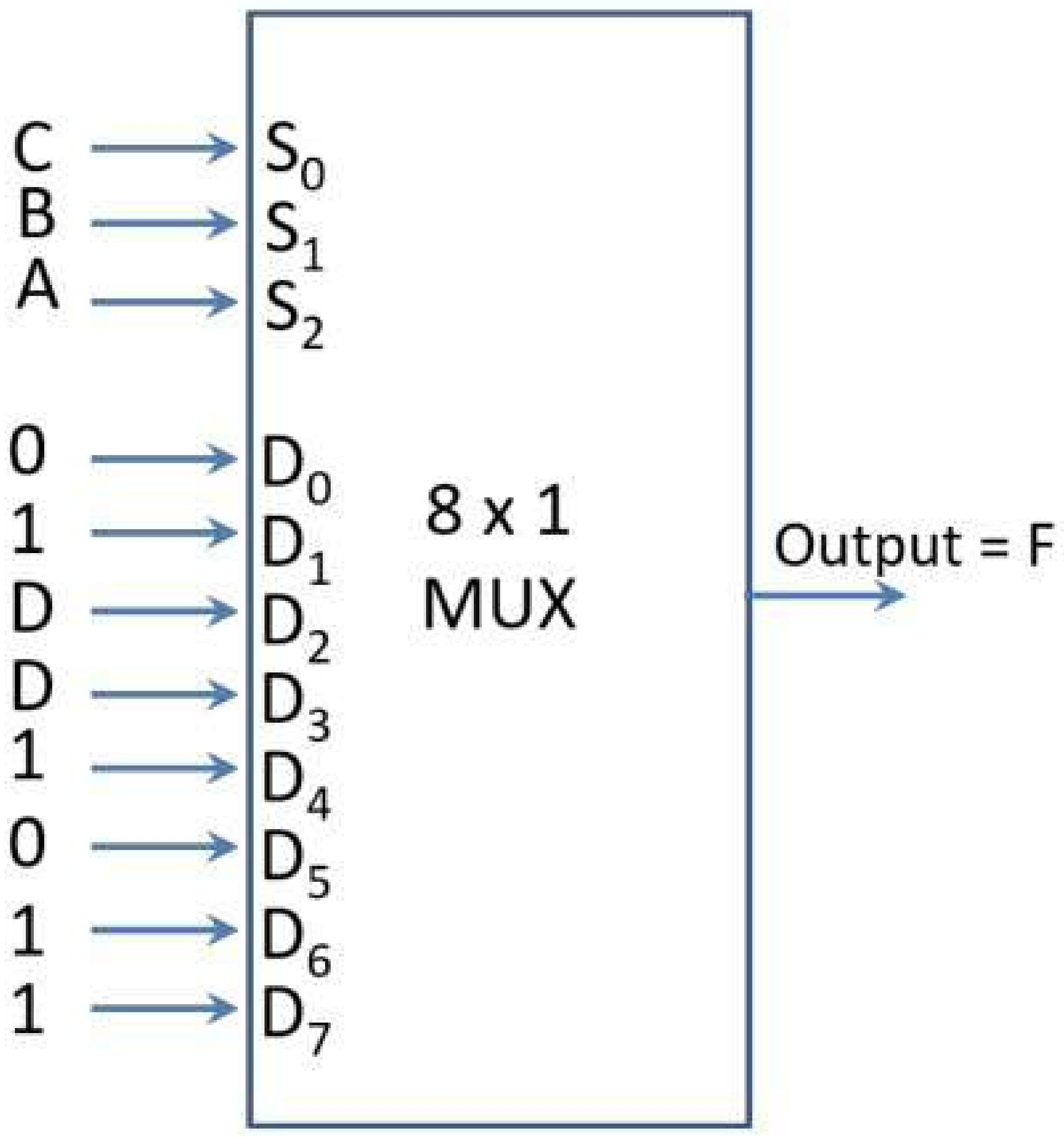
Example - 1 Implement the following function using 8 to 1 mux: f(X, Y, Z) = $\sum_m(0,2,3,5)$

S_2	S_1	S_0	F
x	y	z	
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



Example - 2 Implement the following using 8 to 1 mux: f(A, B, C, D) = $\sum_m(2,3,5,7,8,9,12,13,14,15)$

S_2	S_1	S_0	D	F	
A	B	C			
0	0	0	0	0	F = 0
0	0	0	1	0	
0	0	1	0	1	F = 1
0	0	1	1	1	
0	1	0	0	0	F = D
0	1	0	1	1	
0	1	1	0	0	F = D
0	1	1	1	1	
1	0	0	0	1	F = 1
1	0	0	1	1	
1	0	1	0	0	F = 0
1	0	1	1	0	
1	1	0	0	1	F = 1
1	1	0	1	1	
1	1	1	0	1	F = 1
1	1	1	1	1	



Half Adder

- A half-adder is a combinational circuit with two binary inputs (augend and addend bits) and two binary outputs (sum and carry bits).
- It adds the two inputs (single bit words A and B) and produces the sum (S) and the carry (C) bits.
- The truth table of a half-adder are shown below:

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

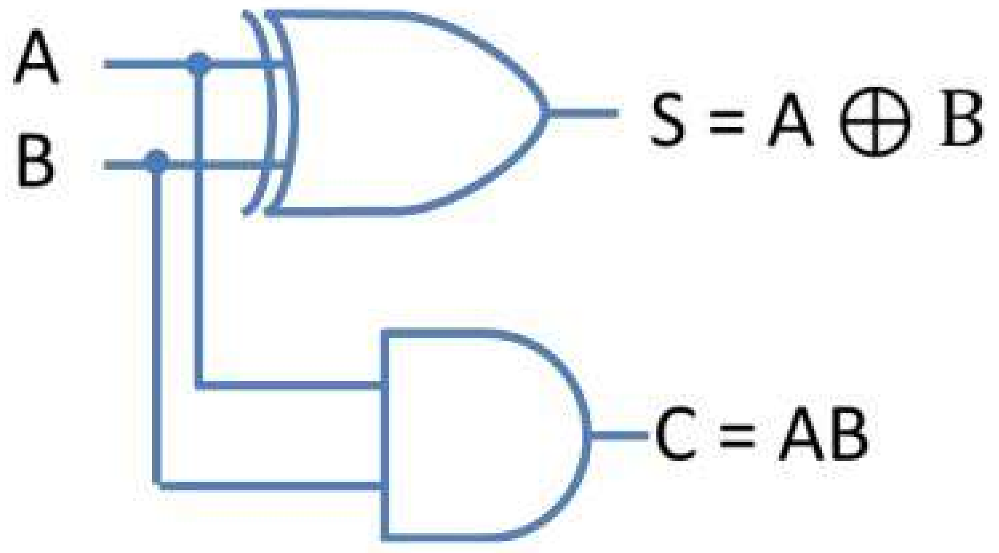
- The Sum (S) is the X-OR of A and B (It represent the LSB of the sum). Therefore,

$$S = AB' + BA' = A \oplus B$$

- The carry (C) is the AND of A and B (It is 0 unless both the inputs are 1). Therefore,

$$C = AB$$

- A half-adder can, therefore, be realized by using one X-OR gate and one AND gate as shown in figure below.



Full Adder

- A full-adder is a combinational circuit that adds two bits and a carry and outputs a sum bit and a carry bit.
- When we want to add two binary numbers, each having two or more bits, the LSBs can be added by using a half-adder.
- The carry resulted from the addition of the LSBs is carried over to the next significant column and added to the two bits in that column.
- The full-adder adds the bits A and B and the carry from the previous column called the carry-in C_{in} and outputs the sum bit S and the carry bit called the carry-out C_{out}.
- The variable S gives the value of the least significant bit of the sum.
- The variable C_{out} gives the output carry.
- The truth table of a full-adder are shown in figure below.

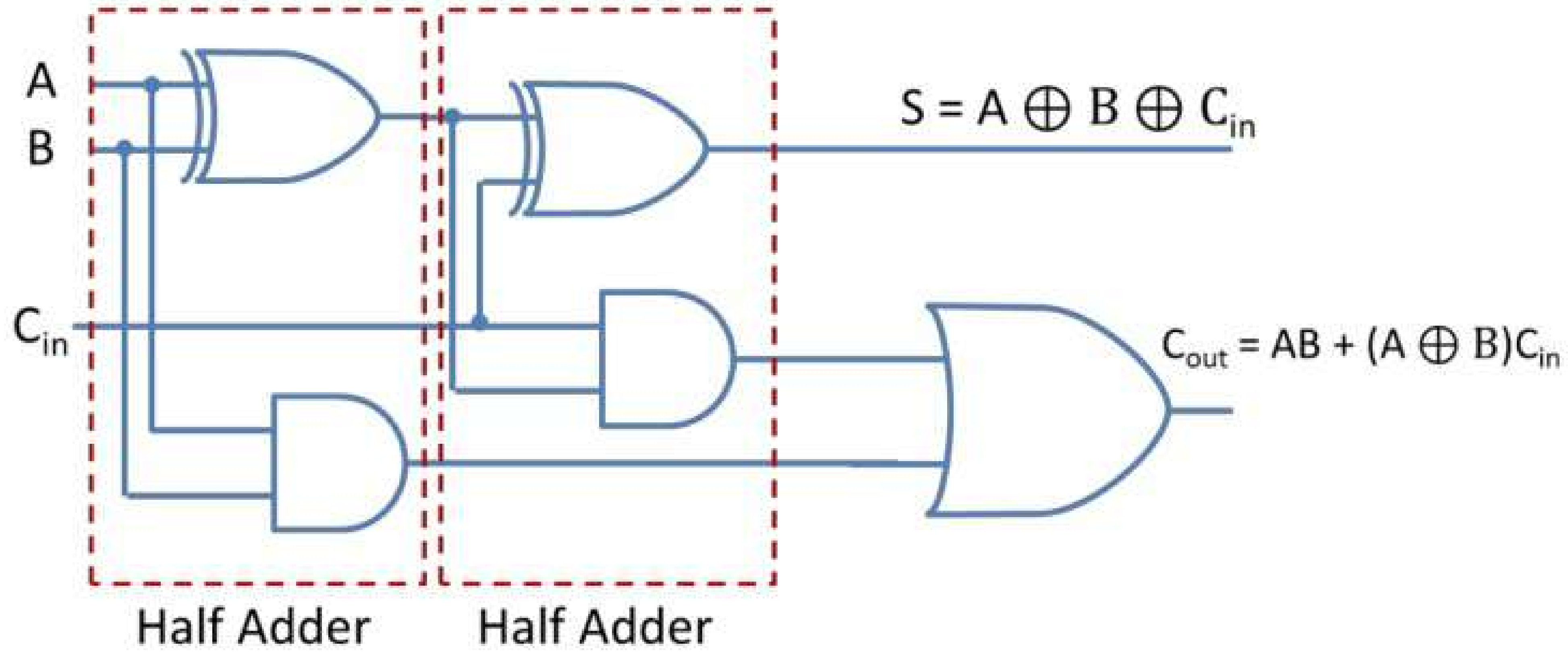
Inputs			Outputs	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- The eight rows under the input variables designate all possible combinations of 1s and 0s that these variables may have.
- When all the bits are 0s, the output is 0.
- The S output is equal to 1 when only 1 input is equal to 1 or when all the inputs are equal to 1.
- The C_{out} has a carry of 1 if two or three inputs are equal to 1.
- From the truth table, a circuit that will produce the correct sum and carry bits in response to every possible combination of A, B, and C_{in} is described by

$$\begin{aligned} S &= A'B'C_{in} + A'BC_{in}' + AB'C_{in}' + ABC_{in} \\ &= (AB' + A'B)C_{in}' + (AB + A'B')C_{in} \\ &= (A \oplus B)C_{in}' + (A \oplus B)'C_{in} \\ &= A \oplus B \oplus C_{in} \end{aligned}$$

$$\begin{aligned} C_{out} &= A'BC_{in} + AB'C_{in} + ABC_{in}' + ABC_{in} \\ &= AB + (A \oplus B)C_{in} \end{aligned}$$

- The sum term of the full-adder is the X-OR of A, B and C_{in}, i.e., the sum bit is the modulo sum of the data bits in that column and the carry from the previous column.
- The logic diagram of the full-adder using two X-OR gates and two AND gates (i.e., two half-adders) and one OR gate is shown in figure below.


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

Half Subtractor

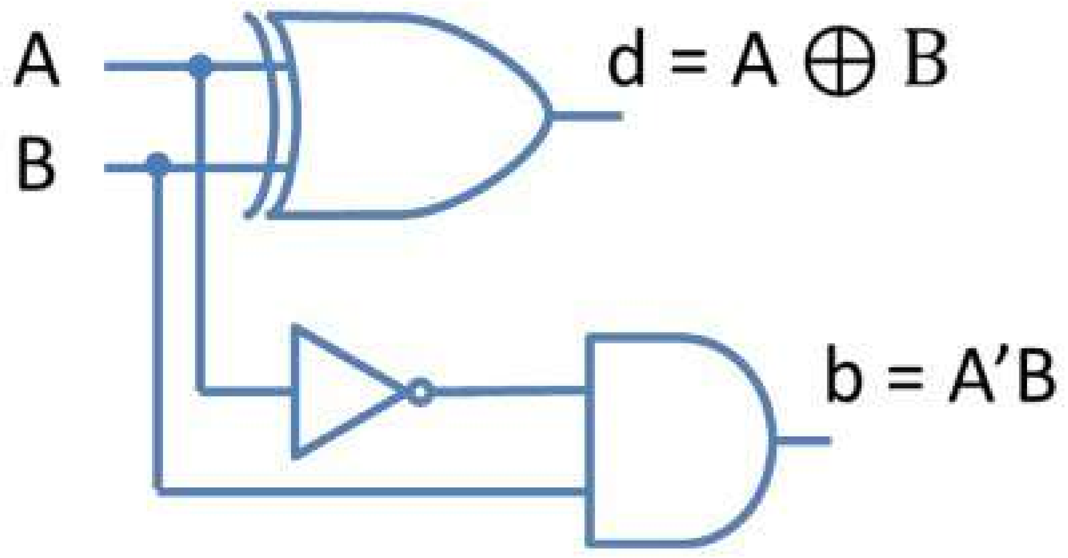
- A half-subtractor is a combinational circuit that subtracts one bit from the other and produces the difference.
- It also has an output to specify if a 1 has been borrowed.
- It is used to subtract the LSB of the subtrahend from the LSB of the minuend when one binary number is subtracted from the other.
- A half-subtractor is a combinational circuit with two inputs A and B and two outputs *d* and *b*.
- *d* indicates the difference and *b* is the output signal generated that informs the next stage that a 1 has been borrowed.
- We know that, when a bit B is subtracted from another bit A, a difference bit (d) and a borrow bit (b) result according to the rules given as follows.

Inputs		Outputs	
A	B	d	b
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

- A circuit that produces the correct difference and borrow bits in response to every possible combination of the two 1-bit numbers is, therefore, described by

$$d = AB' + BA' = A \oplus B \text{ and } d = A'B$$

- That is, the difference bit is obtained by X-ORing the two inputs, and the borrow bit is obtained by ANDing the complement of the minuend with the subtrahend.
- Figure below shows logic diagrams of a half-subtractor.



Full Subtractor

- The half-subtractor can be used only for LSB subtraction.
- If there is a borrow during the subtraction of the LSBs, it affects the subtraction in the next higher column; the subtrahend bit is subtracted from the minuend bit, considering the borrow from that column used for the subtraction in the preceding column.
- Such a subtraction is performed by a full-subtractor.
- It subtracts one bit (B) from another bit (A), when already there is a borrow b_i from this column for the subtraction in the preceding column, and outputs the difference bit (d) and the borrow bit (b) required from the next column.

- So a full-subtractor is a combinational circuit with three inputs (A, B, bi) and two outputs d and b.
- The 1s and 0s for the output variables are determined from the subtraction of A - B - b_i.
- The truth table of a full-subtractor are shown in figure.

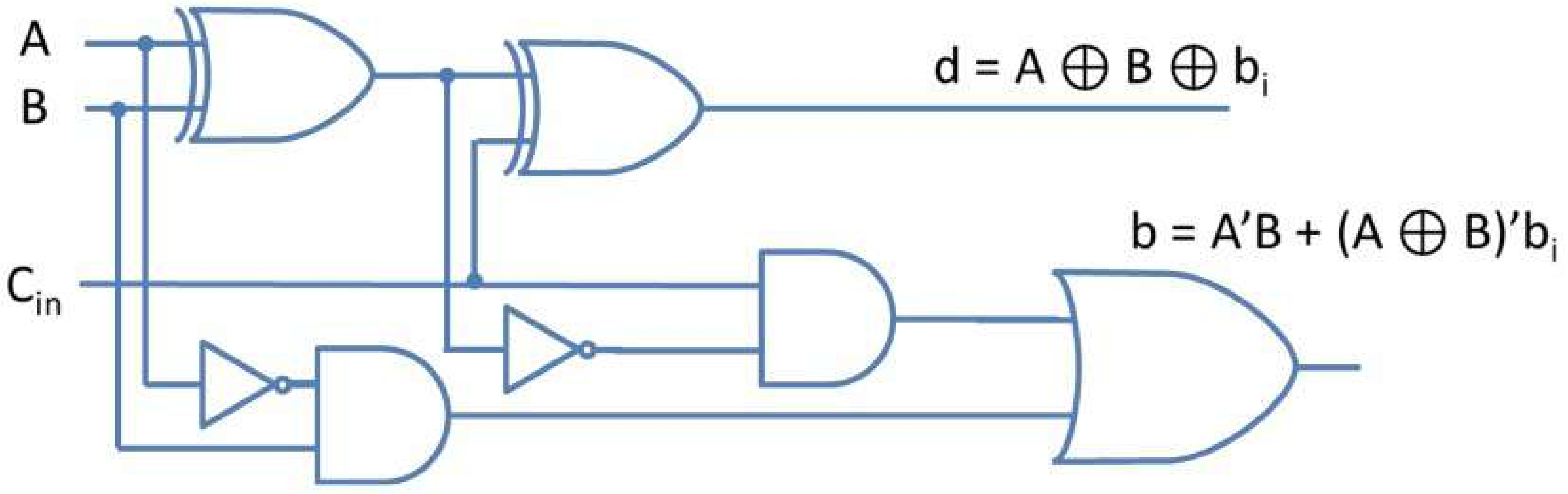
Inputs			Outputs	
A	**B**	**b_i**	**d**	**b**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

- From the truth table, a circuit that will produce the correct difference and borrow bits in response to every possible combination of A, B, and b_i is described by

$$\begin{aligned} d &= A'B'b_i + A'Bb_i' + AB'b_i' + ABb_i \\ &= (AB' + A'B)b_i' + (AB + A'B')b_i \\ &= (A \oplus B)b_i' + (A \oplus B)'b_i \\ &= A \oplus B \oplus b_i \end{aligned}$$

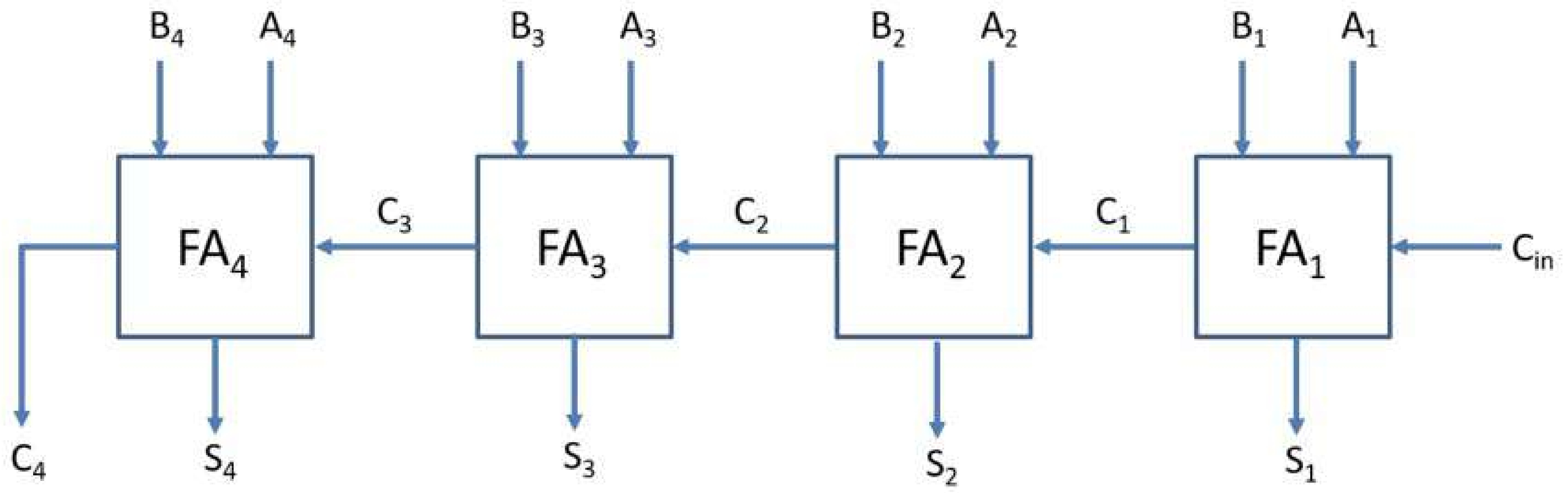
$$\begin{aligned} b &= A'B'b_i + A'Bb_i' + A'Bb_i + ABb_i \\ &= A'B(b_i + b_i') + (AB + A'B')b_i \\ &= A'B + (A \oplus B)'b_i \end{aligned}$$

- A full-subtractor can, therefore, be realized using X-OR gates as shown below.



Binary parallel adder

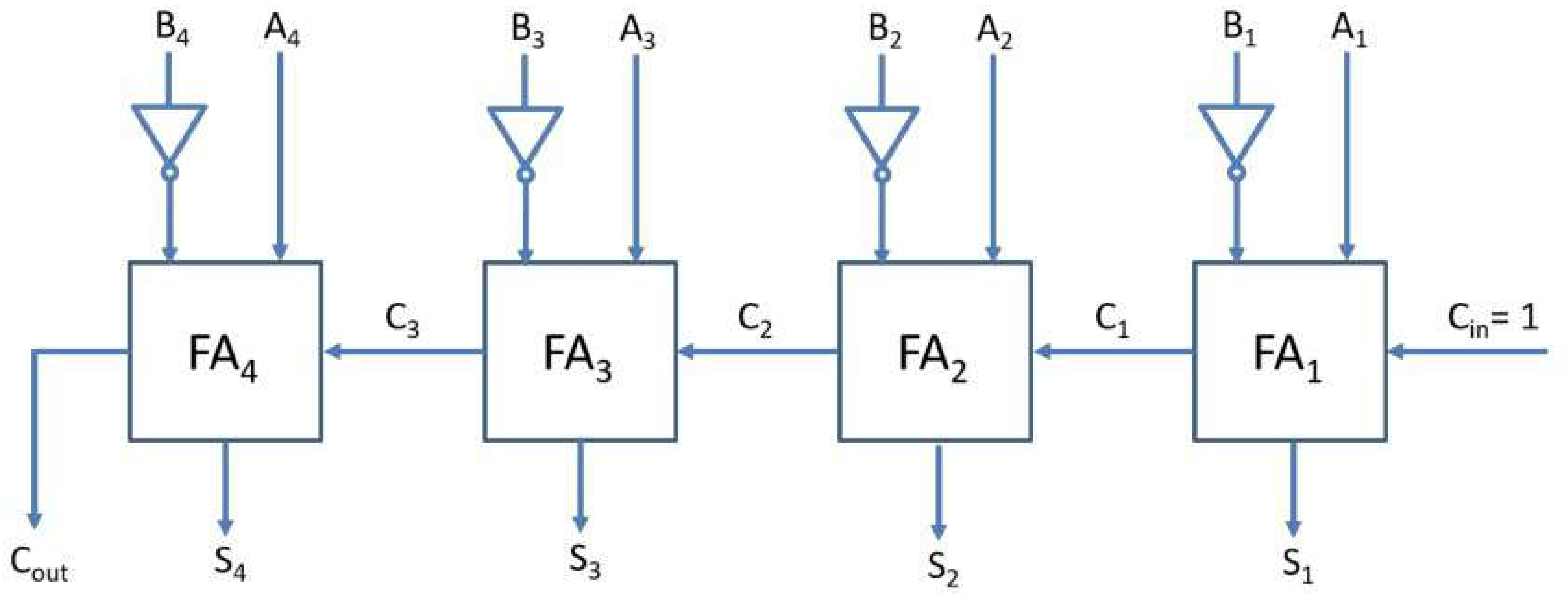
- A binary parallel adder is a digital circuit that adds two binary numbers in parallel form and produces the arithmetic sum of those numbers in parallel form.
- It consists of full adders connected in a chain with the output carry from each full-adder connected to the input carry of the next full-adder in the chain.



- Figure shows the interconnection of four full-adder (FA) circuits to provide a 4-bit parallel adder.
- The augend bits of A and addend bits of B are designated by subscript numbers from right to left, with subscript 1 denoting the lower-order bit.
- The carries are connected in 3 chain through the full-adders.
- The input carry to the adder is C_{in} and the output carry is C_4.
- The S outputs generate the required sum bits.
- When the 4-bit full-adder circuit is enclosed within an IC package, it has four terminals for the augend bits, four terminals for the addend bits, four terminals for the sum bits, and two terminals for the input and output carries.
- An n-bit parallel adder requires n-full adders.
- It can be constructed from 4-bit, 2-bit, and 1-bit full adder ICs by cascading several packages.
- The output carry from one package must be connected to the input carry of the one with the next higher-order bits.
- The 4-bit full adder is a typical example of an MSI function.

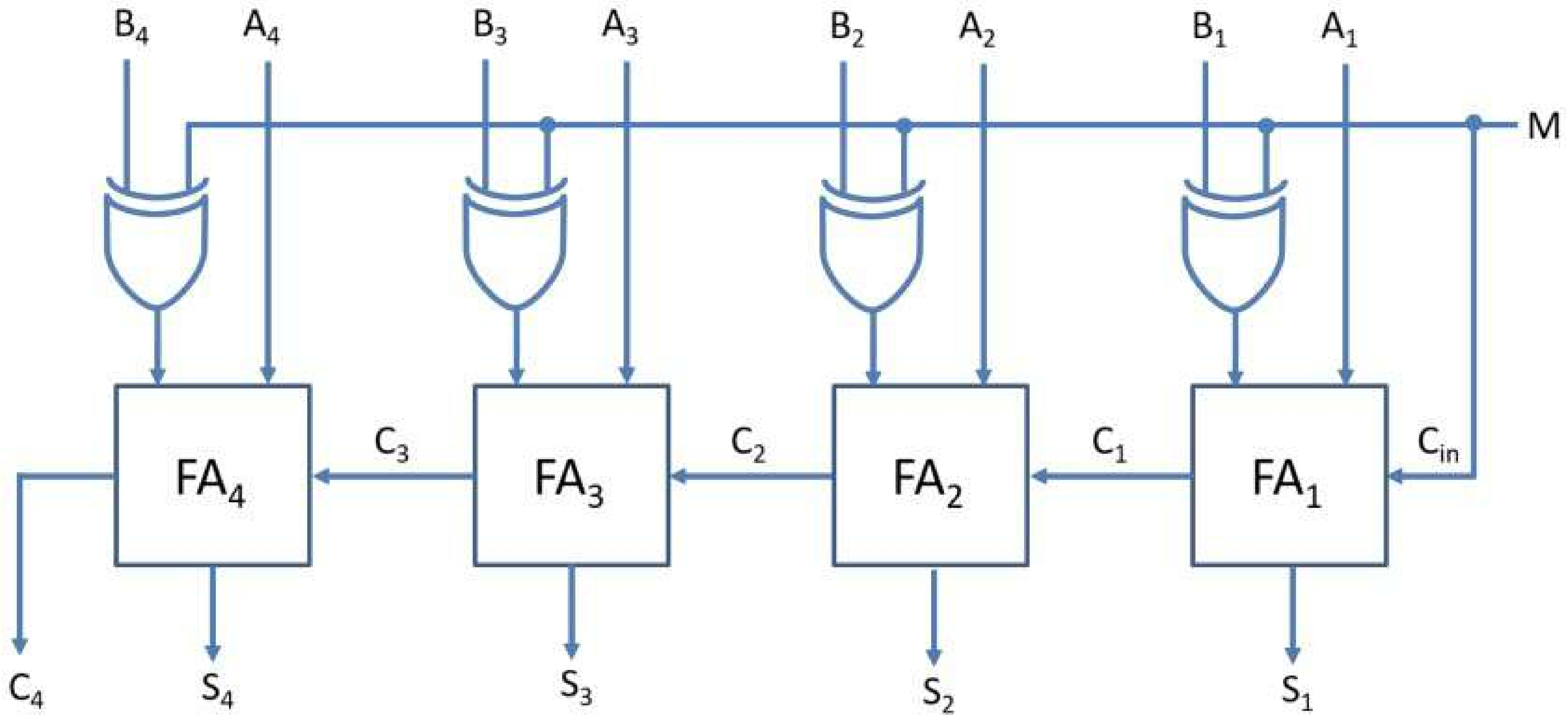
Binary parallel subtractor

- The subtraction of binary numbers can be carried out most conveniently by means of complement.
- 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 as shown in below figure.



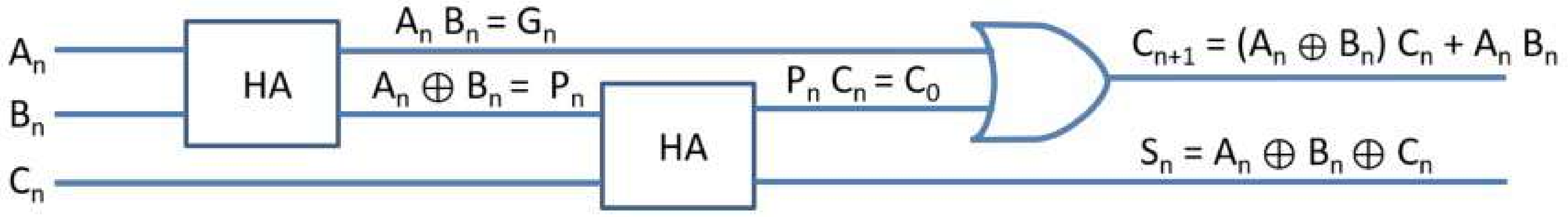
Binary adder subtractor

- Figure shows a 4-bit adder-subtractor circuit.
- Here the addition and subtraction operations are combined into one circuit with one common binary adder.
- This is done by including an X-OR gate with each full-adder.
- The mode input M controls the operation.
- When M = 0, the circuit is an adder, and when M = 1, the circuit becomes a subtractor.
- Each X-OR gate receives input M and one of the inputs of B.
- When M = 0, we have B $\oplus$ 0 = B. The full-adder receives the value of B, the input carry is 0 and the circuit performs A + B.
- When M = 1, we have B $\oplus$ 1 = B' and C_1 = 1. The B inputs are complemented and a 1 is added through the input carry.
- The circuit performs the operation A + B' + 1 (i.e. A – B).



Look ahead carry adder

- In the case of the parallel adder, the speed with which an addition can be performed is governed by the time required for the carries to propagate or ripple through all of the stages of the adder.
- The look-ahead-carry adder speeds up the process by eliminating this ripple carry delay.
- It examines all the input bits simultaneously and also generates the carry-in bits for all the stages simultaneously.
- The method of speeding up the addition process is based on the two additional functions of the full-adder, called the carry generate and carry propagate functions.
- Consider one full adder stage; say the nth stage of a parallel adder shown in Figure.
- We know that it is made of two half-adders and that the half-adder contains an X-OR gate to produce the sum and an AND gate to produce the carry.
- If both the bits A_n and B_n are 1s, a carry has to be generated in this stage regardless of whether the input carry C_{in} is a 0 or a 1.
- This is called generated carry, expressed as $G_n = A_n \cdot B_n$ which has to appear at the output through the OR gate as shown in Figure.


A_n
B_n
C_n
HA
$A_n B_n = G_n$
$A_n \oplus B_n = P_n$
HA
$P_n C_n = C_0$
$C_{n+1} = (A_n \oplus B_n) C_n + A_n B_n$
$S_n = A_n \oplus B_n \oplus C_n$

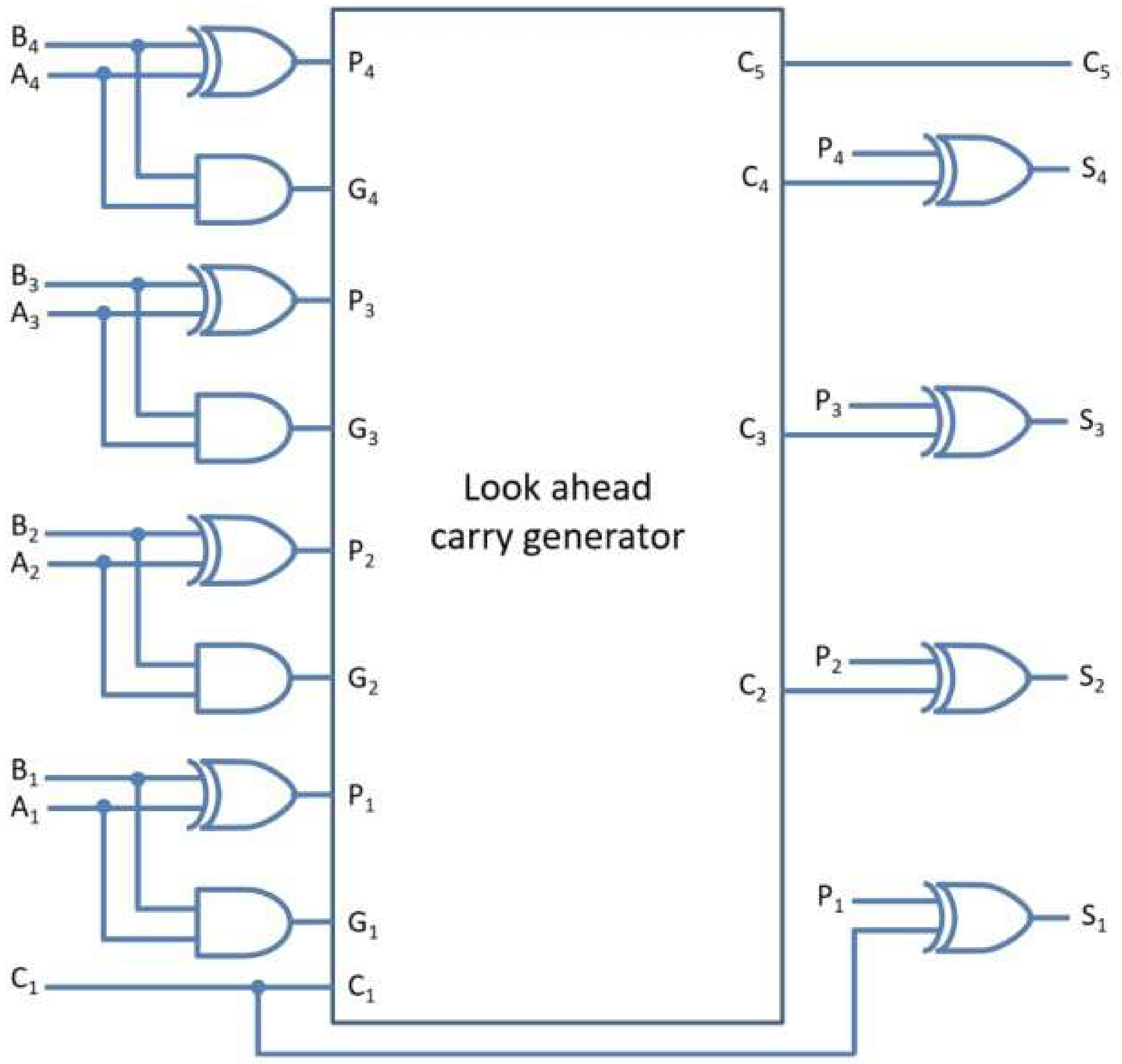
- There is another possibility of producing a carry out. X-OR gate inside the half-adder at the input produces an intermediary sum bit - call it P_n - which is expressed as $P_n = A_n \oplus B_n$.
- Next P_n and C_n, are added using the X-OR gate inside the second half adder to produce the final sum bit $S_n = P_n \oplus C_n = A_n \oplus B_n \oplus C_n$ and output carry $C_O = P_n \cdot C_n = (A_n \oplus B_n) \cdot C_n$ which becomes input carry for the (n+1)th stage.
- Consider the case of both P_n and C_n being 1. The input carry C_n has to be propagated to the output only if P_n is 1.
- If P_n is 0, even if C_n is 1, the AND gate in the second half-adder will inhibit C_n.
- We may thus call P_n as the propagated carry as this is associated with enabling propagation of C_n to the carry output of the nth stage which is denoted as C_{n+1} or C_{0n}.
- So, we can say that the carryout of the nth stage is 1 when either $G_n = 1$ or $P_n \cdot C_n = 1$ or both G_n and $P_n \cdot C_n$ are equal to 1.
- For the final sum and carry outputs of the nth stage, we get the following Boolean expressions.

$$S_n = P_n \oplus C_n \text{ where } P_n = A_n \oplus B_n$$
$$C_{on} = C_{n+1} = G_n + P_nC_n \text{ where } G_n = A_n \cdot B_n$$

- Observe the recursive nature of the expression for the output carry at the nth stage which becomes the input carry for the (n + 1)th stage.
- By successive substitution, it is possible to express the output carry of a higher significant stage in terms of the applied input variables A, B and the carry-in to the LSB adder.
- The carry-in to each stage is the carry-out of the previous stage.
- Based on these, the expressions for the carry-outs of various full adders are as follows:

$$\begin{aligned}
C_2 &= G_1 + P_1 C_1 \\
C_3 &= G_2 + P_2 C_2 \\
&= G_2 + P_2 (G_1 + P_1 C_1) \\
&= G_2 + P_2 G_1 + P_2 P_1 C_1 \\
C_4 &= G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1 \\
C_5 &= G_4 + P_4 C_4 = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 C_1
\end{aligned}$$

- Observe that the final output carry is expressed as a function of the input variables in SOP form, which is a two-level AND-OR or equivalent NAND-NAND form.
- To produce the output carry for any particular stage, it is clear that it requires only that much time required for the signals to pass through two levels only.
- Hence the circuit for look-ahead-carry introduces a delay corresponding to two gate levels.
- The block diagram of a 4-stage look-ahead-carry parallel adder is shown in below figure.
- Observe that the full look-ahead-scheme requires the use of OR gate with (n + 1) inputs and AND gates with number of inputs varying from 2 to (n + 1).



Binary to gray converter

- The input to the 4-bit binary to gray code converter circuit is a 4-bit binary and the output is a 4-bit gray code.
- There are 16 possible combinations of 4-bit binary input and all of them are valid.
- The 4-bit binary and the corresponding gray code are shown in below table.

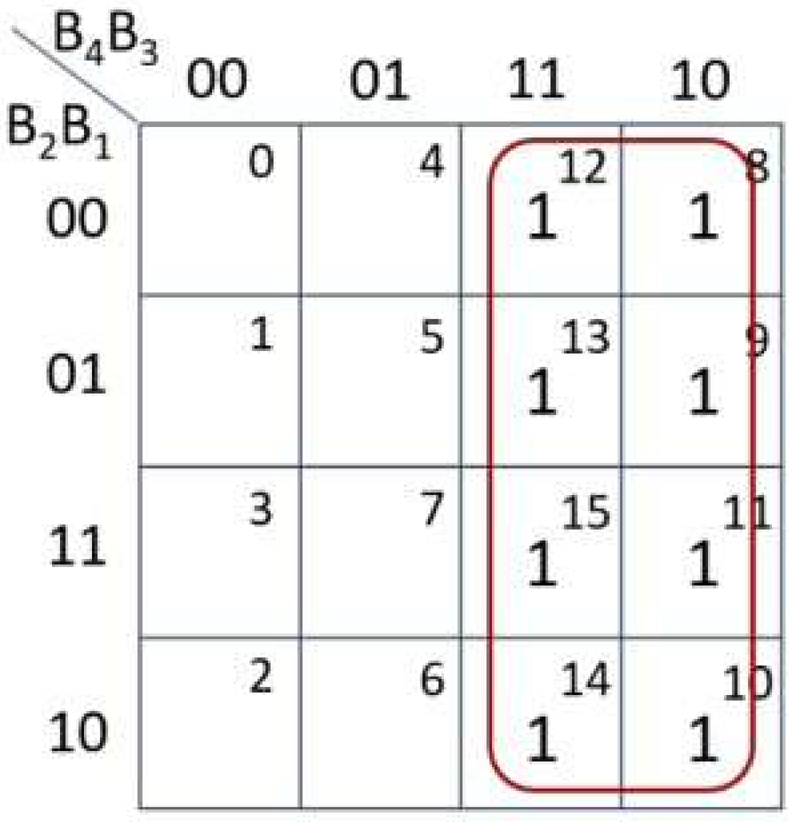
4-bit Binary				4-bit Gray			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

- From the conversion table, we observe that the expressions for the outputs G_4, G_3, G_2, and G_1 are as follows:

$$G_4 = \sum_m(8, 9, 10, 11, 12, 13, 14, 15)$$
$$G_3 = \sum_m(4, 5, 6, 7, 8, 9, 10, 11)$$
$$G_2 = \sum_m(2, 3, 4, 5, 10, 11, 12, 13)$$
$$G_1 = \sum_m(1, 2, 5, 6, 9, 10, 13, 14)$$

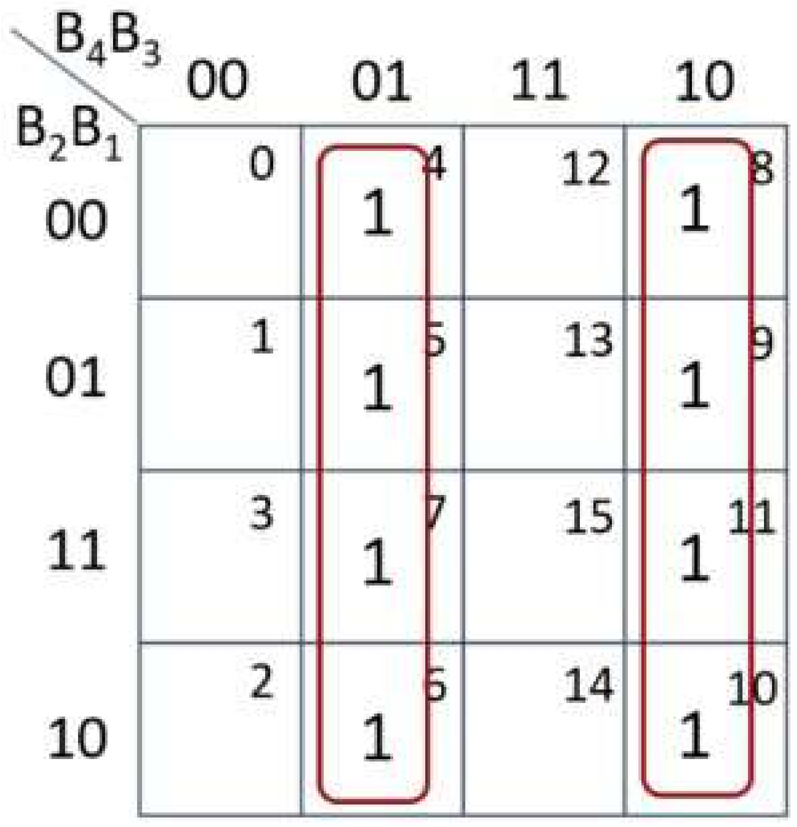
- The K-maps for G_4, G_3, G_2, and G_1 and their minimization are shown below:

For G_4



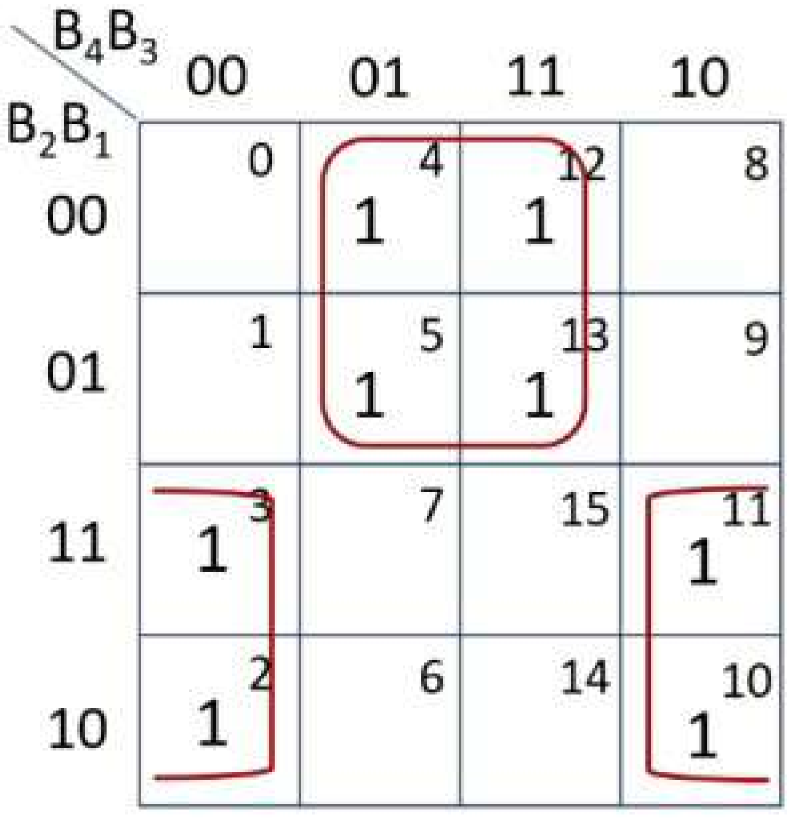
B_2B_1 \ B_4B_3	00	01	11	10
00	0	4	12 1	8 1
01	1	5	13 1	9 1
11	3	7	15 1	11 1
10	2	6	14 1	10 1

For G_3



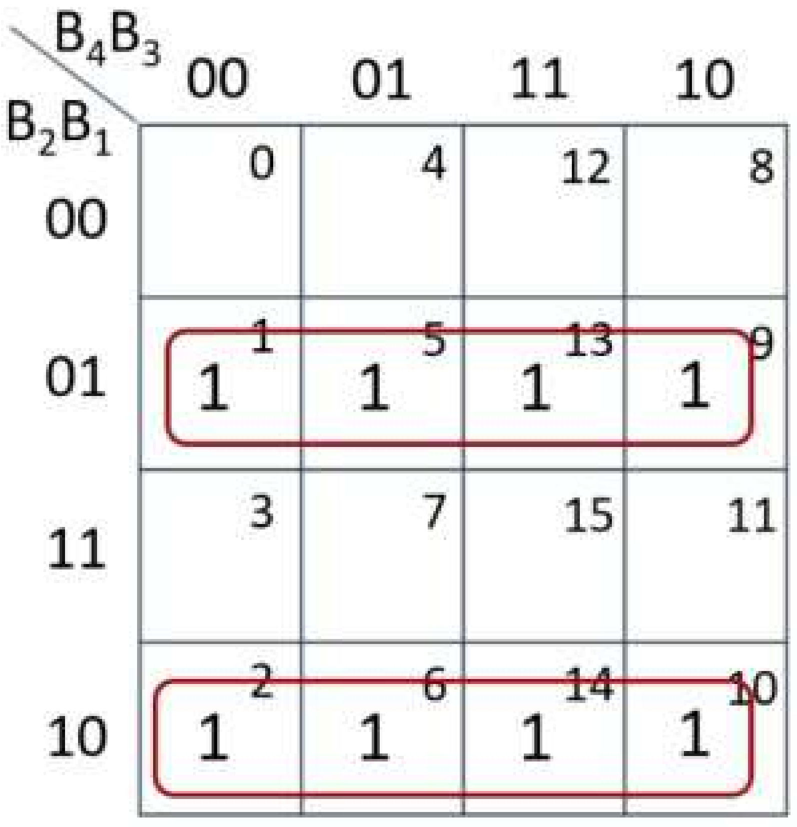
B_2B_1 \ B_4B_3	00	01	11	10
00	0	4 1	12	8 1
01	1	5 1	13	9 1
11	3	7 1	15	11 1
10	2	6 1	14	10 1

For G_2



B_2B_1 \ B_4B_3	00	01	11	10
00	0	4 1	12 1	8
01	1	5 1	13 1	9
11	3 1	7	15	11 1
10	2 1	6	14	10 1

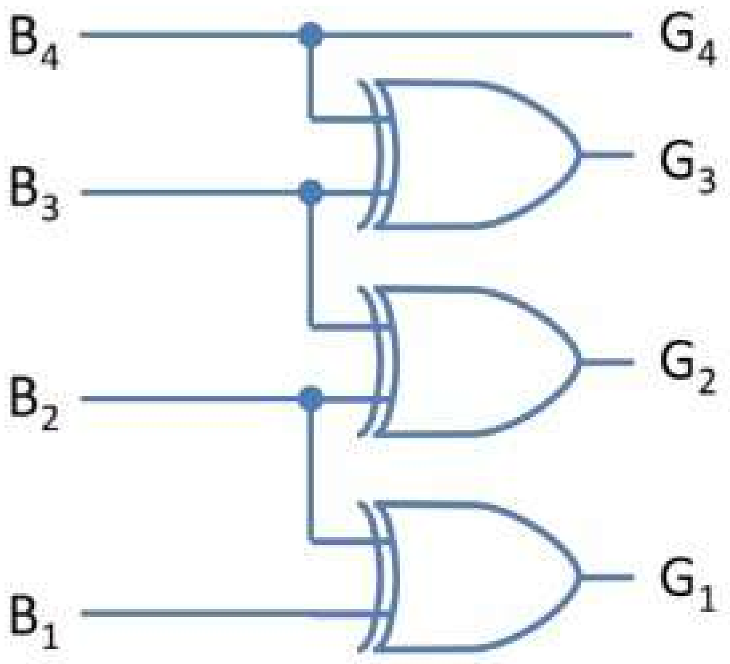
For G_1



B_2B_1 \ B_4B_3	00	01	11	10
00	0	4	12	8
01	1 1	5 1	13 1	9 1
11	3	7	15	11
10	2 1	6 1	14 1	10 1

- The minimal expressions for the outputs obtained from the K-map are:

$G_4 = B_4$
$G_3 = B_4' B_3 + B_4 B_3' = B_4 \oplus B_3$
$G_2 = B_3' B_2 + B_3 B_2' = B_3 \oplus B_2$
$G_1 = B_2' B_1 + B_2 B_1' = B_2 \oplus B_1$



BCD to XS-3 code converter

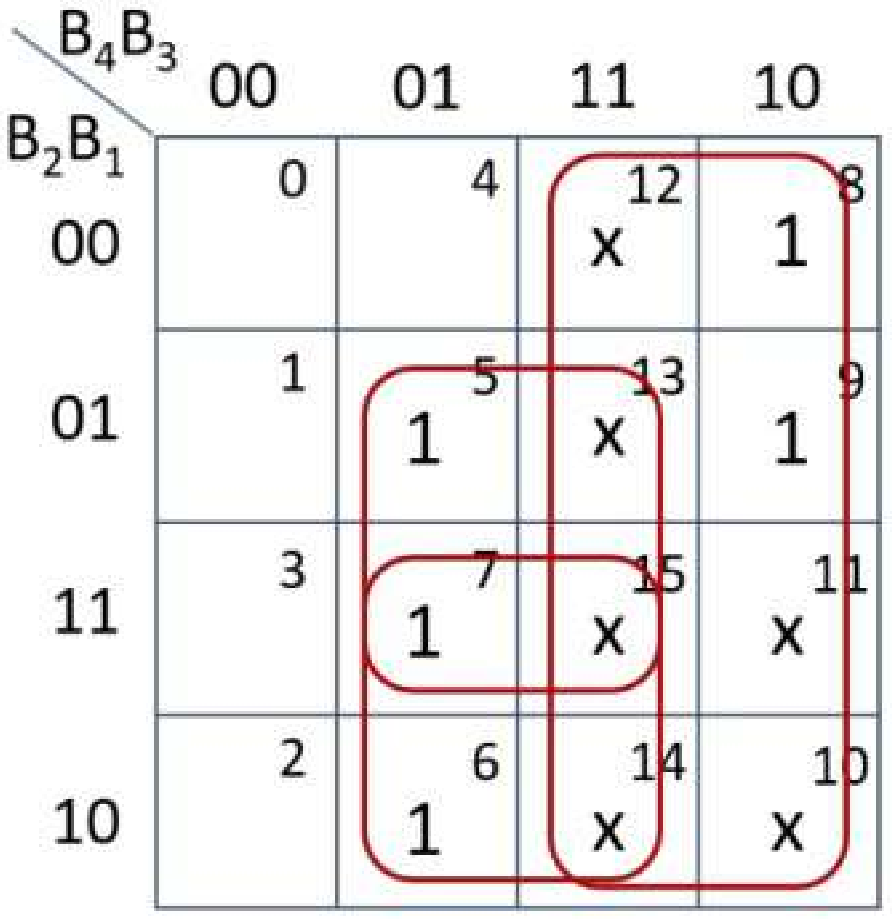
- BCD means 8421 BCD.
- The 4-bit input BCD code (B4 B3 B2 B1) and the corresponding output XS-3 code (X4 X3 X2 X1) numbers are shown in the conversion table in figure.

8421 code				XS-3 code			
B_4	B_3	B_2	B_1	X_4	X_3	X_2	X_1
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

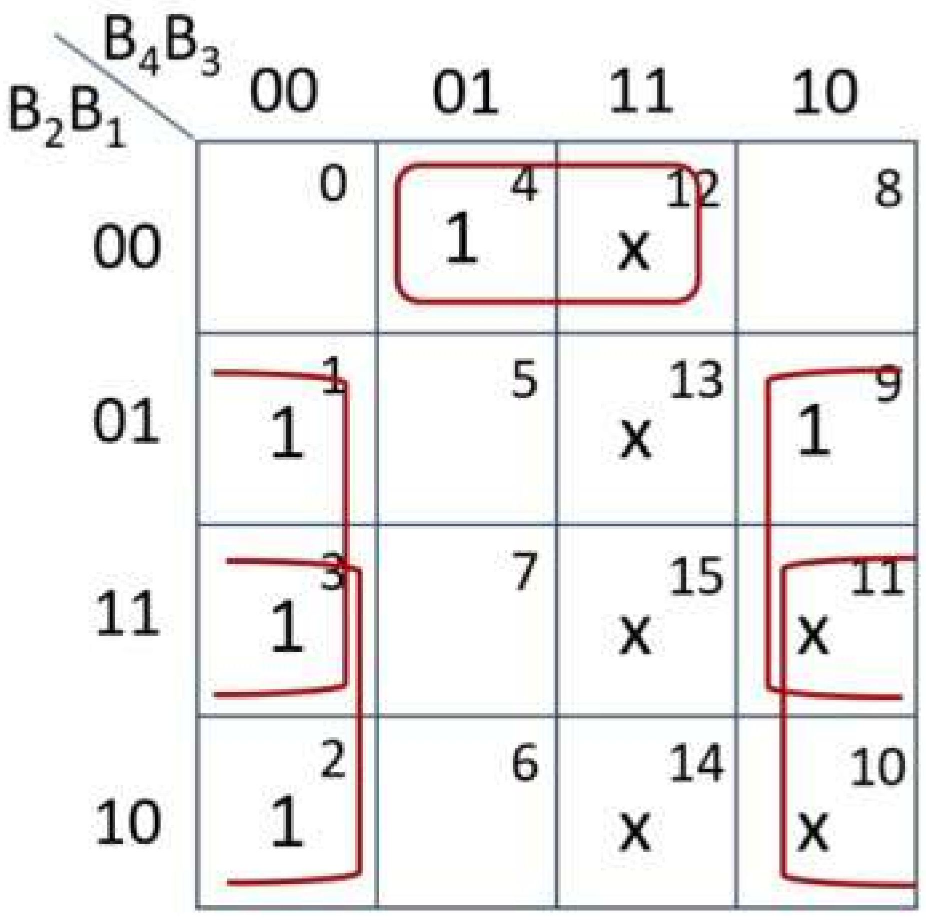
- The input combinations 1010, 1011, 1100, 1101, 1110, and 1111 are invalid in BCD. So they are treated as don't cares.
- From the above truth table, function can be realized as follows:

$$X_4 = \boldsymbol{\Sigma}\, m(5, 6, 7, 8, 9) + d(10, 11, 12, 13, 14, 15)$$
$$X_3 = \boldsymbol{\Sigma}\, m(1, 2, 3, 4, 9) + d(10, 11, 12, 13, 14, 15)$$
$$X_2 = \boldsymbol{\Sigma}\, m(0, 3, 4, 7, 8) + d(10, 11, 12, 13, 14, 15)$$
$$X_1 = \boldsymbol{\Sigma}\, m(0, 2, 4, 6, 8) + d(10, 11, 12, 13, 14, 15)$$

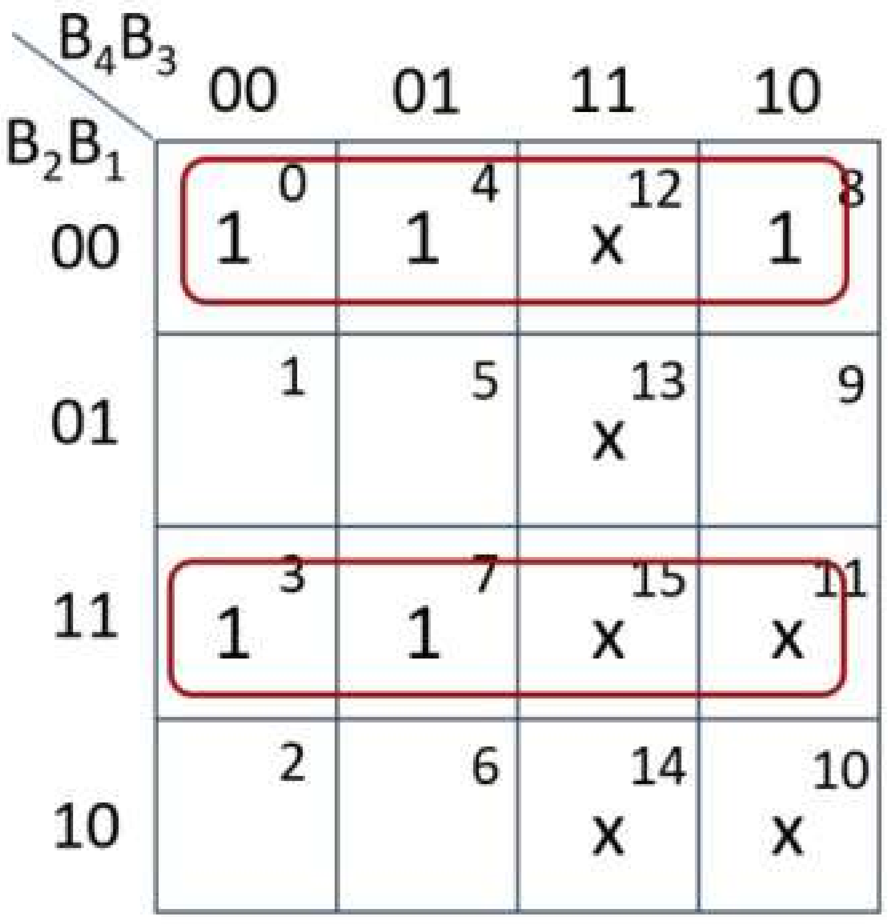
- Drawing K-maps for the outputs X_4, X_3, X_2, and X_1 in terms of the inputs B_4, B_3, B_2, and B_1 and simplifying them, as shown.



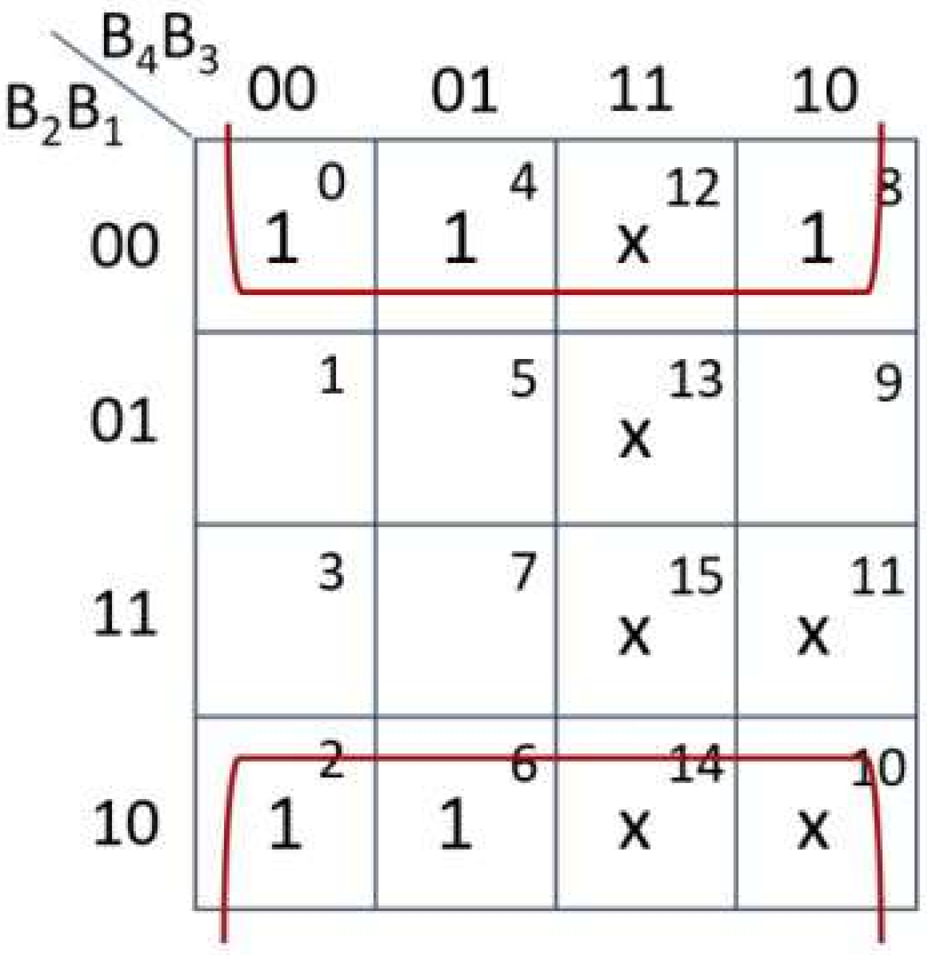
B_2B_1 \ B_4B_3	00	01	11	10
00	0	4	12 x	8 1
01	1	5 1	13 x	9 1
11	3	7 1	15 x	11 x
10	2	6 1	14 x	10 x



B_2B_1 \ B_4B_3	00	01	11	10
00	0	4 1	12 x	8
01	1 1	5	13 x	9 1
11	3 1	7	15 x	11 x
10	2 1	6	14 x	10 x

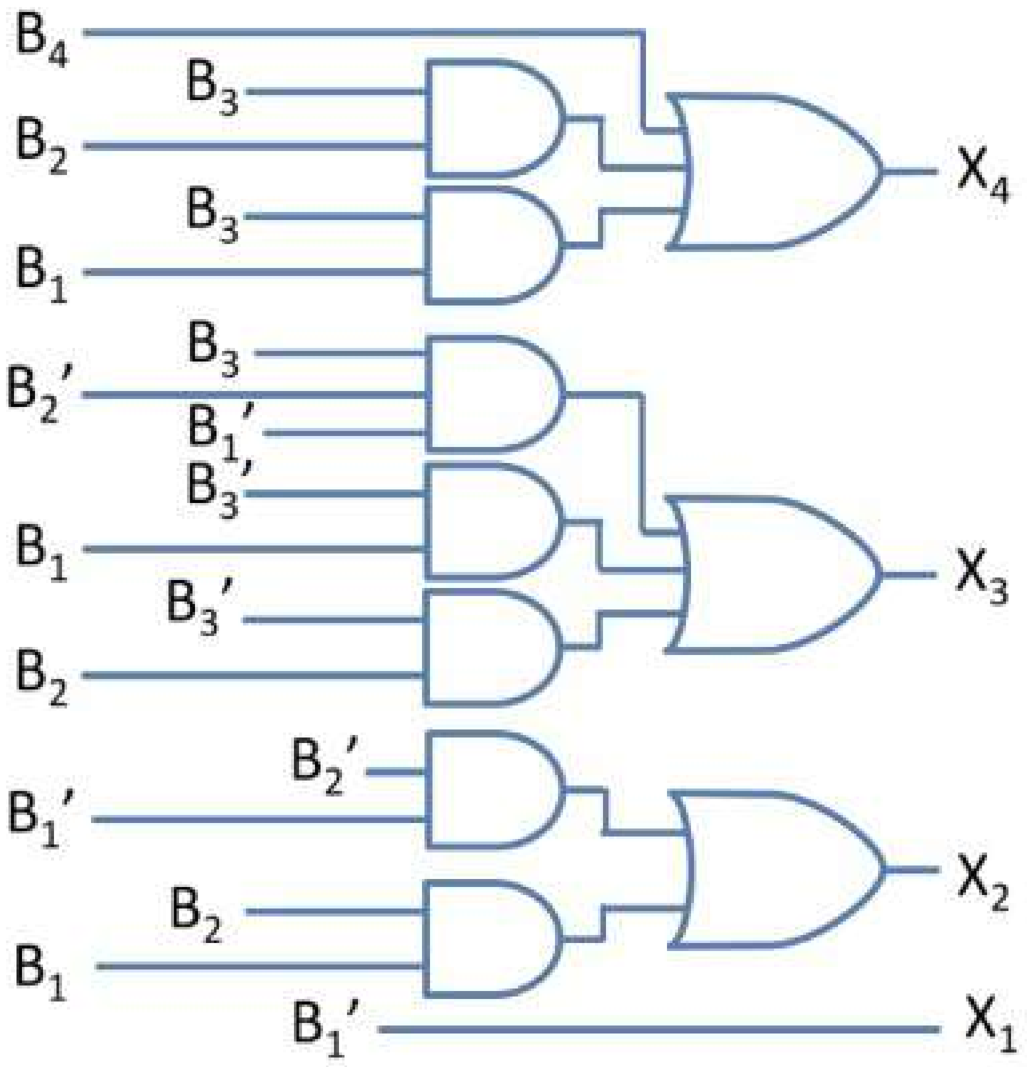


B_2B_1 \ B_4B_3	00	01	11	10
00	0 1	4 1	12 x	8 1
01	1	5	13 x	9
11	3 1	7 1	15 x	11 x
10	2	6	14 x	10 x



B_2B_1 \ B_4B_3	00	01	11	10
00	0 1	4 1	12 x	8 1
01	1	5	13 x	9
11	3	7	15 x	11 x
10	2 1	6 1	14 x	10 x

- The minimal expressions are
  $X_4 = B_4 + B_3B_2 + B_3B_1$
  $X_3 = B_3B_2'B_1' + B_3'B_1 + B_3'B_2$
  $X_2 = B_2'B_1' + B_2B_1$
  $X_1 = B_1'$



1-bit magnitude Comparator

- Let the 1-bit numbers be $A = A_0$ and $B = B_0$
- If $A_0 = 1$ and $B_0 = 0$ then A > B

$$A > B : G = A_0B_0'$$

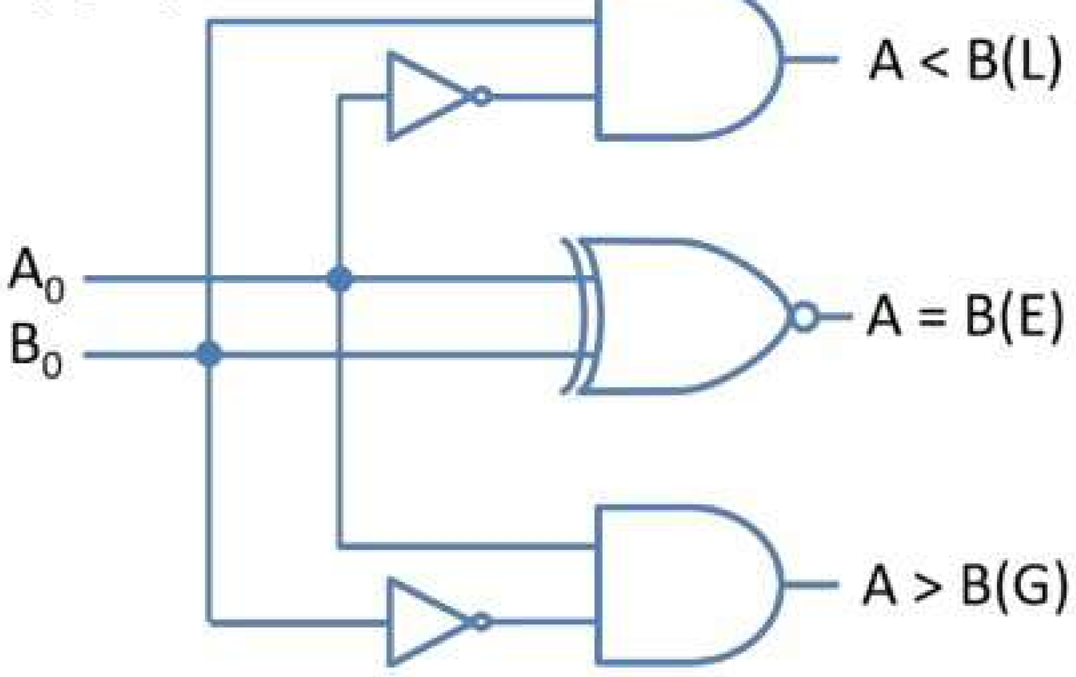
- If $A_0 = 0$ and $B_0 = 1$ then A <B

$$A < B : L = A_0'B_0$$

- If $A_0 = 1$ and $B_0 = 1$ (coincides) then A = B

$$A = B : E = A_0 \odot B_0$$

A_0	B_0	L	E	G
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0



2-bit magnitude Comparator

- The logic for a 2-bit magnitude comparator: Let the two 2-bit numbers be $A = A_1A_0$ and $B = B_1B_0$.
    1. If $A_1 = 1$ and $B_1 = 0$, then A > B or
    2. If A_1 and B_1 coincide and $A_0 = 1$ and $B_0 = 0$, then A > B. So the logic expression for A > B is

$$A > B : G = A_1B_1' + (A1 \odot B1)\, A_0B_0'$$

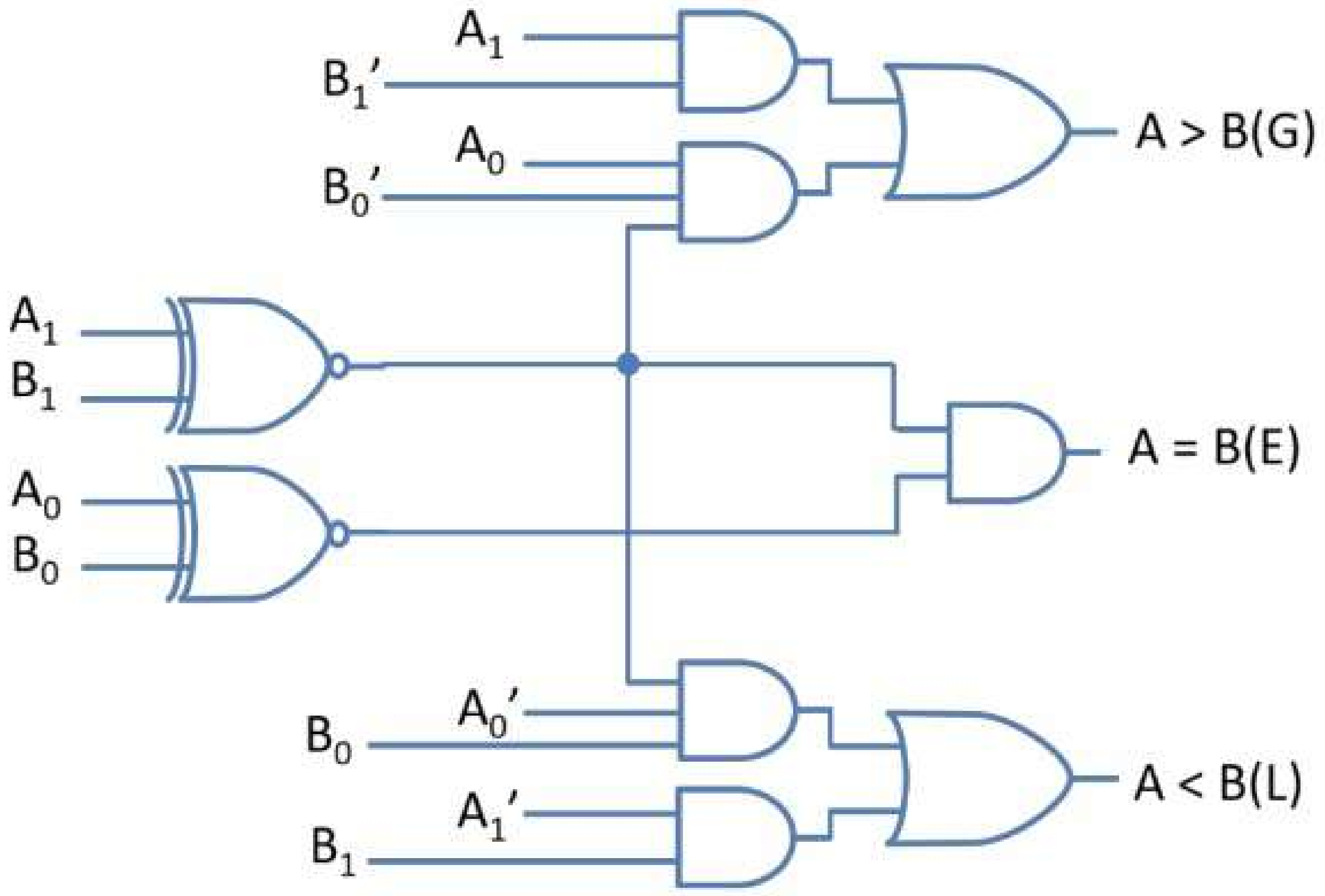
    1. If $A_1 = 0$ and $B_1 = 1$, then A < B or
    2. If A_1 and B_1 coincide and $A_0 = 0$ and $B_0 = 1$, then A < B. So the expression for A < B is

$$A < B : L = A_1'B_1 + (A_1 \odot B_1)\, A_0'B_0$$

If A_1 and B_1 coincide and if A_0 and B_0 coincide then A = B. So the expression for A = B is

$$A = B : E = (A_1 \odot B_1)(A_0 \odot B_0)$$

- The logic diagram for a 2-bit comparator is as shown below:



Parity generator

- Exclusive-OR functions are very useful in systems requiring error detection and correction codes.
- Binary data, when transmitted and processed, is susceptible to noise that can alter its 1s to Os and 0s to 1s.
- To detect such errors, an additional bit called the *parity bit* is added to the data bits and the word containing the data bits and the parity bit is transmitted.
- At the receiving end the number of 1s in the word received are counted and the error, if any, is detected.
- This parity check, however, detects only single bit errors.
- The circuit that generates the parity bit in the transmitter is called a parity generator.
- The circuit that checks the parity in the receiver is called a parity checker.
- A parity bit, a 0 or a 1 is attached to the data bits such that the total number of 1s in the word is even for even parity and odd for odd parity.
- The parity bit can be attached to the code group either at the beginning or at the end depending on system design.
- A given system operates with either even or odd parity but not both.
- So, a word always contains either an even or an odd number of 1s.

- At the receiving end, if the word received has an even number of 1s in the odd parity system or an odd number of 1s in the even parity system, it implies that an error has occurred.
- In order to check or generate the proper parity bit in a given code word, the basic principle used is, "the modulo sum of an even number of 1s is always a 0 and the modulo sum of an odd number of 1s is always a 1".
- Therefore, in order to check for an error, all the bits in the received word are added.
- If the modulo sum is a 0 for an odd parity system or a 1 for an even parity system, an error is detected.

Example - 1 Design 3-bit parity generator using even parity bit.

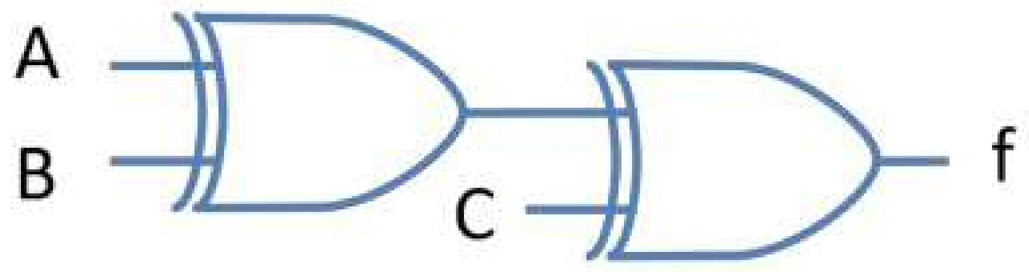
Inputs			Outputs parity bit (f)
A	B	C	
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$f = A'B'C + A'BC' + ABC + AB'C'$$

$$f = A'(B'C + BC') + A(BC + B'C')$$

$$f = A'(B \oplus C) + A(B \oplus C)'$$

$$f = A \oplus B \oplus C$$



Demultiplexer

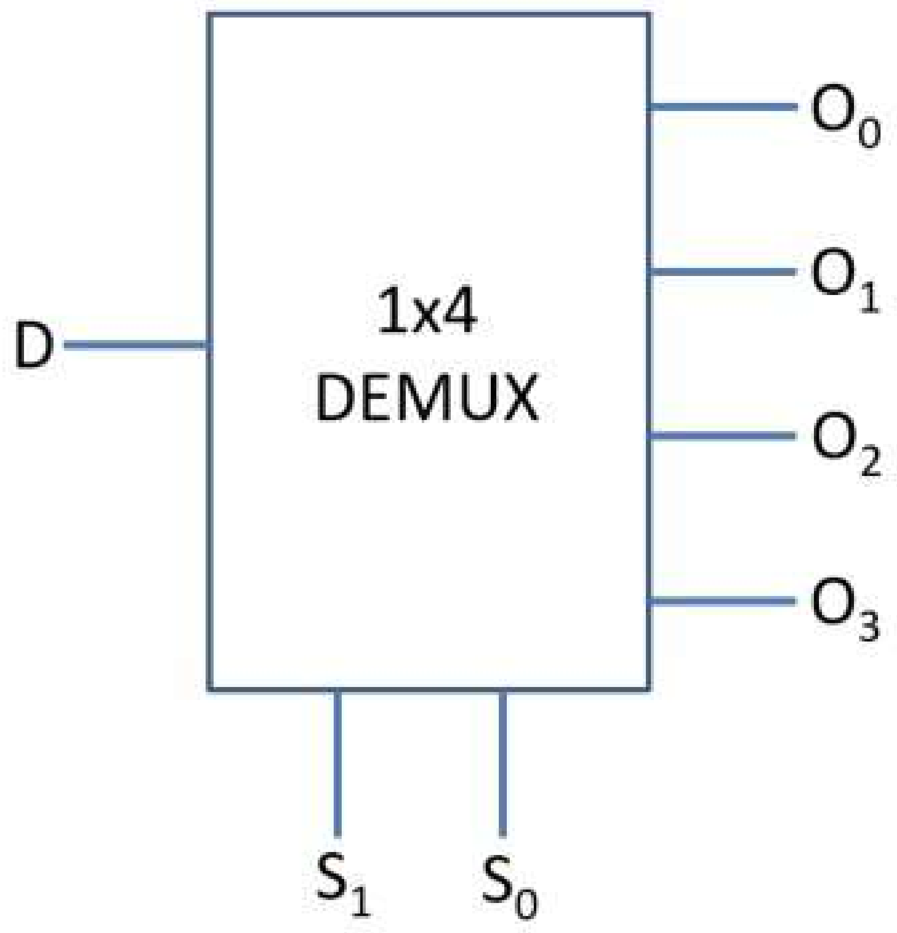
- A multiplexer takes several inputs and transmits one of them to the output.
- A demultiplexer performs the reverse operation; it takes a single input and distributes it over several outputs.
- So, a demultiplexer can be thought of as a 'distributor', since it transmits the same data to different destinations.
- Thus, whereas a multiplexer is an N-to-1 device, demultiplexer is a 1-to-N device.
- Consider an integer 'm', which is constrained by the following relation:

m = 2^n, where m and n are both integers.
A **1-to-m** Demultiplexer has
One Input: D
m Outputs: O_0, O_1, O_2, ... $O_{(m-1)}$
n Control inputs: S_0, S_1, S_2, ... $S_{(n-1)}$
One (or more) Enable input(s)

- Such that D may be transfer to one of the outputs, depending upon the control inputs.

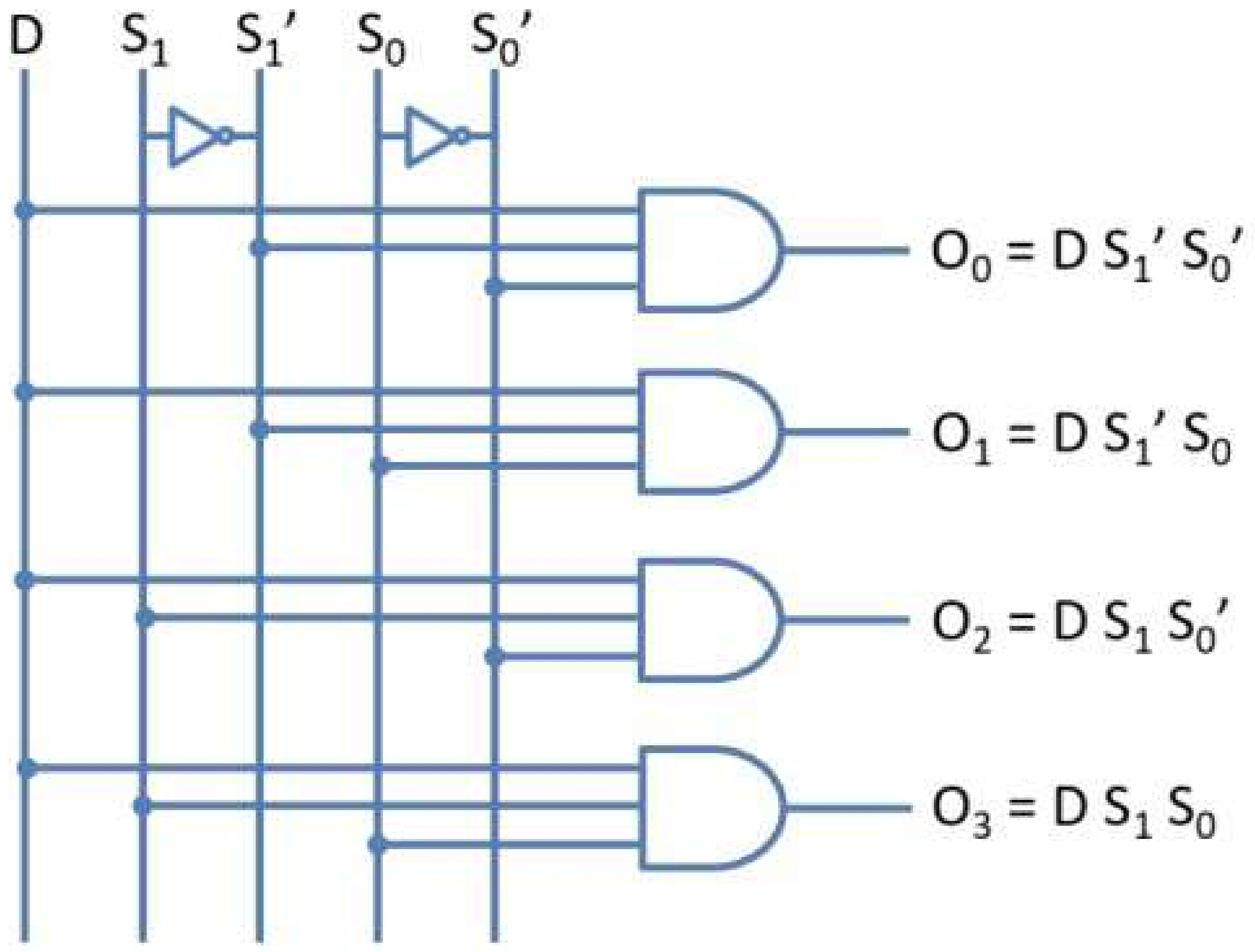
1x4 Demultiplexer

- The block diagram and truth table of 1 x 4 demultiplexer is as follows.



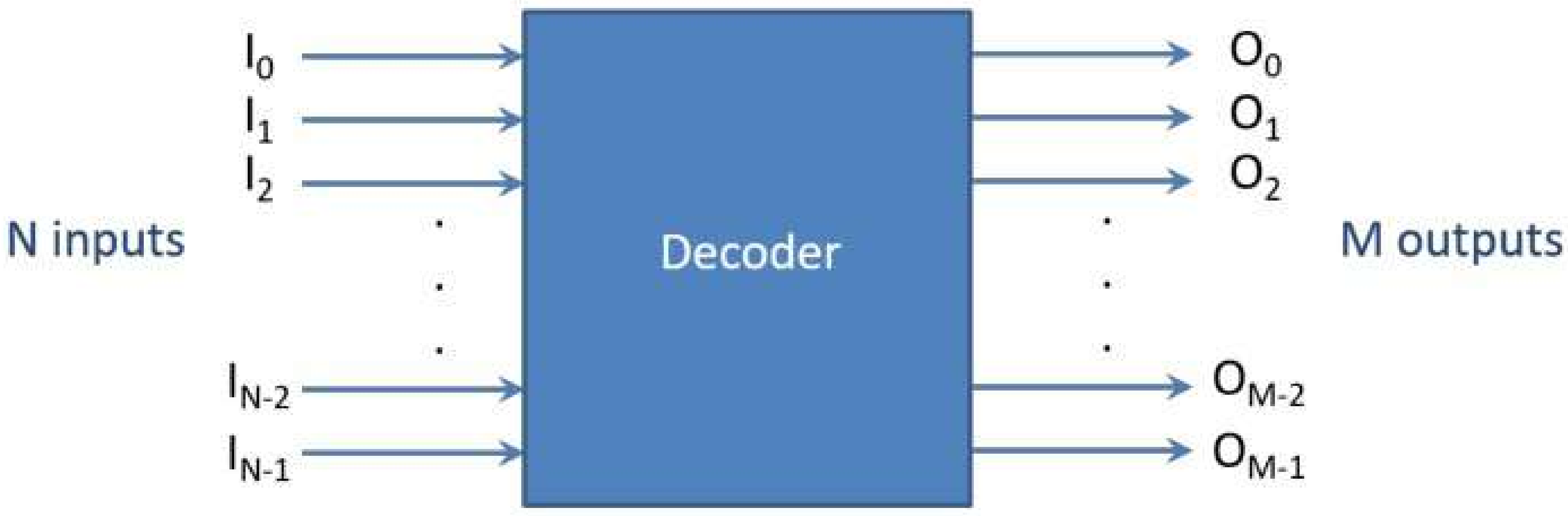
Select code		Outputs			
S_1	S_0	O_3	O_2	O_1	O_0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

- The following figure describes the logic circuit for 1 x 4 demultiplexer.



Decoder

- A decoder is a logic circuit that converts an N-bit binary input code into M output lines such that only one output line is activated for each one of the possible combinations of inputs.
- In other words, we can say that a decoder identifies or recognizes or detects a particular code.



- Figure shows the general decoder diagram with N inputs and M outputs.
- Since each of the N inputs can be a 0 or a 1, there 2^N possible input combinations or codes.

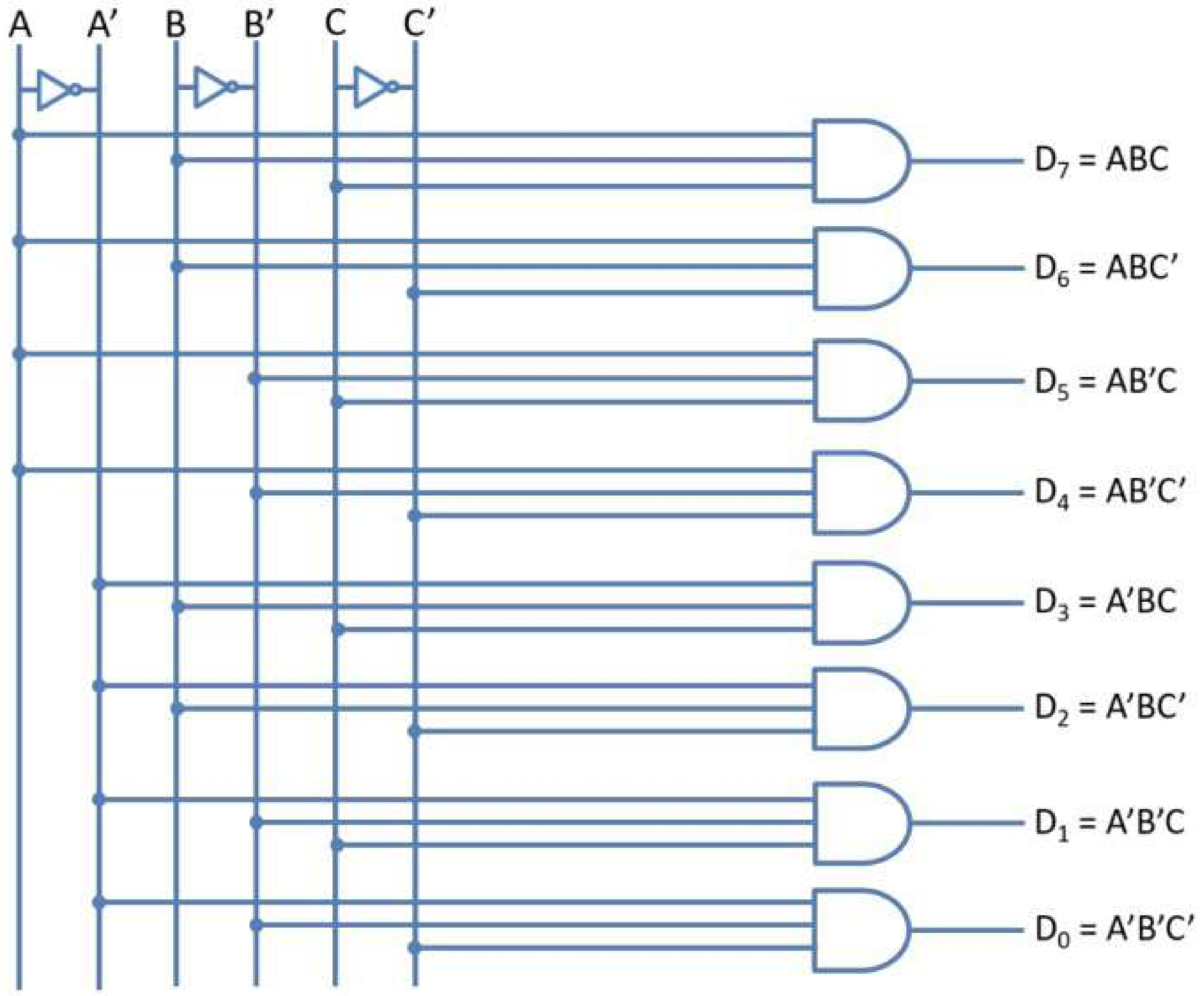
- For each of these input combinations, only one of the M outputs will be active, all the other outputs will remain inactive.

3 to 8 Decoder

- Figure shows the circuitry for a decoder with three inputs and eight outputs.
- It uses all AND gates, and therefore, the outputs are active-HIGH. For active-LOW outputs, NAND gates are used
- The truth table of the decoder is shown below.

Inputs			Outputs							
A	B	C	D_0 A'B'C'	D_1 A'B'C	D_2 A'BC'	D_3 A'BC	D_4 AB'C'	D_5 AB'C	D_6 ABC'	D_7 ABC
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

- This decoder can be referred to in several ways.
- It can be called a 3-line to 8-line decoder because it has three lines and eight output lines.
- It is also called a binary-to-octal decoder because it takes a 3-bit binary input code and activates one of the eight (octal) outputs corresponding to that code.
- It is also referred to as a 1-of-8 decoder because only one of the eight outputs is activated at one time.



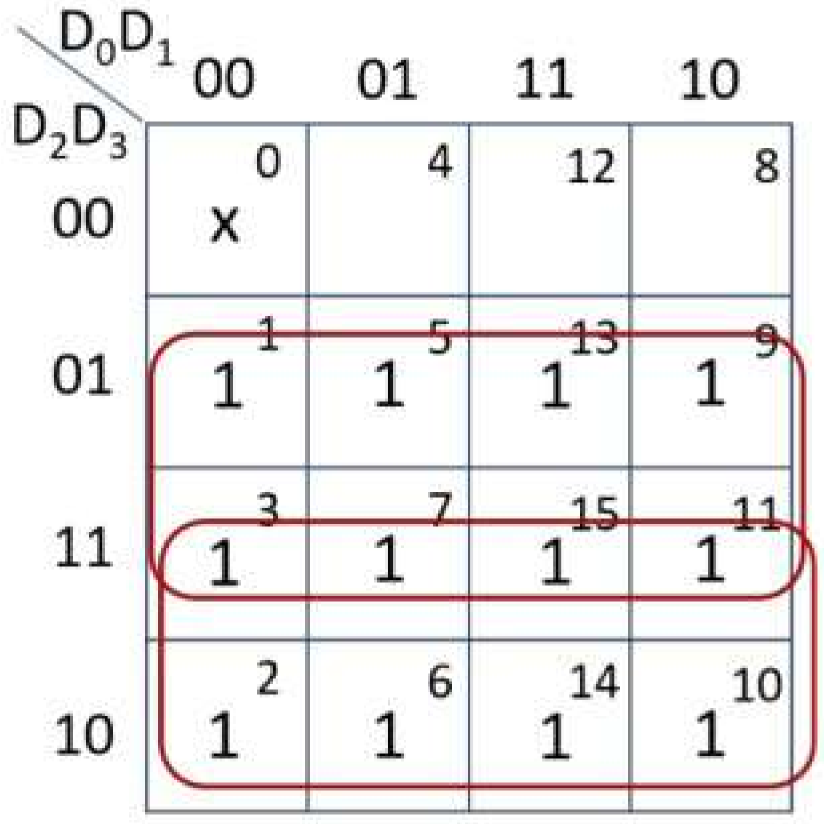
Priority encoder

- A priority encoder is a logic circuit that responds to just one input in accordance with some priority system, among all those that may be simultaneously HIGH.
- The most common priority system is based on the relative magnitudes of the inputs; whichever decimal input is the largest, is the one that is encoded.
- The truth table of 4-input priority encoder and expression are given below:

Inputs				Outputs		
D_0	D_1	D_2	D_3	A	B	V
0	0	0	0	x	x	0
1	0	0	0	0	0	1
x	1	0	0	0	1	1
x	x	1	0	1	0	1
x	x	x	1	1	1	1

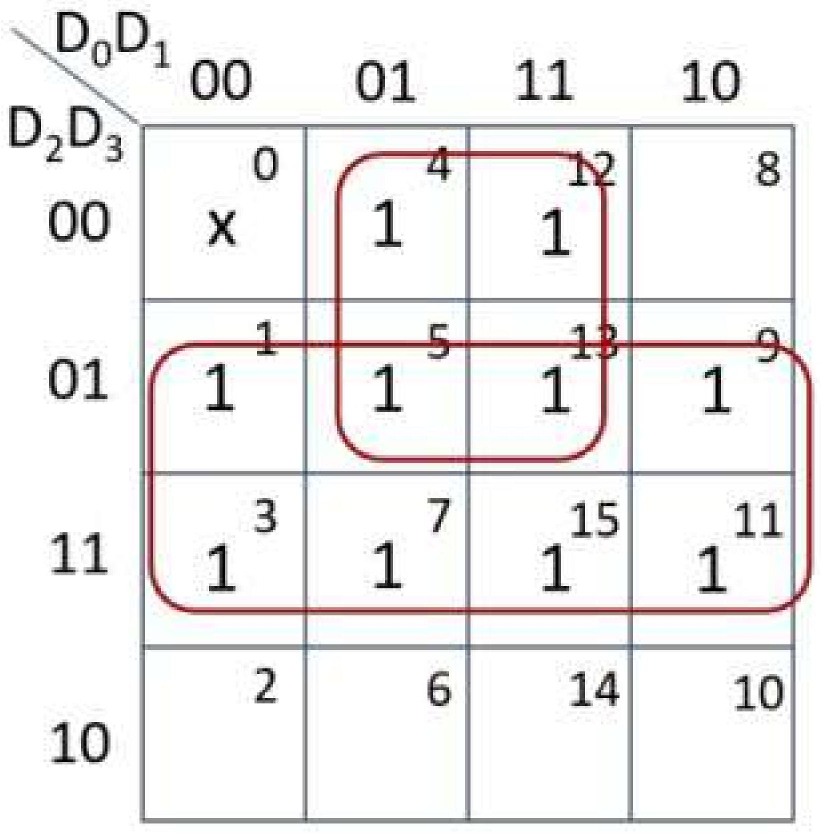
A = $\sum_m(1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15)$
B = $\sum_m(1, 3, 4, 5, 7, 9, 11, 12, 13, 15)$
V = $\sum_m(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)$

- In addition to the outputs A and B, the circuit has a third output designated by V.
- This is a valid bit indicator that is set to 1 when one or more inputs are equal to 1.
- If all inputs are 0, there is no valid input and V is equal to 0, the two other outputs are not inspected when V equals 0 and are specified as don't care conditions.
- According to the truth table, the higher the subscript number, the higher the priority of the input.
- The K-map for A, B and V with minimized expression are shown below:



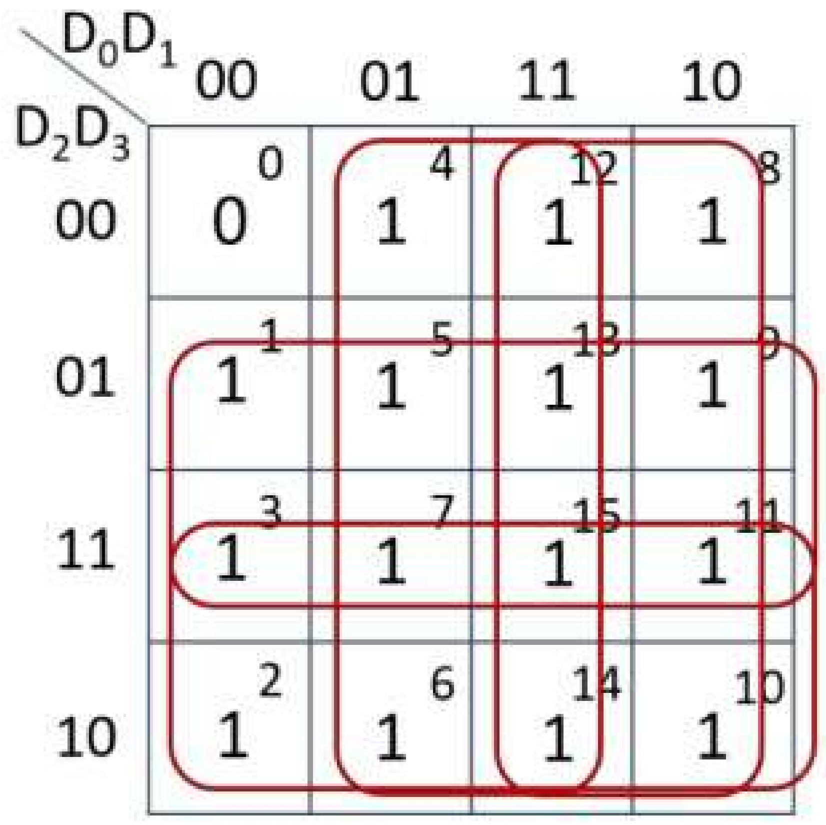
D_2D_3 \ D_0D_1	00	01	11	10
00	0 x	4	12	8
01	1 1	5 1	13 1	9 1
11	3 1	7 1	15 1	11 1
10	2 1	6 1	14 1	10 1

A = $D_3 + D_2$



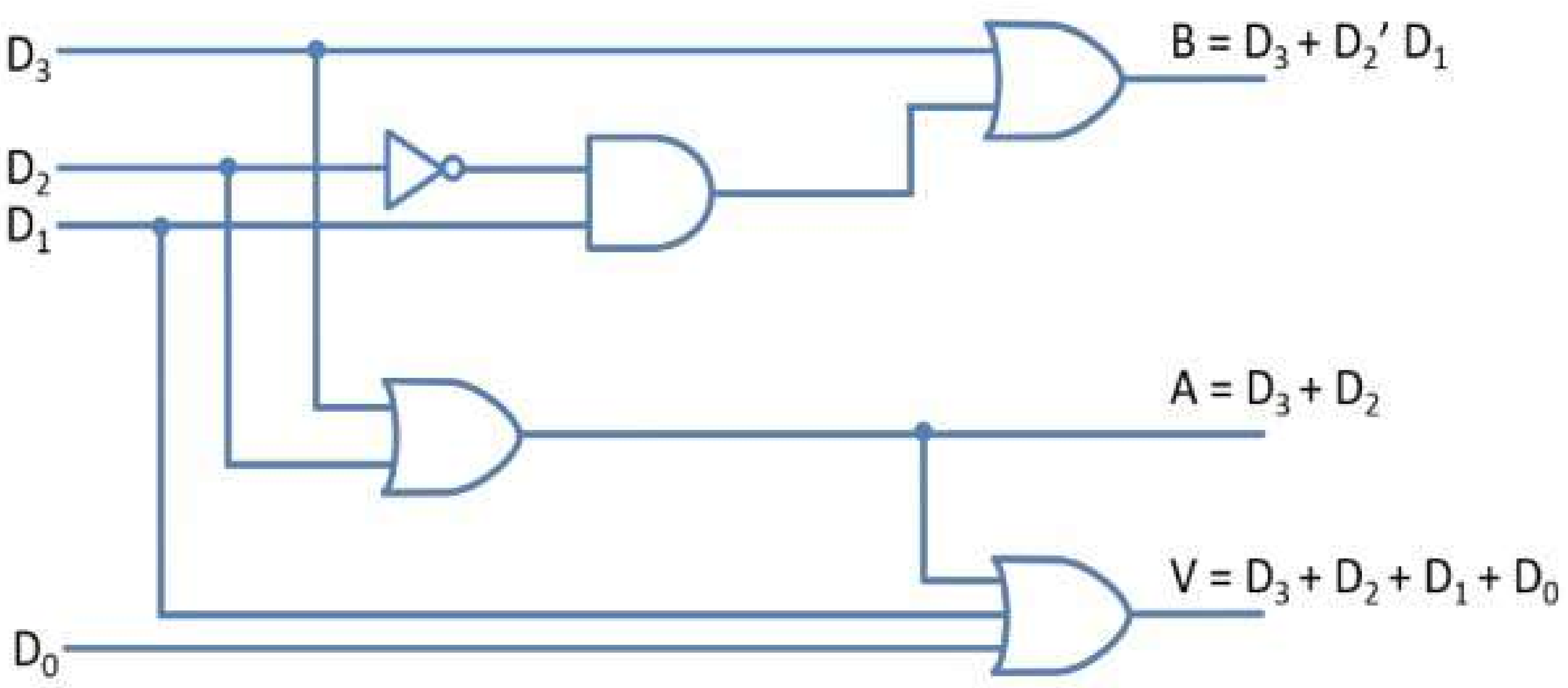
D_2D_3 \ D_0D_1	00	01	11	10
00	0 x	4 1	12 1	8
01	1 1	5 1	13 1	9 1
11	3 1	7 1	15 1	11 1
10	2	6	14	10

B = $D_3 + D_2'D_1$



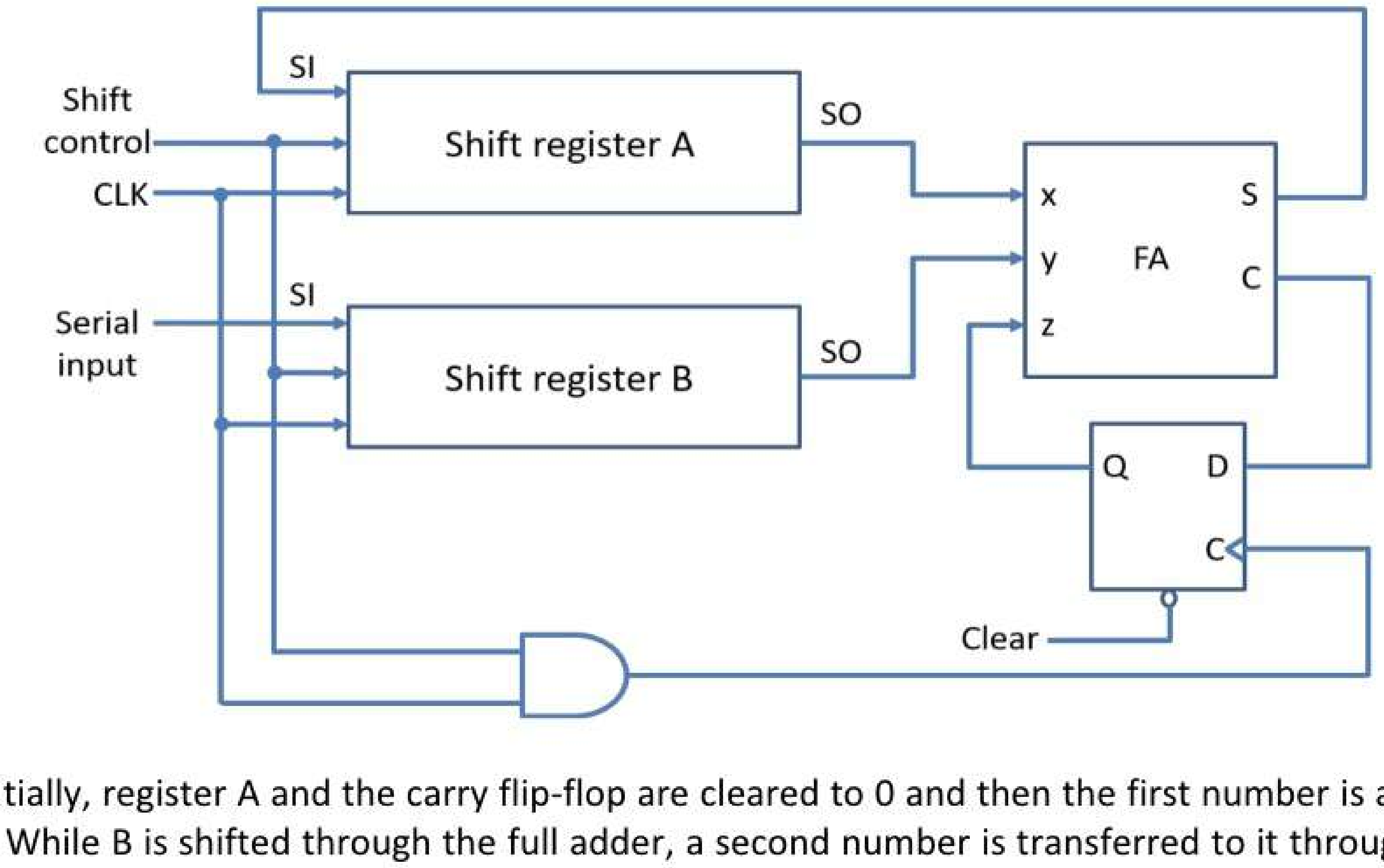
D_2D_3 \ D_0D_1	00	01	11	10
00	0 0	4 1	12 1	8 1
01	1 1	5 1	13 1	9 1
11	3 1	7 1	15 1	11 1
10	2 1	6 1	14 1	10 1

V = $D_3 + D_2 + D_1 + D_0$



Serial adder

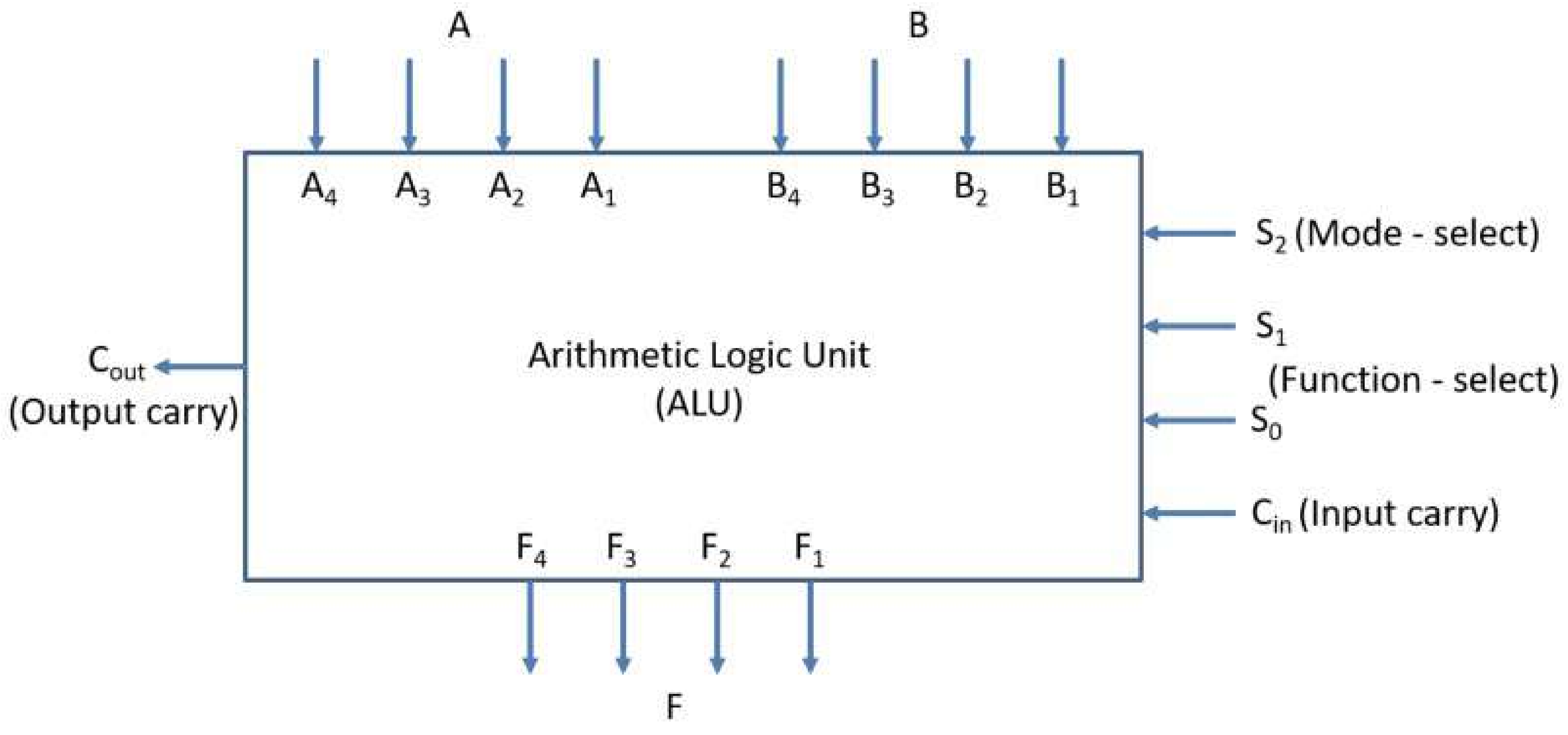
- A serial udder is used to add binary numbers in serial form.
- The two binary numbers to be added serially are stored in two shift registers A and B.
- Bits are added one pair at a time through a single full adder (FA) circuit as shown in Figure.
- The carry out of the full-adder is transferred to a D flip flop.
- The output of this flip-flop is then used as the carry input for the next pair of significant bits.
- The sum bit from the S output of the full adder could be transferred to a third shift register.
- By shifting the sum into A while, the bits of A are shifted out, it is possible to use one register for storing both augend and the sum bits.
- The serial input register B can be used to transfer a new binary number while the addend bits are shifted out during the addition.
- The operation of the serial adder is as follows.
- Initially register A holds the augend, register B holds the addend and the carry flip-flop is cleared to 0. The outputs (SO) of A and B provide a pair of significant bits for the full-adder at x and y.
- The shift control enables both registers and carry flip flop, so, at the clock pulse both registers are shifted once to the right, the sum bit from S enters the left most flip-flop of A, and the output carry is transferred into flip-flop Q.
- The shift control enables the registers for a number of clock pulses equal to the number of bits of the registers.
- For each succeeding clock pulse a new sum bit is transferred to A, a new carry is transferred to Q, and both registers are shifted once to the right.
- This process continues until the shift control is disabled.
- Thus, the addition is accomplished by passing each pair of bits together with the previous carry through a single full adder circuit and transferring the sum, one bit at a time, into register A.



- Initially, register A and the carry flip-flop are cleared to 0 and then the first number is added from B. While B is shifted through the full adder, a second number is transferred to it through its serial input.
- The second number is then added to the content of register A while a third number is transferred serially into register B.
- This can be repeated to form the addition of two, three, or more numbers and accumulate their sum in register A.

Arithmetic logic unit (ALU)

- A very popular and widely used combinational circuit is ALU which is capable of performing arithmetic as well as logical operations.
- To perform a microoperation, the contents of specified registers are placed in the inputs of common ALU. The ALU performs an operation and the result of the operation is then transferred to a destination register.
- The ALU is a combinational circuit in which the entire register transfer operation from the source registers through the ALU and into destination register can be performed during one clock pulse.
- Figure shows the basic block diagram of ALU and different kind of operations performed by ALU depending on select lines.



Selection				Output	Function
S_2	S_1	S_0	C_{in}		
0	0	0	0	F = A	Transfer A
0	0	0	1	F = A + 1	Increment A
0	0	1	0	F = A + B	Addition
0	0	1	1	F = A + B + 1	Add with carry
0	1	0	0	F = A – B – 1	Subtract with borrow
0	1	0	1	F = A – B	Subtraction
0	1	1	0	F = A – 1	Decrement A
0	1	1	1	F = A	Transfer A
1	0	0	X	F = A + B	OR
1	0	1	X	F = A ⊕ B	XOR
1	1	0	X	F = A . B	AND
1	1	1	X	F = A'	Complement A