Digital System Design

Lecture 12

Combinational Logic Design

Binary Adder-Subtractor

Objectives:

1. **Half Adder.**
2. **Full Adder.**
3. **Binary Adder.**
4. **Binary Subtractor.**
5. **Binary Adder-Subtractor.**

1. Half Adder

Half Adder: is a combinational circuit that performs the addition of two bits, this circuit needs two binary inputs and two binary outputs.

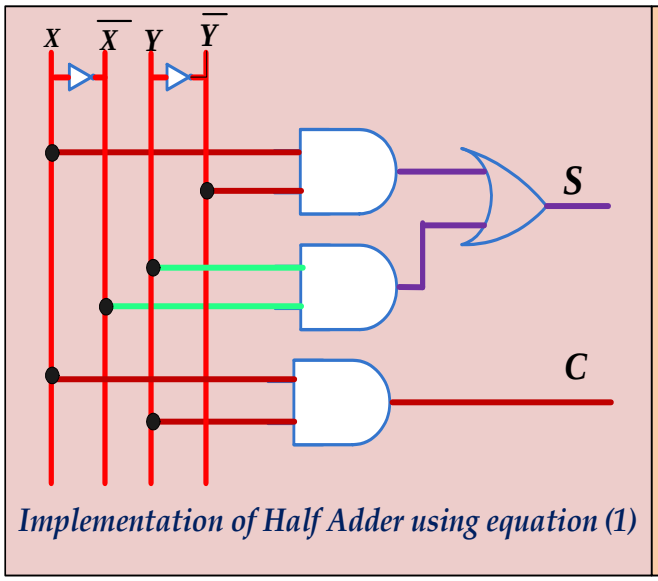
Inputs		*Outputs*	
X	*Y*	*C*	*S*
0	*0*	*0*	*0*
0	*1*	*0*	*1*
1	*0*	*0*	*1*
1	*1*	*1*	*0*
Truth table			

The simplified Boolean function from the truth table:

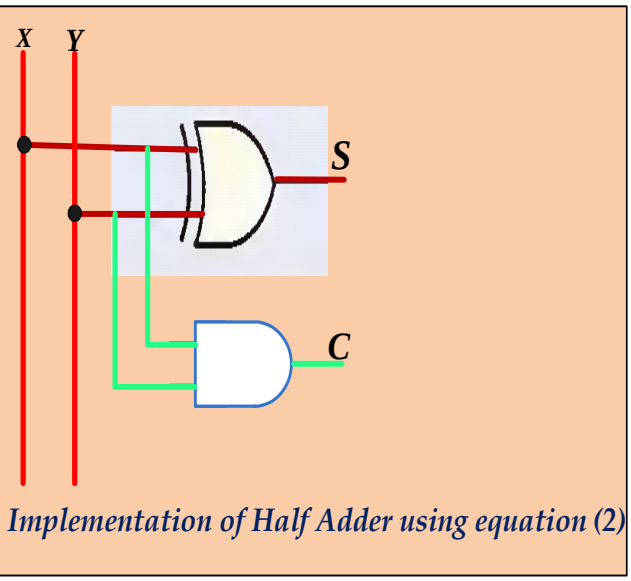
$$\begin{cases} S = \bar{X}Y + X\bar{Y} \\ C = XY \end{cases} \quad 1 \quad \text{(Using sum of product form)}$$

Where ***S*** is the sum and ***C*** is the carry.

$$\begin{cases} S = X \oplus Y \\ C = XY \end{cases} \quad 2 \quad \text{(Using XOR and AND Gates)}$$



Implementation of Half Adder using equation (1)



Implementation of Half Adder using equation (2)

- The implementation of half adder using ***exclusive–OR*** and an ***AND*** gates is used to show that two half adders can be used to construct a full adder.
- The inputs to the ***XOR*** gate are also the inputs to the ***AND*** gate.

2. Full Adder

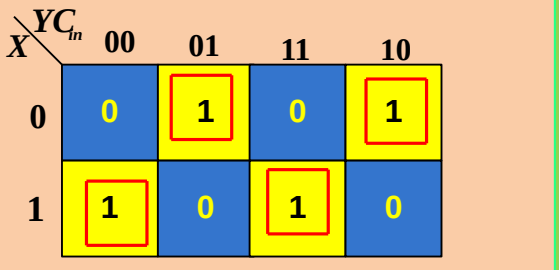
Full Adder is a combinational circuit that performs the addition of three bits (two significant bits and previous carry).

- It consists of ***three inputs and two outputs***, two inputs are the bits to be added, the third input represents the carry form the previous position.
- The full adder is usually a component in a cascade of adders, which add 8, 16, etc, binary numbers.

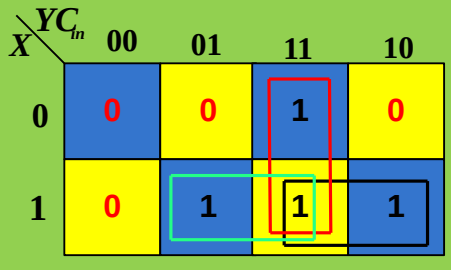
Inputs			Outputs	
X	Y	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Truth table for the full adder

- The S output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1.
- The C_{out} output has a carry 1 if two or three inputs are equal to 1.
- The Karnaugh maps and the simplified expression are shown in the following figures:



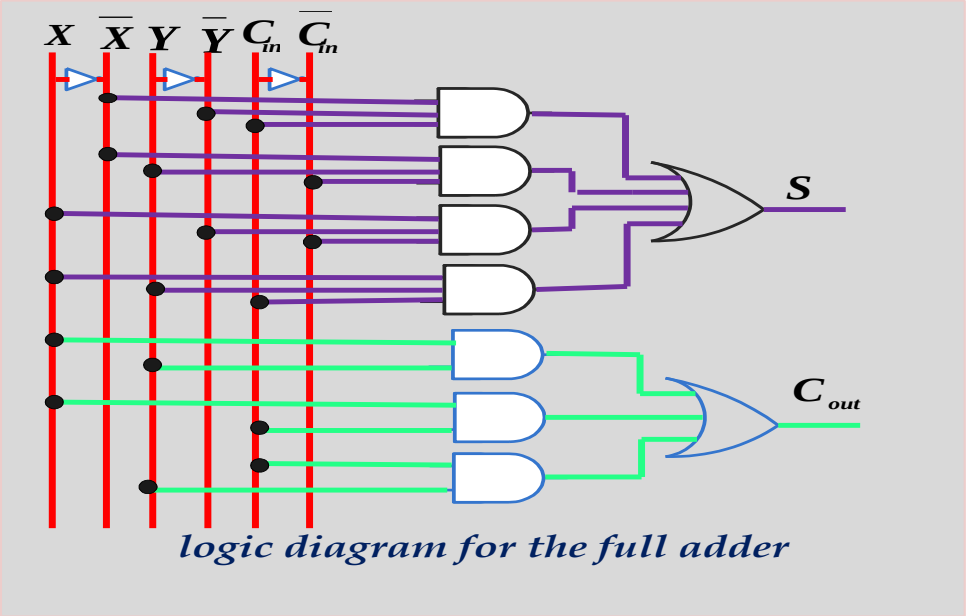
$$S = \overline{X}\overline{Y}C_{in} + \overline{X}Y\overline{C_{in}} + X\overline{Y}\overline{C_{in}} + XYC_{in}$$



$$C_{out} = XY + XC_{in} + YC_{in}$$

$$\left.\begin{cases} S = \overline{X}\,\overline{Y}C_{in} + \overline{X}Y\overline{C_{in}} + X\overline{Y}\,\overline{C_{in}} + XYC_{in} \\ C_{out} = XY + XC_{in} + YC_{in} \end{cases}\right\} 1 \quad \textit{(Sum of products)}$$

- The ***logic diagrams*** for the full adder implemented in ***sum-of-products*** form are the following:



logic diagram for the full adder

- It can also be implemented using ***two half adders*** and ***one OR gate*** (using ***XOR*** gates).

$$\left.\begin{matrix} S = C_{in} \oplus (X \oplus Y) \\ C_{out=}\, C_{in} \cdot (X \oplus Y) + XY \end{matrix}\right\}$$

Proof:

The sum:

$$S = \overline{X}\,\overline{Y}C_{in} + \overline{X}Y\overline{C_{in}} + X\overline{Y}\,\overline{C_{in}} + XYC_{in}$$

$$= \overline{C_{in}}(\overline{X}Y + X\overline{Y}) + C_{in}(\overline{X}\,\overline{Y} + XY)$$

$$= \overline{C_{in}}(\overline{X}Y + X\overline{Y}) + C_{in}\overline{(\overline{X}Y + X\overline{Y})}$$

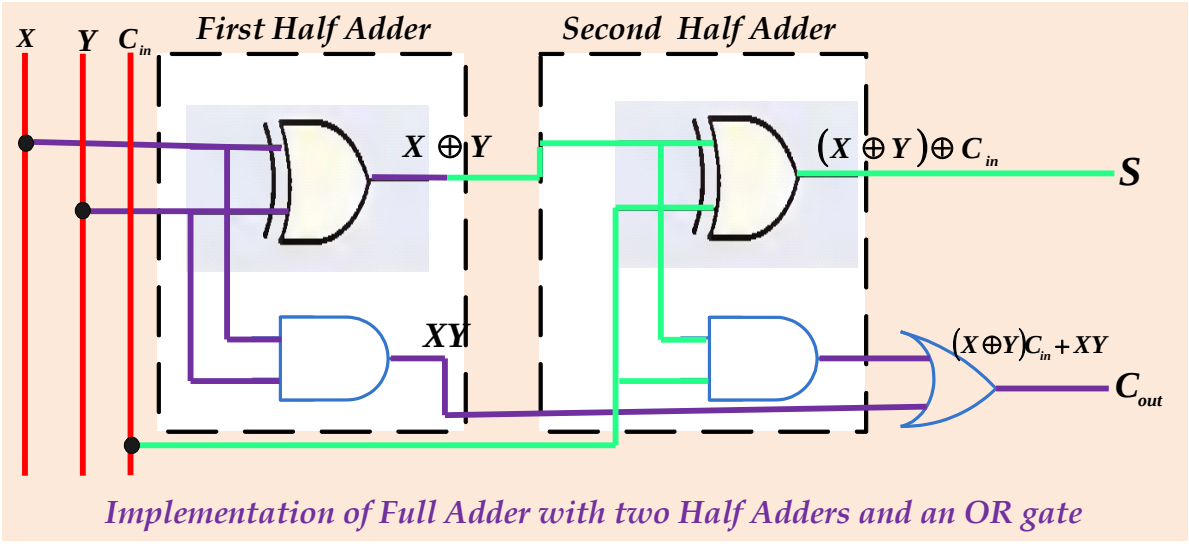
$$S = C_{in} \oplus (X \oplus Y)$$

The carry output:

$$C_{out} = \overline{X}YC_{in} + X\overline{Y}C_{in} + XYC_{in} + XY\,\overline{C_{in}}$$

$$= C_{in}(\overline{X}Y + X\overline{Y}) + XY(C_{in} + \overline{C_{in}})$$

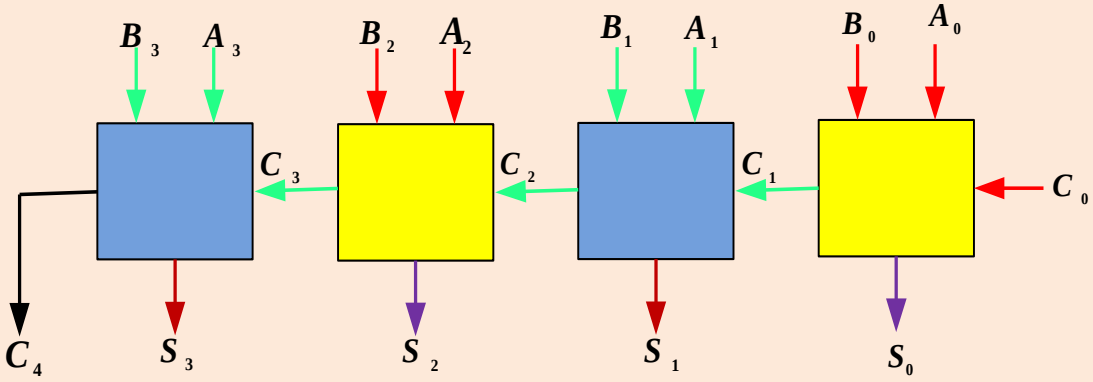
$$C_{out=}\, C_{in} \cdot (X \oplus Y) + XY$$



Implementation of Full Adder with two Half Adders and an OR gate

3. Binary Adder (Asynchronous Ripple-Carry Adder)

- A binary adder is a digital circuit that produces the ***arithmetic sum of two binary numbers***.
- A binary adder can be constructed with ***full adders connected in cascade*** with the output carry form each full adder connected to the input carry of the next full adder in the chain.
- The ***four-bit adder*** is a typical example of a ***standard component*** .It can be used in many application involving arithmetic operations.



Four-bit Adder (Ripple Carry Adder)

- The input carry to the adder is C_0 and it ripples through the full adders to the output carry C_4.
- n-bit binary adder requires n full adders.

Example:

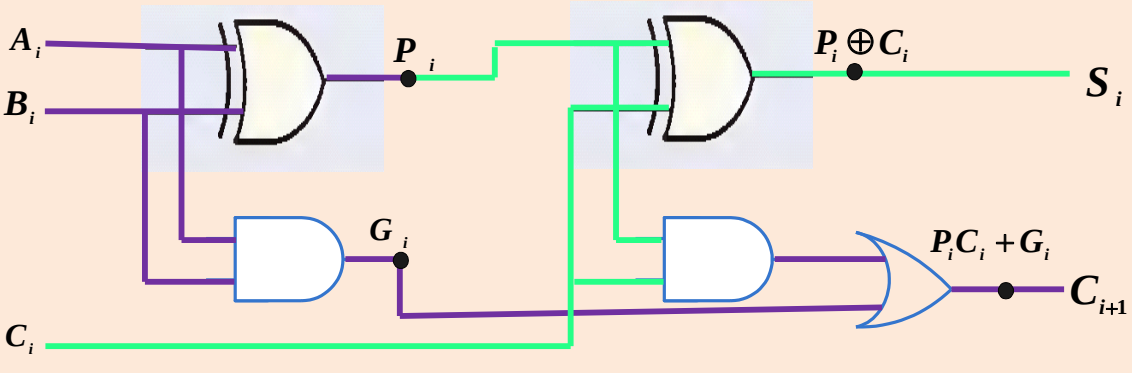
$A + B$ $(A = 1011)$ and $(B = 0011)$

Subscript i	3	2	1	0	
Input Carry	0	1	1	0	C_i
A	1	0	1	1	A_i
+					
B	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output Carry	0	0	1	1	C_{i+1}

$C_0 = 0$

Carry Propagation

- The addition of $A + B$ binary numbers in ***parallel*** implies that all the bits of A and B are available for computation at the same time.
- As in any combinational circuit, the signal must ***propagate*** through the gates before the correct output sum is available.
- The output will not be correct unless the signals are given enough time to propagate through the gates connected form the input to the output.
- The longest ***propagation delay time*** in an adder is the time it takes the carry to propagate through the full adders.



Full Adder with P and G

- The signal form the carry input C_i to the output carry C_{i+1} propagates through an ***AND*** gate and an ***OR*** gate, which equals ***2 gate levels***.
  - If there are ***4*** full adders in the binary adder, the output carry C_4 would have ***2×4=8 gate levels***, form C_0 to C_4
  - For an n-bit adder, $2n$ gate levels for the carry to propagate form input to output are required.

- The ***carry propagation time*** is an important attribute of the adder because it limits the speed with which two numbers are added.
- To reduce the carry propagation delay time:
  1) Employ faster gates with reduced delays.
  2) Employ the principle of Carry Lookahead Logic.

Proof: ***(using carry lookahead logic)***

$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$

The output sum and carry are:

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_i C_i$$

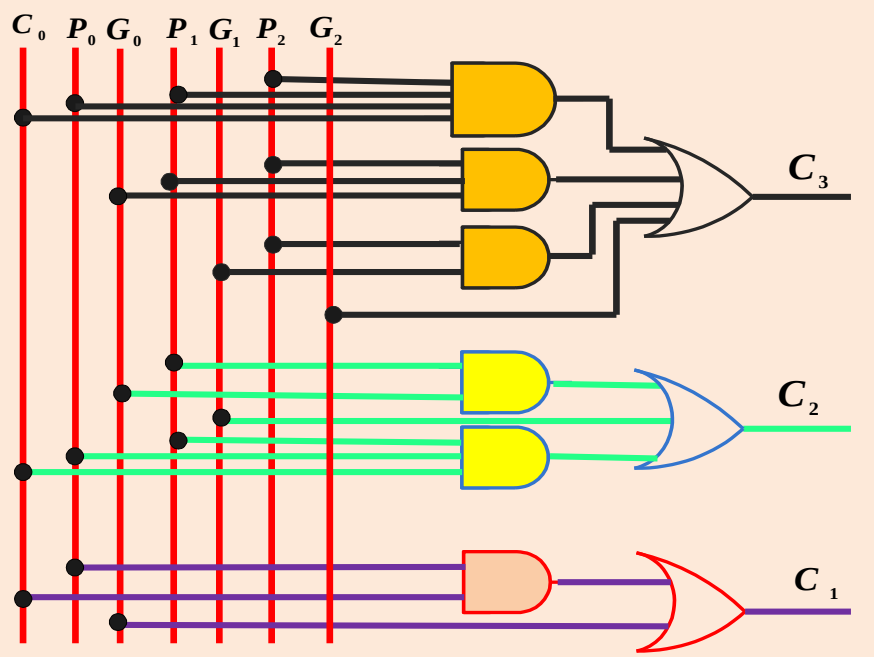
- ✓ G_i-called a ***carry generate***, and it produces a carry of 1 when both A_i and B_i are 1.
- ✓ P_i-called a ***carry propagate***, it determines whether a carry into stage i will propagate into stage $i + 1$.
- ✓ The ***Boolean function*** for the carry outputs of each stage and substitute the value of each C_i form the previous equations:

$$\left\{\begin{array}{l} C_0 = input\ carry \\ C_1 = G_0 + P_0 C_0 \\ C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0) \\ \quad = G_1 + P_1 G_0 + P_1 P_0 C_0 \\ C_3 = G_2 + P_2 C_2 = G_2 + P_2(G_1 + P_1 G_0 + P_1 P_0 C_0) \\ \quad = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0 \end{array}\right\}$$

- The three Boolean functions C_1, C_2 and C_3 are implemented in the ***carry lookahead generator.***

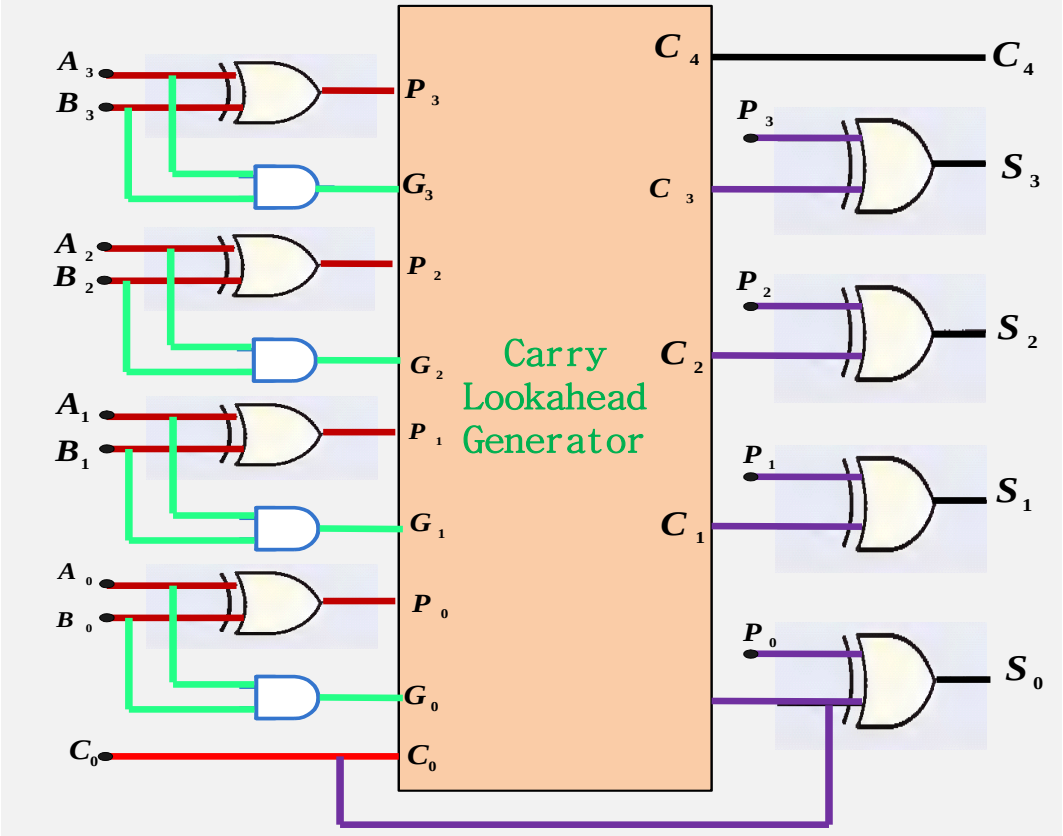
> ***The two level-circuit for the output carry C_4 is not shown, it can be easily derived by the equation.***

- C_3 does not have to wait for C_2 and C_1 to propagate, in fact C_3 is propagated at the same time as C_1 and C_2.



Logic Diagram for Carry Lookahead Generator

- The construction of a ***four-bit adder with a carry lookahead scheme*** is the following:



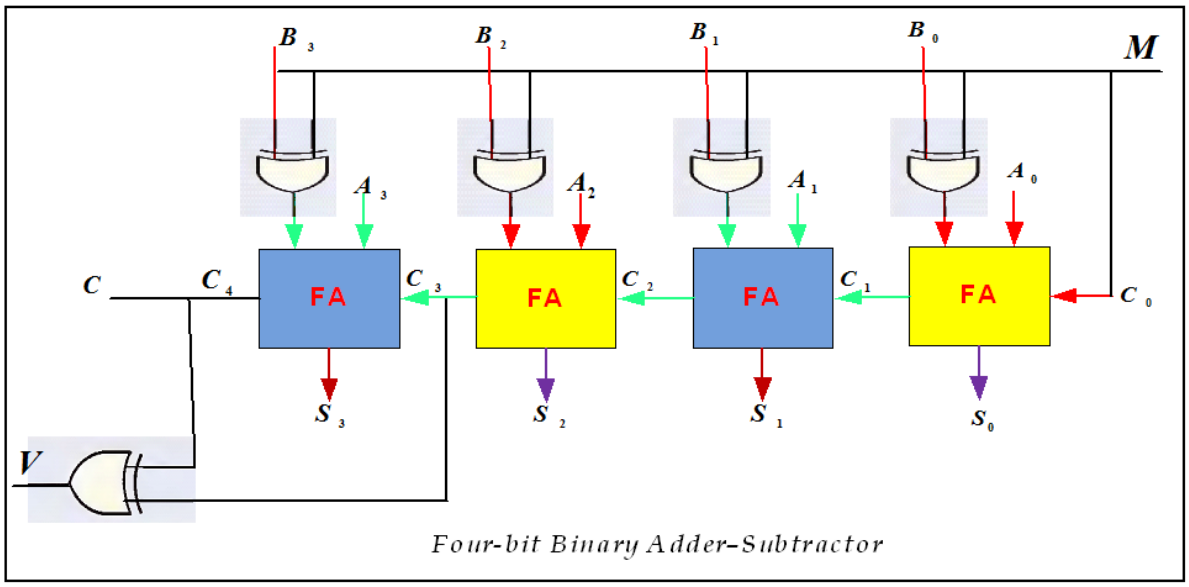
four-bit adder with a carry lookahead scheme

4. Binary Subtractor

- To perform the subtraction $-B$, we can use the ***2's complements***, so the subtraction can be converted to addition.
- ***2's complement*** can be obtained by talking the *1's complement* and adding *1* to the *LSD* bit.
    1) *1's complement* can be implemented with inventors.
    2) *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*.

5. Binary Adder–Subtractor

- The addition and subtraction operations can be combined into one circuit with one common binary adder by including an *exclusive-OR* gate with each full-adder.



Four-bit Binary Adder-Subtractor

The mode input M controls the operation as the following:

- ($M = 0$ →adder.
- $M = 1$ → subtractor.

- Each *XOR* gate receives M signal and B
    - When $M = 0$ then $B \oplus 0 = B$ and the carry = **0,** then the circuit performs the operation $A + B$**.**
    - When $M = 1$ then $B \oplus 1 = \bar{B}$ and the carry = **1,** then the circuit performs the operation $A - B$**.**
- The *exclusive-OR* with output V is for detecting an overflow.