UNIT-II.

Addition and Subtraction of Signed Numbers - Problem Solving - Design of fast adders - Ripple carry adder and Carry look ahead adder - Multiplication of positive numbers - Problem Solving - Signed Operand Multiplication - Problem Solving - Fast multiplication Bit pair recoding of Multipliers - Problem Solving - Carry Save Addition of Summands - Integer division - Restoring Division - Non-Restoring Division - Floating point numbers and operations.

—x—

NUMBER REPRESENTATION:

NUMBER SYSTEM - THE BASICS.

* Almost all modern computers are digital computers, which means that they can recognize only 2 distinct electronic states. These states are identified as 0 and 1, (or) false and true (or) off and on. Sequences of 0's and 1's are binary numbers.

Decimal Number System:

* Throughout the world, the main system of mathematical notation is decimal number system, also called base-10.

* The term base refers to the number of distinct symbols that can be found. In decimal system, there are ten symbols, called digits which are universally 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.

* Every digit position has a weight which is a power of 10.

Example:

1) $234 = 2 \times 10^2 + 3 \times 10^1 + 4 \times 10^0 = 2 \times 100 + 3 \times 10 + 4 \times 1$
$= 200 + 30 + 4 = 234$.

2) $250.67 = 2 \times 10^2 + 5 \times 10^1 + 0 \times 10^0 + 6 \times 10^{-1} + 7 \times 10^{-2}$

Scanned with CamScanner

Binary Number System :

* It has only 2 digits - 0 and 1. It is of base 2.
* Every digit position has a weight which is power of 2.

Example :

1) $110 = 1\times2^2 + 1\times2^1 + 0\times2^0$

2) $101.01 = 1\times2^2 + 0\times2^1 + 1\times2^0 + 0\times2^{-1} + 1\times2^{-2}$

Positional Number Systems :

(i) Decimal Numbers ,

* 10 Symbols $\{0,1,2,3,4,5,6,7,8,9\}$, Base or radix is 10..
* $136.25 = 1\times10^2 + 3\times10^1 + 6\times10^0 + 2\times10^{-1} + 5\times10^{-2}$

(ii) Binary Numbers :

* 2 symbols $\{0,1\}$. Base or radius is 2.
* $101.01 = 1\times2^2 + 0\times2^1 + 1\times2^0 + 0\times2^{-1} + 1\times2^{-2}$

(iii) Octal Numbers :

* 8 symbols $\{0,1,2,3,4,5,6,7\}$. Base or radius is 8.
* $621.03 = 6\times8^2 + 2\times8^1 + 1\times8^0 + 0\times8^{-1} + 3\times8^{-2}$

(iv) Hexadecimal Numbers :

* 16 Symbols $\{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$. Base is 16.
* $6AF.3C = 6\times16^2 + 10\times16^1 + 15\times16^0 + 3\times16^{-1} + 12\times16^{-2}$.

Number Conversions :

↳ Binary to Decimal Conversion
↳ Decimal to Binary Conversion

Binary to Decimal Conversion :

* A binary number :

$B = b_{n-1}\, b_{n-2} \cdots b_1 b_0 . b_{-1}\, b_{-2} \cdots b_{-m}$

Scanned with CamScanner

* The corresponding value in decimal :

$$D = \sum_{i=-m}^{n-1} b_i 2^i$$

Examples :

Eg1 : 101011

↳ $1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

↳ $32 + 0 + 8 + 0 + 2 + 1$

↳ 43.

$(101011)_2 = (43)_{10}$

Eg 2: $.0101 \rightarrow 0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 1 \times 2^{-4}$

$\rightarrow 0 + 1 \times \frac{1}{2^2} + 0 + 1 \times \frac{1}{2^4}$

$\rightarrow 0 + \frac{1}{4} + 0 + \frac{1}{16} \rightarrow \frac{4+1}{16} \rightarrow \frac{5}{16} \rightarrow 0.3125$

$(.0101)_2 = (0.3125)_{10}$

Eg 3: $101.11 \rightarrow 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$

$\rightarrow 4 + 0 + 1 + \frac{1}{2} + \frac{1}{4}$

$\rightarrow 5 + \frac{1}{2} + \frac{1}{4}$

$\rightarrow 5.75$

$(101.11)_2 = (5.75)_{10}$

Decimal to Binary : – Integer Part
 \ Fraction Part

Integer Part :

* Repeatedly divide the given number by 2 and go on accumulating the remainders, until the number becomes zero.

* Arrange the remainders in reverse order.

Eg 1: 89

2 | 89
2 | 44 - 1
2 | 22 - 0
2 | 11 - 0
2 | 5 - 1
2 | 2 - 1
1 - 0

$(89)_{10} = (1011001)_2$

Eg 2: 239

2 | 239
2 | 119 - 1
2 | 58 - 1
2 | 29 - 1
2 | 14 - 1
2 | 7 - 0
2 | 3 - 1
1 - 1

$(239)_{10} = (11101111)_2$

Fraction Part:

* Repeatedly multiply the given fraction by 2.
  - Accumulate the integer part (0 or 1).
  - If the integer part is 1, chop it off.
* Arrange the integer parts in the order obtained.

Eg 1: 0.634

$.634 \times 2 = 1.268$

$.268 \times 2 = 0.536$

$.536 \times 2 = 1.072$

$.072 \times 2 = 0.144$

$.144 \times 2 = 0.288$

⋮

$(0.634)_{10} = (.10100....)_2$

Eg 2: 0.0625

$.0625 \times 2 = 0.125$

$.125 \times 2 = 0.25$

$.25 \times 2 = 0.5$

$0.5 \times 2 = 1.0$

$(.0625)_{10} = (.0001)_2$

Example: $(37.0625)_{10}$

$(37)_{10} = (100101)_2$

$(.0625)_{10} = (.0001)_2$

$(37.0625)_{10} = (100101.0001)_2$

Scanned with CamScanner

NUMBER REPRESENTATION - SIGNED NUMBER:

* There is a need to represent both positive and negative numbers. Three systems are used for such representation.

→ Sign and magnitude
→ 1's complement
→ 2's complement

* In all 3 systems, leftmost bit is 0 for positive numbers and 1 for negative numbers.

* In all 3 systems, positive values have identical representation, but negative values have different representation.

Binary $b_3 b_2 b_1 b_0$	Sign and magnitude	1's complement	2's complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	−0	−7	−8
1 0 0 1	−1	−6	−7
1 0 1 0	−2	−5	−6
1 0 1 1	−3	−4	−5
1 1 0 0	−4	−3	−4
1 1 0 1	−5	−2	−3
1 1 1 0	−6	−1	−2
1 1 1 1	−7	−0	−1

* Sign and magnitude System:

* In this, negative values are represented by changing the most significant bit from 0 to 1 of the corresponding positive value.

Scanned with CamScanner

eg. +5 is represented by 0101 and -5 by 1101.

* 1's complement :

* In this, negative values are obtained by complementing (0 to 1 (or) 1 to 0) each bit of the corresponding positive number.

eg. +3 is 0011, The complement of 0011 is 1100 that represents -3.

* 2's complement :

* In this, 2's complement of a number is obtained by adding 1 to the ~~1st~~ 1's complement of that number.

Addition of Positive Numbers :

* Consider adding two 1-bit numbers.

$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 1 \\ +\ 1 \\ \hline 10 \end{array}$$

↑ Carryout

* Sum of 1 and 1 requires 2-bit vector 10 to represent value 2.

* Bit pairs are added starting from the low-order (right) end of bit vectors, propagating carries toward high-order (left) end.

Problem :

Consider the decimal value 108. Represent the value in unsigned and signed binary representation.

Unsigned : Decimal to Binary Conversion.

```
2 | 108
2 | 54 - 0
2 | 27 - 0
2 | 13 - 1
2 | 6 - 1
2 | 3 - 0
    1 - 1
```

$(108)_{10} = (1101100)_2$

Signed Representation : +108 and −108 ·

+108 = $\boxed{0}$1101100
sign bit

Negative value : (−108)
Sign and magnitude ⇒ 11101100
1's complement form ⇒ 1's complement of +ve number (01101100)
↳ 10010011

2's complement form ⇒ 1's complement + 1
⇒ 10010011 + 1
⇒ 10010100

ADDITION AND SUBTRACTION OF SIGNED NUMBERS.

* There are three systems to represent signed numbers - Sign and magnitude, 1's complement and 2's complement.

* The 2's compliment method is the most efficient system for performing arithmetic addition and subtraction.

Rules governing addition and subtraction of n-bit signed numbers using 2'complement representation system.

1. To add two numbers, add their n-bit representations, ignoring the carry out signal from the most significant bit position. The sum will be algebraically correct value as long as the answer is in the range -2^{n-1} through $+2^{n-1}-1$.

2. To subtract 2 numbers X and Y, form 2's complement of Y and then add it to X. Again, the result will be correct value as long as the answer is in range -2^{n-1} through $+2^{n-1}-1$.

* Some examples for addition and subtraction. Examples take 4-bit, the answers fall into the representable range of −8 through +7. When answers do not fall within the representable range, then arithmetic overflow has occurred.

Scanned with CamScanner

(a) $(+2)+(+3)$

$$\begin{array}{rrr} & 0010 & (+2) \\ (+) & 0011 & (+3) \\ \hline & 0101 & (+5) \end{array}$$

(b) $(+4)+(-6)$

$$\begin{array}{rrr} & 0100 & (+4) \\ (+) & 1010 & (-6) \\ \hline & 1110 & (-2) \end{array}$$

(c) $(-5)+(-2)$

$$\begin{array}{rrr} & 1011 & (-5) \\ (+) & 1110 & (-2) \\ \hline \text{ignore carry} \leftarrow \boxed{1} & \cancel{1}001 & -7 \end{array}$$

(d) $(+7)+(-3)$.

$$\begin{array}{rrr} & 0111 & (+7) \\ (+) & 1101 & (-3) \\ \hline & 0100 & (+4). \end{array}$$

(e) $(-3)-(-7)$

$$\begin{array}{rrr} & 1101 & (-3) \\ (-) & 1001 & (-7) \\ \hline \end{array}$$

$\Rightarrow$ Take 2'comp. of (-7)

2's comp = 1's comp +1

1's comp = 0110

$$\begin{array}{rr} & 0110 \\ + & 1 \\ \hline \end{array}$$

2's compl = 0111

$$\begin{array}{rrr} & 1101 & \\ (+) & 0111 & \\ \hline & 0100 & (+4) \end{array}$$

(f) $(+2)-(+4)$

$$\begin{array}{rr} & 0010 \\ (-) & 0100 \\ \hline \end{array} \Rightarrow \begin{array}{rrr} & 0010 & \\ (+) & 1100 & \\ \hline & 1110 & (-2) \end{array}$$

$$\begin{array}{r} 1011 \\ 1 \\ \hline 1100 \end{array}$$

(g) $(+6)-(+3)$

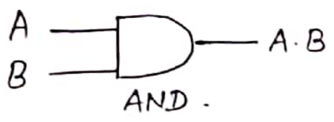
$$\begin{array}{rr} & 0110 \\ (-) & 0011 \\ \hline \end{array} \Rightarrow \begin{array}{rrr} & 0110 & \\ (+) & 1101 & \\ \hline & 0011 & (+3). \end{array}$$

$$\begin{array}{r} 1100 \\ 1 \\ \hline 1101 \end{array}$$

LOGIC GATES.

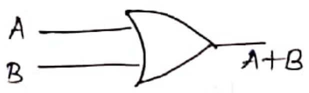
* Digital systems are constructed by logic gates. These gates are the AND, OR, NOT, NAND, NOR, EXOR and EXNOR gates.

AND gate : → Output is high (1) only if all its input are high.



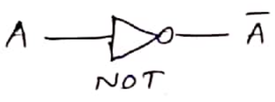
A	B	A·B
0	0	0
0	1	0
1	0	0
1	1	1

OR gate : → output is high (1) only if all its inputs are high.



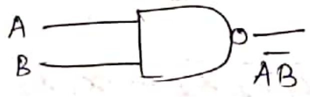
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT gate : → output is the inversion of its input.



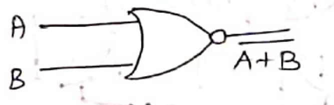
A	$\bar{A}$
0	1
1	0

NAND gate : → This is a NOT-AND gate. AND gate followed by NOT gate.
→ output is high (1) if any of the inputs are low.



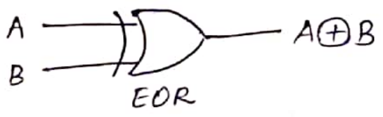
A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

NOR gate : → This is a NOT-OR gate. OR gate followed by NOT gate.
→ output is low (0) if any of the inputs are high.



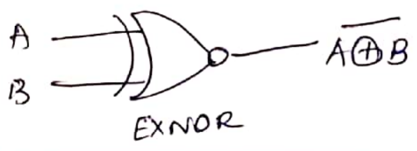
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

EXOR gate : → The Exclusive-OR gate is a circuit which will give a high output if either (not both) of its inputs are high.



A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

EXNOR gate → The Exclusive-NOR gate does opposite of EXOR gate. It will give a low output if either (but not both) of its input are high.



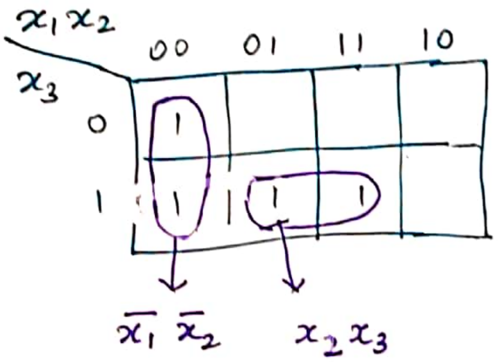
A	B	$\overline{A\oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

Scanned with CamScanner

KARNAUGH MAP:

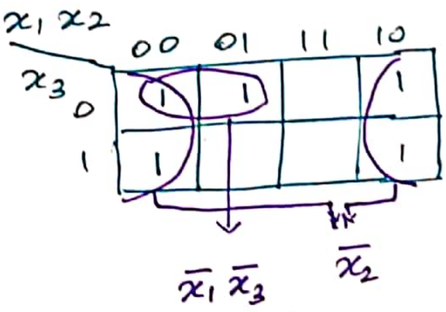
* K-Map is used for Minimization of Boolean expression.
* For a three-variable function, the map is rectangle composed of eight squares arranged in two rows of 4 squares each.
* When two adjacent squares contain 1s, they indicate possibility of an algebraic simplification.

eg. $f_1 = x_1x_2x_3 + \bar{x}_1x_2x_3 + \bar{x}_1\bar{x}_2x_3 + \bar{x}_1\bar{x}_2\bar{x}_3$



$f_1 = \bar{x}_1\bar{x}_2 + x_2x_3$

eg2. $f_2 = \bar{x}_1\bar{x}_2\bar{x}_3 + \bar{x}_1x_2\bar{x}_3 + x_1\bar{x}_2\bar{x}_3 + \bar{x}_1\bar{x}_2x_3 + x_1\bar{x}_2x_3$

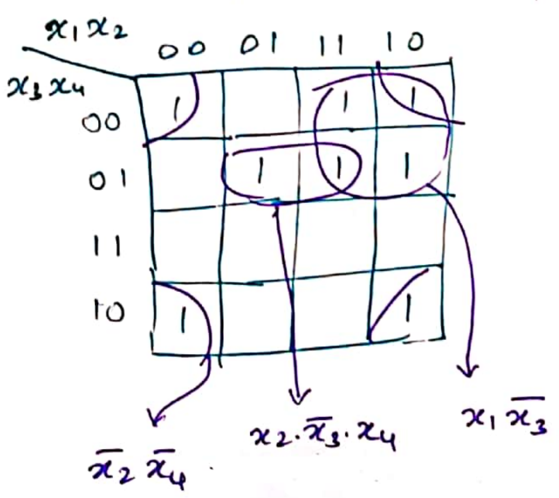


$f_2 = \bar{x}_2 + \bar{x}_1\bar{x}_3$

* K-map can also be constructed for more than 3 variables.

4-variable map.

$f = \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_4 + x_1x_2\bar{x}_3\bar{x}_4 + x_1\bar{x}_2\bar{x}_3\bar{x}_4 + \bar{x}_1x_2\bar{x}_3x_4 + x_1x_2\bar{x}_3x_4 + x_1\bar{x}_2\bar{x}_3x_4 + \bar{x}_1\bar{x}_2x_3\bar{x}_4 + x_1\bar{x}_2x_3\bar{x}_4$

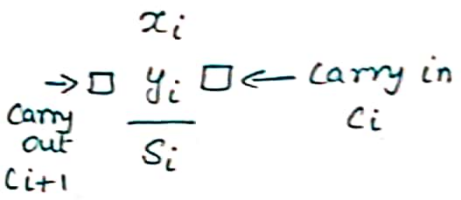


$f = x_1\bar{x}_3 + \bar{x}_2\bar{x}_4 + x_2\bar{x}_3x_4$

Scanned with CamScanner

DESIGN OF FAST ADDERS:

$$\begin{array}{rcrcr} X & & 7 & & 0\;1\;1\;1 \\ +\,Y & = & +6 & = & +_0\;0_1\;1_1\;1_1\;0_0\;{}^0 \\ \hline Z & & 13 & & 1\;1\;0\;1 \end{array}$$



TRUTH TABLE:

x_i	y_i	carry-in, c_i	Sum, S_i	carryout, c_{i+1}
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 figure shows the logic truth table for the sum and carry-out functions for adding equally weighted bits x_i and y_i in two numbers X and Y.

* Each stage of addition process must accomodate a carry-in bit represented as c_i in ith stage, which is the carry out from $(i-1)$st stage.

Logic Expressions:

$$S_i = \bar{x}_i \bar{y}_i c_i + \bar{x}_i y_i \bar{c}_i + x_i \bar{y}_i \bar{c}_i + x_i y_i c_i$$

x_i \ y_ic_i	00	01	11	10
0		1		1
1	1		1	

$$= \bar{x}_i(\bar{y}_i c_i + y_i \bar{c}_i) + x_i(\bar{y}_i \bar{c}_i + y_i c_i)$$

$$= \bar{x}_i(y_i \oplus c_i) + x_i(\overline{y_i \oplus c_i})$$

$$\boxed{S_i = x_i \oplus y_i \oplus c_i}$$

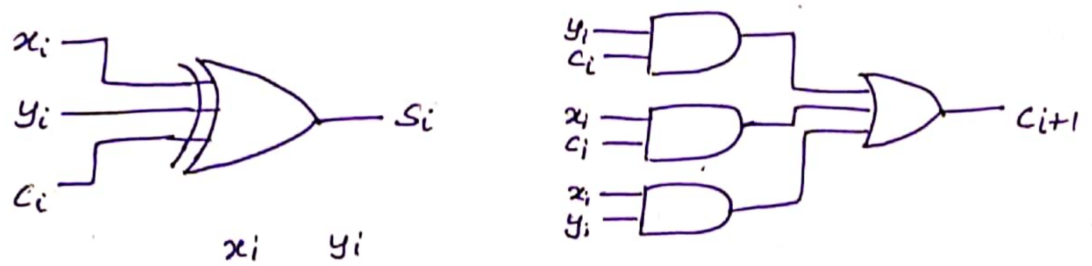
$$C_i = \bar{x}_i y_i c_i + x_i \bar{y}_i c_i + x_i y_i \bar{c}_i + x_i y_i c_i$$

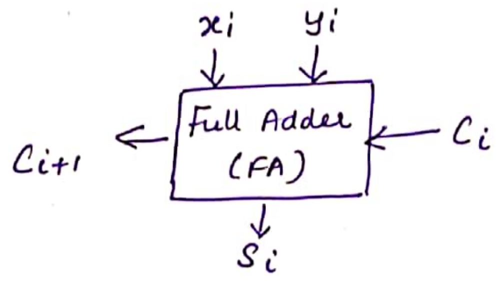
x_i \ y_ic_i	00	01	11	10
0			1	
1		1	1	1

$$\boxed{C_i = y_i c_i + x_i c_i + x_i y_i}$$

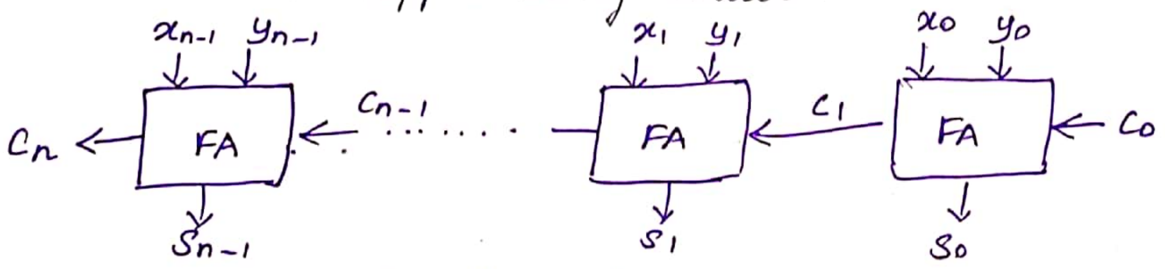
* The logic expression for S_i can be implemented with a 3-input X-OR gate. The carry-out function C_{i+1}, can be implemented with a two level AND-OR logic circuit.

* A convenient symbol for the complete circuit for a single stage of addition is called a Full Adder (FA).



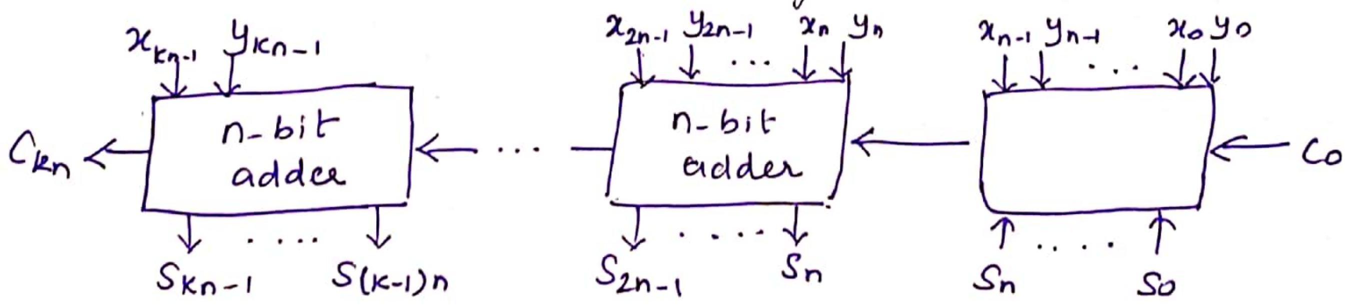


* A cascaded connection of n full adder blocks, can be used to add two n-bit numbers. Since the carries must propagate or ripple through this cascade, the configuration is called an n-bit ripple carry adder.



n-bit ripple carry adder

* The carry signals are also useful for interconnecting k adder to form an adder capable of handling input numbers that are kn bits long.

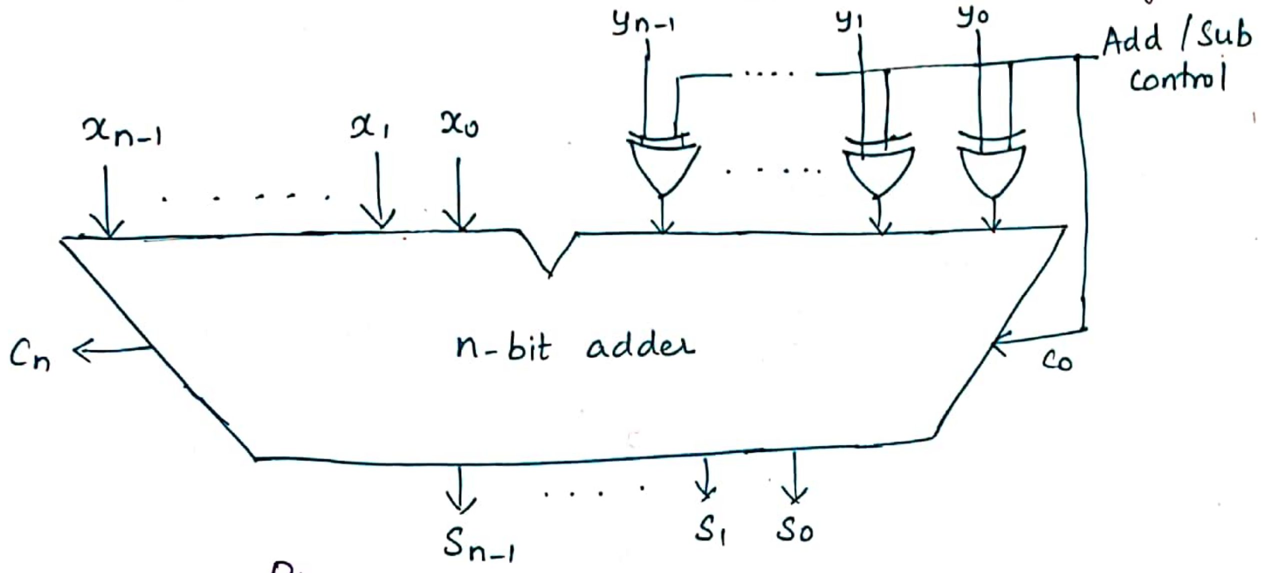


x_{kn-1} y_{kn-1}
x_{2n-1} y_{2n-1} ... x_n y_n
x_{n-1} y_{n-1} ... x_0 y_0
C_{kn}
n-bit adder
n-bit adder
C_0
S_{kn-1} .... $S_{(k-1)n}$
S_{2n-1} .... S_n
S_n .... S_0

Cascade of k n-bit adders

* Overflow can only occur when the signs of the 2 operands are the same. In this case, overflow occurs if the sign of the result is different.

* A circuit for detecting overflow can be obtained by implementing the expression $C_n \oplus C_{n-1}$ with an XOR gate.



Binary addition /Subtraction logic network

* The logic circuit can be used to perform either addition or subtraction based on the value applied to Add/Sub control line.

* This line is set to 0 for addition, applying Y vector unchanged to one of the adder inputs along with a carry-in signal, C_0, of 0.

* When the Add/Sub control line is set to 1, then Y vector is 1's complemented by XOR gates and C_0 is set to 1 to complete the 2's complementation of Y.

CARRY LOOK AHEAD ADDER:

Motivation behind carry look ahead adder

* In ripple carry adders, for each adder block, the two bits that are to be added are available instantly. However, each adder block waits for the carry to arrive from its previous block.

Scanned with CamScanner

* So, it is not possible to generate the sum and carry of any block until the input carry is known.

* The ith block waits for the (i-1) block to produce its carry. So, there will be considerable time delay which is carry propagation delay.

* Hence, to reduce the carry propogation delay, carry look ahead adder is designed.

* The basic idea behind carry look ahead adder is to calculate carry at each stage with the available inputs x_i, y_i and C_0.

* The logic expressions for S_i (Sum) and C_{i+1} (carry out) of stag i is $S_i = x_i \oplus y_i \oplus c_i$ and $C_{i+1} = x_i y_i + x_i c_i + y_i c_i$

$$C_{i+1} = x_i y_i + c_i (x_i + y_i)$$
$$= x_i y_i + (x_i + y_i) c_i$$
$$= G_i + P_i \cdot C_i .$$

$$\underset{\text{carry generate function}}{G_i = x_i y_i} \quad \text{and} \quad P_i = x_i + y_i \longrightarrow \text{carry propagate function}$$

From truth table,

x_i	y_i	c_i	C_{i+1}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

* $C_{i+1} = 1$ if both x_i and y_i is 1, irrespective of C_i

* $C_{i+1} = 1$, if either x_i or y_i is 1 and $C_i = 1$.

$(\bar{x}_i y_i + x_i \bar{y}_i) C_i = (x_i \oplus y_i) C_i$

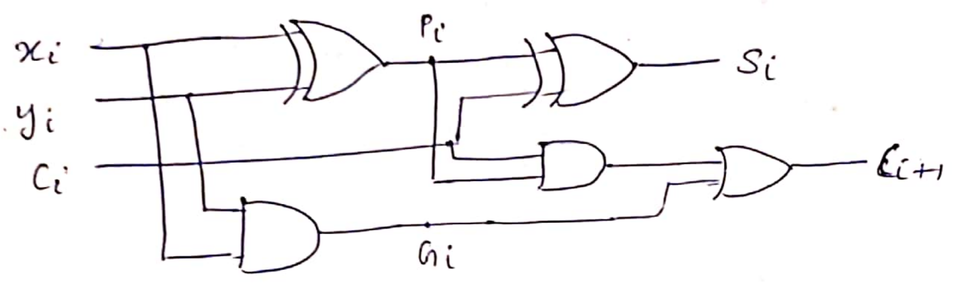
$\therefore C_{i+1} = x_i y_i + (x_i \oplus y_i) C_i$

$G_i = x_i y_i \quad P_i = x_i \oplus y_i$

* All G_i and P_i functions can be formed independently and in parallel in one logic gate delay after X and Y vectors are applied to the inputs of an n-bit adder.

Scanned with CamScanner

* Each bit stage contains an AND gate to calculate G_i and XOR gate to calulate P_i. A cascade of two 2-input XOR gates are used instead of 3 input XOR.



The expression to find C_{i+1},

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

Sub, $i=0$.

$$C_1 = G_0 + P_0 C_0$$

Sub $i = 1$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_0) = G_1 + G_0 P_1 + P_1 P_0 C_0.$$

Sub $i = 2$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 (G_1 + G_0 P_1 + P_1 P_0 C_0)$$
$$= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0.$$

Sub $i=3$,

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0)$$
$$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0.$$

* Continuing this expansion, the final expression for any carry variable is,

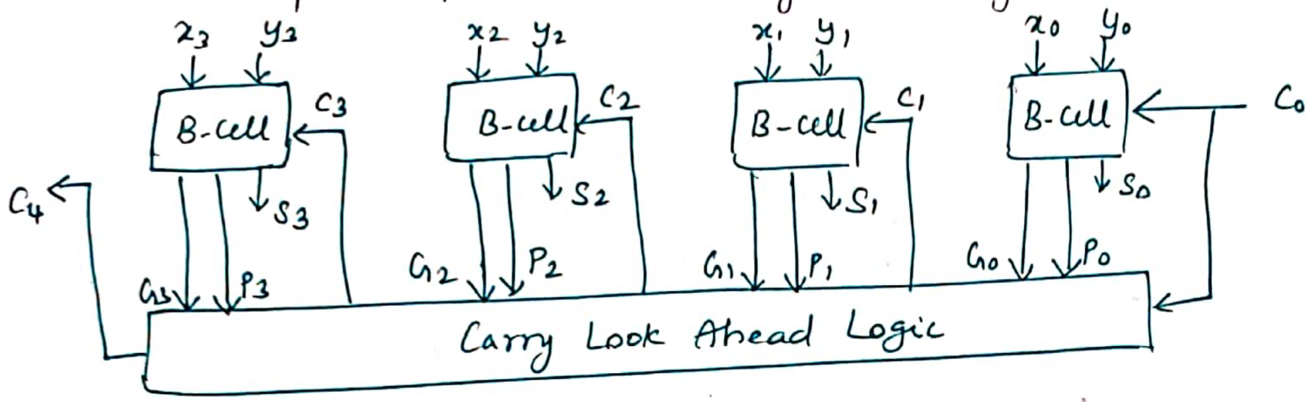
$$C_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + \cdots + P_i P_{i-1} \cdots P_1 G_0 + P_i P_{i-1} \cdots P_0 C_0$$

* Thus, all carries can be obtained three gate delays after the input signals x, y and C_0 are applied because only one gate delay is needed to develop all P_i and G_i signals, followed by 2 gate delays in AND-OR circuit for C_{i+1}. After a further XOR gate delay, all sum bits are available.

* Therefore, independent of n, n-bit addition requires only 4 gate delays.

Scanned with CamScanner

* The complete 4-bit adder is given in figure



* The carries are implemented in the block labeled carry look ahead logic. An adder implemented in this form is called carry look ahead adder.

* Delay through the adder is 3 gate delays for all carry bits and 4 gate delays for all sum bits.

Scanned with CamScanner