Excess-3 to BCD Code Converter

I/P Decimal No.	Excess-3 W X Y Z	BCD A B C D	O/P Decimal Digits
3	0 0 1 1	0 0 0 0	3-3=0
4	0 1 0 0	0 0 0 1	4-3=1
5	0 1 0 1	0 0 1 0	5-3=2
6	0 1 1 0	0 0 1 1	6-3=3
7	0 1 1 1	0 1 0 0	7-3=4
8	1 0 0 0	0 1 0 1	8-3=5
9	1 0 0 1	0 1 1 0	9-3=6
10	1 0 1 0	0 1 1 1	10-3=7
11	1 0 1 1	1 0 0 0	11-3=8
12	1 1 0 0	1 0 0 1	12-3=9

Excess-3 code minus (−) 3 = BCD code

Again here also Excess-3 code is of 4-bit number, so it has to start from 0000 to 1111. But for Excess-3 code 0000, 0001, 0010, 1101, 1110, 1111 we get undefined, so for these numbers, output BCD code is don't care conditions.

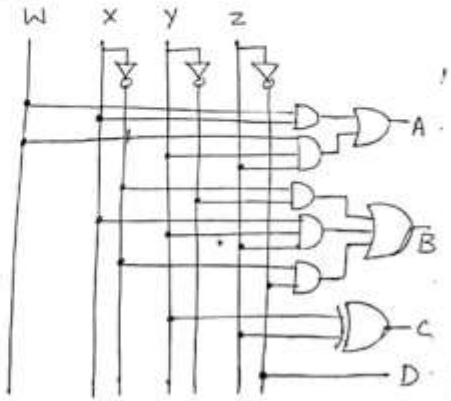
$\therefore A = WX + WYZ$

$B = \overline{X}\,\overline{Y} + XYZ + \overline{X}\,\overline{Z}$

$C = Y\overline{Z} + \overline{Y}Z = Y \oplus Z$

$D = \overline{Z}$!

These equations got after mapping A, B, C, D in K-maps.



Binary to Gray Code Converter.

Procedure to convert given binary (4 bit) number to its equivalent 4-bit Gray code number.

1) The MSB bit of the gray code will be exactly equal to the MSB bit of the given binary number.
2) The second bit of the gray code will be exclusive-OR of the MSB bit & second bit of binary number.
3) The third bit of gray code will be the exclusive-OR of the second bit and third bit of binary number and goes on.

for example: Binary number : $B_3\ B_2\ B_1\ B_0$

Gray code number : $G_3\ G_2\ G_1\ G_0$

$\therefore \boxed{G_3 = B_3}$; $\boxed{G_2 = B_3 \oplus B_2}$; $\boxed{G_1 = B_2 \oplus B_1}$; $\boxed{G_0 = B_1 \oplus B_0}$

$\therefore$ In general $\boxed{G_i = B_{i+1} \oplus B_i}$ where $i = 0, 1, 2, 3 \ldots$

B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
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

P.T.O

Code Converters,
Binary Adders and Subtractor

CODE CONVERTERS

When a combinational circuit has two or more outputs, each output must be expressed seperately as a function of all the input variables.

An example of a multiple-output circuit is a Code Converter, which is a circuit that translates information from one binary code to another.

The inputs to the circuit provide the bit combination of the elements as specified by the first code, and the outputs generate the corresponding bit combination of the second code.

The combinational circuits performs the transformation from one code to the other is called CODE CONVERTER

BCD to Excess-3 code Converter

Truth Table

Decimal Digit	Input BCD A	B	C	D	output Excess-3 W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1				

The excess 3 code for a decimal digit is the binary combination corresponding to the decimal digit plus 3.

for eg: excess -3 code for decimal digit 5 is the binary combination for 5+3=8, which is 1000.

Since both BCD and excess-3 use four bits to represent each decimal digit, there must be four i/p variables and four o/p variables.

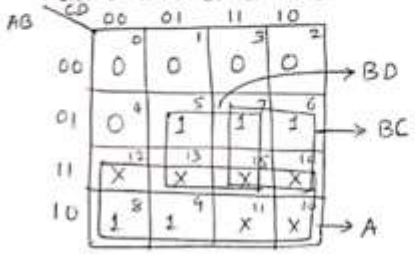
Designate the i/ps by A, B, C, D and the o/ps by W, X, Y, Z. The truth table is as shown above.

The excess -3 code word is easily obtained from a BCD code word by adding binary 0011 (3) to it

Note that four binary variables may have 16 combinations, but only 10 are listed in the truth table.

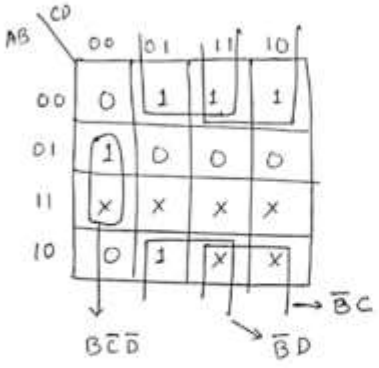
The 6 combinations 1010 through 1111 are not listed under the i/ps. These combinations have no meaning in the BCD code, we can assume that they will never occur.

Hence, it does not matter what binary values we assign to the outputs, and therefore, we can treat them as don't care conditions.

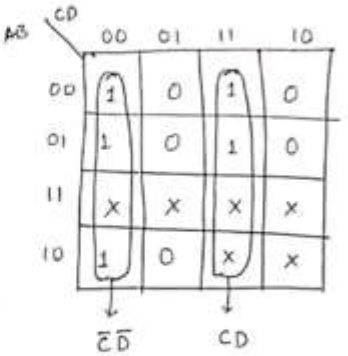


$\therefore W = A + BD + BC$

P.T.O



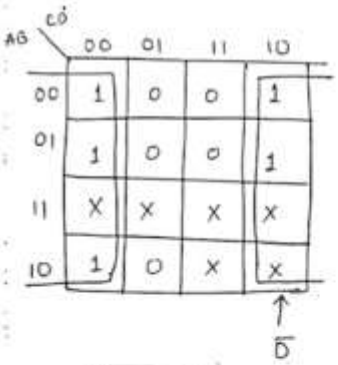
$\therefore X = \bar{B}C + \bar{B}D + B\bar{C}\bar{D}$



$\therefore Y = \bar{C}\bar{D} + CD$

$= C \odot D$

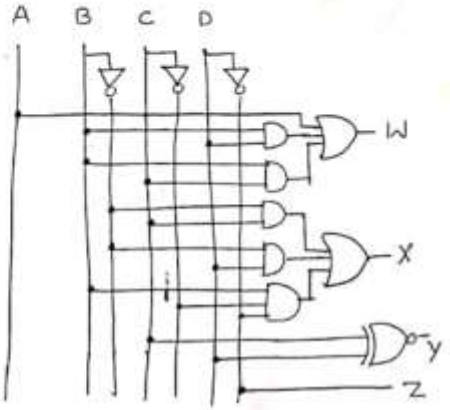
$Y = \overline{C \oplus D}$



$\therefore Z = \bar{D}$

BCD code plus(+) 3
= Excess 3 code

Realization of BCD to Excess-3 Code Converter using gates



B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_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

When we map all the o/p Gray codes into a K-map and get their equations, we get the same what we listed as above.

Gray to Binary Code Converter

Procedure to convert given gray code to its equivalent binary code number.

1) The MSB bit of the binary code will be exactly equal to the MSB bit of the given gray code number.
2) The second bit of the binary code will be exclusive-OR of the MSB bit of gray code and second bit of gray code
3) The third bit of the binary code will be exclusive-OR of the MSB bit, second bit and third bit of gray code.
4) The fourth bit of the binary code will be exclusive-OR of MSB bit, second bit, third bit & fourth bit of gray code and goes on.

P.T.O

for example: Gray code number: $G_3\ G_2\ G_1\ G_0$

Binary number: $B_3\ B_2\ B_1\ B_0$

$\therefore$ $B_3 = G_3$; $B_2 = G_3 \oplus G_2$; $B_1 = G_3 \oplus G_2 \oplus G_1$

& $B_0 = G_3 \oplus G_2 \oplus G_1 \oplus G_0$

In simple way, we can rewrite these equations as

$B_3 = G_3$; $B_2 = B_3 \oplus G_2$; $B_1 = B_2 \oplus G_1$

& $B_0 = B_1 \oplus G_0$

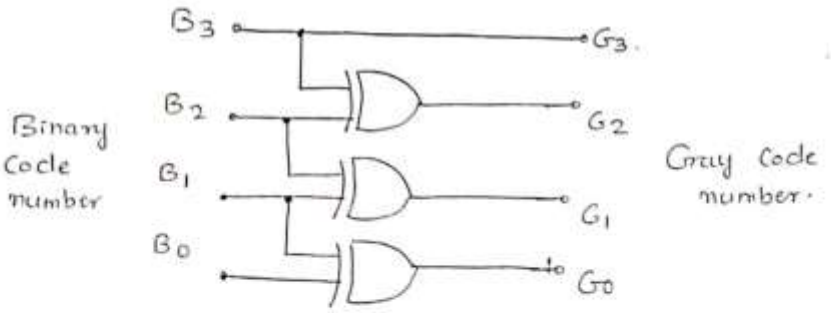
So in general $B_i = B_{i+1} \oplus G_i$ where $i = 0, 1, 2, 3 \cdots$

G_3	G_2	G_1	G_0	B_3	B_2	B_1	B_0
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	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0

Implementation of Binary to Gray code and Gray to Binary code converter using gates.

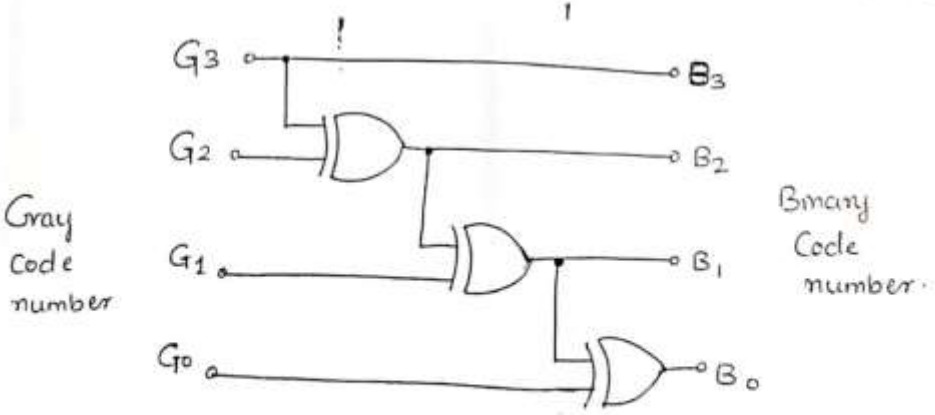
Binary to Gray Code Converter

$G_3 = B_3$; $G_2 = B_3 \oplus B_2$; $G_1 = B_2 \oplus B_1$; $G_0 = B_1 \oplus B_0$.



Gray Code to Binary Code Converter

$B_3 = G_3$; $B_2 = B_3 \oplus G_2$; $B_1 = B_2 \oplus G_1$; $B_0 = B_1 \oplus G_0$



Combinational Functions

Arithmetic Operations:

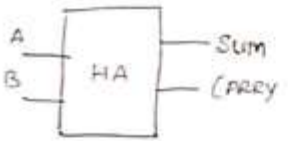
Binary Addition

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

(carry, sum)

When we add two binary numbers, we get two binary o/ps sum and carry as shown above

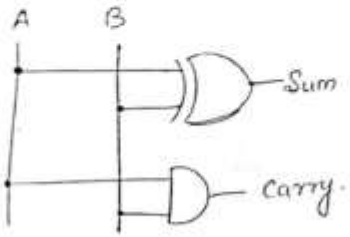
Adding two 1-bit binary numbers is called HALF ADDER



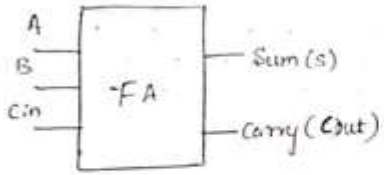
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$\therefore$ $Sum = \bar{A}B + A\bar{B} = A \oplus B$

$Carry = AB$



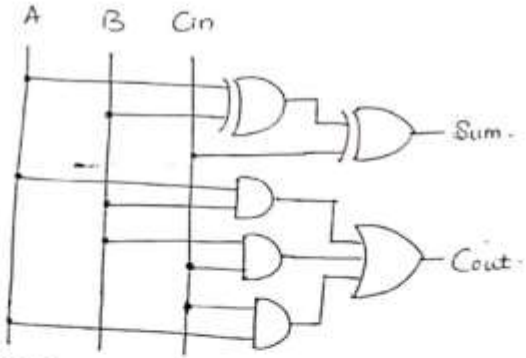
Adding 3 1-bit binary numbers is called FULL ADDER



A	B	Cin	S	Cout
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

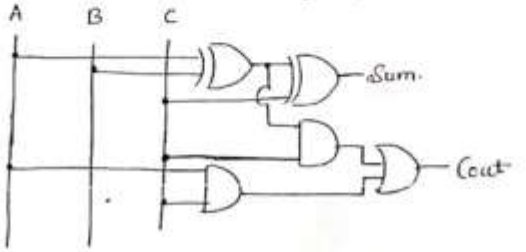
$\therefore$ $S = A \oplus B \oplus Cin$

$Cout = AB + BCin + CinA$



FULL ADDER Using 2 HA's

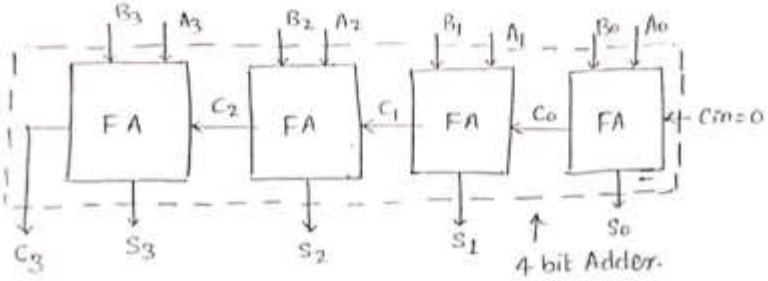
$S = A \oplus B \oplus C$; $Cout = AB + (A \oplus B)Cin$



4-Bit Binary Adder Using FA's

A = $A_3\ A_2\ A_1\ A_0$; B = $B_3\ B_2\ B_1\ B_0$; $C_{in} = 0$

C_3	C_2	C_1	C_0	C_{in}
	A_3	A_2	A_1	A_0
	B_3	B_2	B_1	B_0
C_3	S_3	S_2	S_1	S_0



4-Bit Parallel Adder using FA's

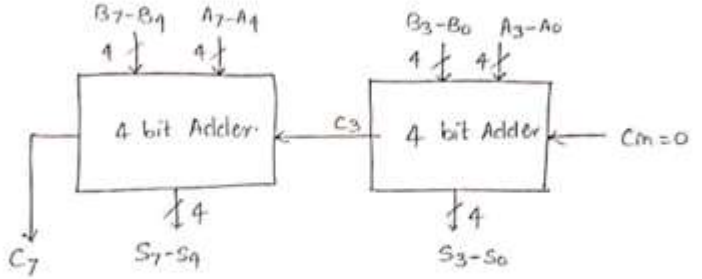
We can observe here, the FA's are cascaded to each other.

So the i^{th} stage FA will be getting carry bit as input from $(i-1)^{th}$ stage. along with A & B direct inputs.

So we can say that the correct result of addition of i^{th} stage will occur only when the correct value of its previous stage $(i-1)^{th}$ appears.

The carry is rippling through each stages and hence this parallel adder is referred to as RIPPLE ADDER

8 bit Adder using 4-bit Adders



We observed that the outputs of any full adder strongly depends on the carry out of previous stage FA. So this leads to propagation delay in the system. To overcome this we need to go for special type of adders called FAST ADDERS

Look Ahead Carry Adder (CLA or Fast Adders)

Consider the eqn of Cout

$$C_{i+1} = A_iB_i + B_iC_i + C_iA_i$$

$$\boxed{C_{i+1} = A_iB_i + (A_i + B_i)C_i}$$ → FA using only logic gates.

OR $$\boxed{C_{i+1} = A_iB_i + (A_i \oplus B_i)C_i}$$ → FA using 2 HA's. → (1)

the term (A_iB_i) relates to the carry formed at i^{th} stage and is refered to as the carry generate function

$$\boxed{G_i = A_iB_i}$$ → (2)

the term $(A_i \oplus B_i)$ or $(A_i + B_i)$ relates to the carry C_i generated at the previous stage and thus $(A_i \oplus B_i)$ or $(A_i + B_i)$ is referred to as the carry propagate function.

$$\boxed{P_i = A_i \oplus B_i}$$ OR $$\boxed{P_i = A_i + B_i}$$ → (3)

Observe that both G_i & P_i are functions of only the parallel I/ps A_i & B_i. Comparing equations (1), (2) & (3) we can write

$$C_{i+1} = G_i + P_i C_i \quad \longrightarrow (4)$$

Now let us look at the carry generated at every stage of the 4-bit parallel adder.

for $i=0$; $C_1 = G_0 + P_0 C_0$ $\longrightarrow (5)$

for $i=1$; $C_2 = G_1 + P_1 C_1$

$= G_1 + P_1 (G_0 + P_0 C_0)$

$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0$ $\longrightarrow (6)$

for $i=2$; $C_3 = G_2 + P_2 C_2$

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

$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$ $\longrightarrow (7)$

for $i=3$; $C_4 = G_3 + P_3 C_3$

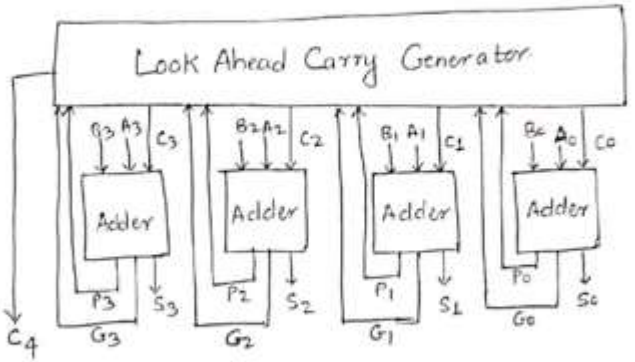
$= G_3 + P_3 [G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0]$

$C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$ $\longrightarrow (8)$

and so on.

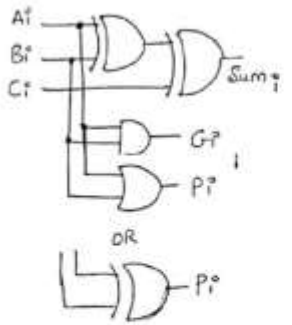
The boolean expression for C_1, C_2, C_3, C_4 etc can themselves be implemented using gates and the carry required at each stage of the parallel adder can be made available simultaneously, there by increasing the speed of addition.

P.T.O

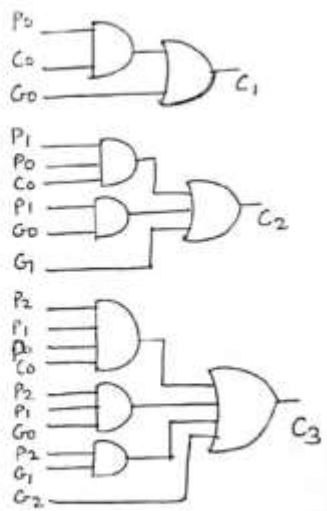


4-bit Parallel Fast Adder (CLA)

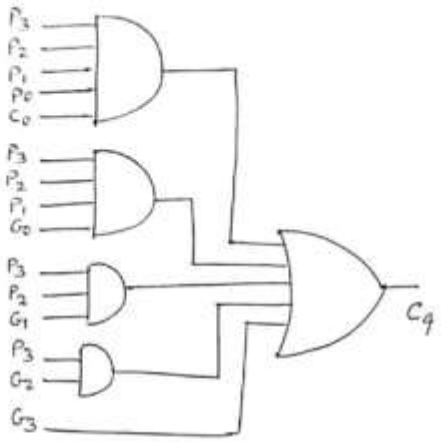
The Look Ahead Carry generator block is an implementation of eqn (5), (6), (7) & (8) where as Adder blocks are the implementation of $S_i = A \oplus B \oplus C_i$ & eqn's (2) & (3)



Adder Block Implementation



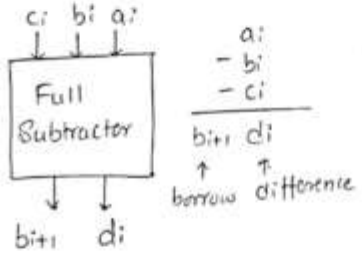
P.T.O



We can observe from these figures all carry outs are carried out only by two levels of gate structure.
∴ Much of the propagation delay is removed and it can compute faster than compared to ripple carry adder.

BINARY SUBTRACTION

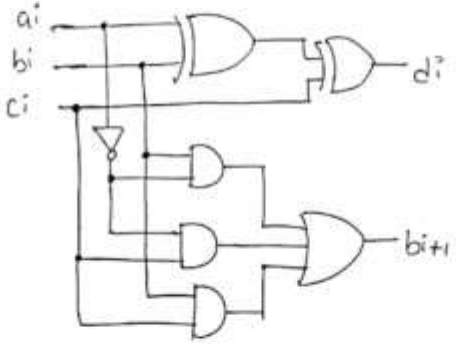
Binary subtractors can be configured on the same lines as adders. The block diagram of Full Subtractor is as shown below



Cell No	ai	bi	ci	di	bi+1
0	0	0	0	0	0
1	0	0	1	1	1
2	0	1	0	1	1
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	0
6	1	1	0	0	0
7	1	1	1	1	1

$\therefore \boxed{di = ai \oplus bi \oplus ci}$

& $\boxed{b_{i+1} = \overline{ai}\,ci + \overline{ai}\,bi + bi\,ci}$



C_0:

10
0 0 1 0 → Borrows
1 0 0 1 1 → Minuend M.
− 1 1 1 1 0 → Subtrahend N
1 0 1 0 1 → Difference.
−0 1 0 1 1 → Correct difference.

In the example, $100000 - 10101 = 01011$

2^n ↑ difference ↑ correct difference

Here n = no of bits of difference ∴ n = 5 in the example

∴ $2^5 = 32$ & its binary representation is 100000

In general, the subtraction of two n-digit numbers, M−N, in base 2 can be done as follows.

i) Subtract the subtrahend N from minuend M.

ii) If no end borrow occurs, then M>N, & the result is non negative (positive) and correct

iii) If an end borrow occurs, then N>M, and the difference N−M is subtracted from 2^n, & a minus sign is appended to the result

Subtraction of a binary number from 2^n to obtain an n-digit result is called taking 2's Complement of the number. So in Step '3', we are taking the 2's complement of the difference N−M. Use of 2's complement in subtraction is illustrated by the following example.

Example: Perform binary subtraction 01100100 − 10010110

```
  0 1 1 0 0 1 0 0
- 1 0 0 1 0 1 1 0
-----------------
  1 1 0 0 1 1 1 0  → difference
```

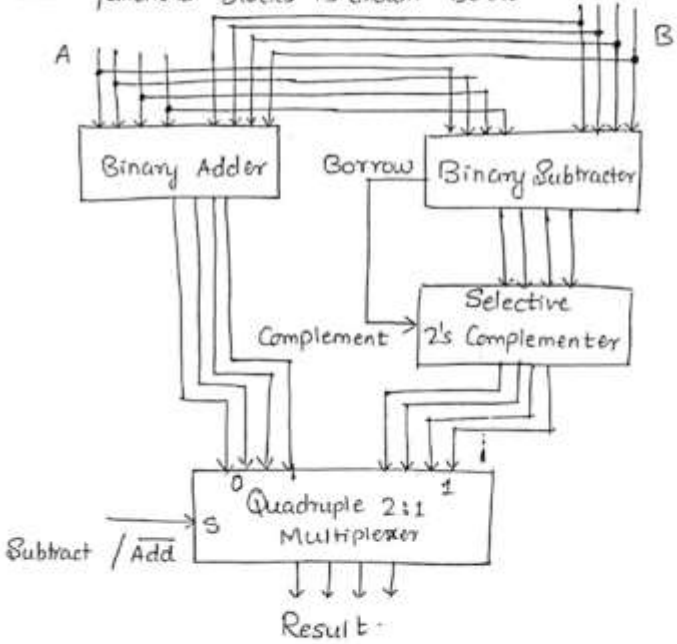
Taking 2's complement of result

```
  0 0 1 1 0 0 0 1   ← 1's complement of result
+               1   Add 1
-----------------
-(0 0 1 1 0 0 1 0)  ← Correct difference
```

To perform subtraction using this method required a Subtractor for the initial subtraction. In addition, when necessary, either the subtractor must be used a second time to perform the correction, or a seperate 2's Complementer circuit must be provided.

So, thus far, we require a subtractor, an adder, and possibly a 2's complementer to perform both addition and subtraction.

The block diagram for a 4-bit adder-subtractor using these functional blocks is shown below.



Binary Adder-Subtractor

→ The i/ps are applied to both adder and subtractor, so both operations are performed in parallel.

→ If an end borrow value of '1' occurs in the subtraction, then the selective 2's complementor receives value of 1 i/p to its complement i/p. This circuit then takes the 2's complement of the o/p of subtractor.

→ If end borrow has value of '0', the selective 2's complementor passes the o/p of subtractor through unchanged.

→ If subtraction is the operation, then '1' is applied to 'S' of the multiplexer that selects the output of the complementor.

→ If addition is the operation, then a '0' is applied to 'S', thereby selecting the o/p of the adder.

As we observe, this circuit is more complex than necessary.

We know that **Subtraction = 2's Complement Addition**

So by this, we can define a binary subtraction procedure that uses addition and the corresponding complement logic without using subtractor.

The subtraction of two n-bit unsigned number, M−N, in binary can be done as follows.

i) Add the 2's complement of the subtrahend (−ve number) N to the minuend (+ve number) M. ie, **M + 2's Comp. of N**

ii) If M≥N, the sum produces an end carry, Discard the end carry, leaving result (M−N) which is correct.

iii) If M<N, the sum does not produce end carry. So perform the correction by taking 2's complement of initial result and append minus(−) sign in front to obtain the result −(M−N) correctly.

Example 1010100 − 1000011 using 2's complement
X = 1010100, Y = 1000011

```
               1111
  + X          1010100
2's comp of Y  0111101
             ---------
            (1)0010001
             ↑
          Discard carry
```

```
1000011  ← Y
0111100  ← 1's comp
+     1  ← Add 1
-------
0111101  ← 2's comp of Y (−ve no.)
```

∴ Correct ans, is **0010001**

If Y−X has to be done, ie, 1000011 − 1010100

```
  1000011  ← Y
+ 0101100  ← 2's comp of X
  -------
  1101111
```

```
1010100  ← X
0101011  ← 1's comp.
+     1  ← Add 1
-------
0101100  ← 2's comp of X
```

∴ Here no end carry is generated & the result is in −ve. ∴ to get correct ans. take 2's comp of result and append (−) minus

```
   1101111   ← initial result
   0010000   ← 1's comp of result
         1   ← Add 1
  ---------
 −(0010001)  ← 2's comp of result
```

∴ **Y−X = − 0010001** Correct ans

Remember, in subtraction using 1's complement method, end carry is added back to result instead of discarding to get the correct result.

P.T.O

So in general, in 1's complement we add carry generated, to the sum & in 2's complement we add '1' to the 1's complement.

∴ finally **Subtraction ≡ 2's Complement Addition**

Using either 2's or 1's complement, we have eliminated the subtraction operation and need only the appropriate complementer and an adder.

When performing subtraction we complement the subtrahend, and when performing addition we do not complement.

These operations can be accomplished by using a selective complementer and adder interconnected to form an Adder-Subtractor.

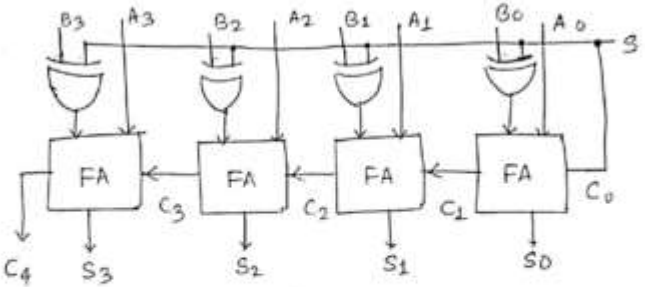
2's Complement = 1's Complement + 1

1's Complement = inversion of given binary number easily with inverter circuits. and we can add '1' to the sum by making the i/p carry of the parallel adder equal to '1'.

Thus, by using 1's complement and an unused adder input, the 2's complement is obtained inexpensively.

The circuit for subtracting (A−B) consists of parallel adder as shown below. with inverters placed between each 'B' terminal and the corresponding full-adder input. The input carry C_0 must be equal to 1.

P.T.O



Adder-Subtractor Circuit

The operation that is performed becomes

A plus 1's comp. of B plus 1

= A plus 2's Complement of B

For unsigned numbers, it gives A−B if A≥B or the 2's complement of B−A if A<B.

The addition and subtraction operations can be obtained into one circuit with one common binary adder.

This is done by including XOR gate with each full adder.

4-bit adder-subtractor circuit is shown above.

Input 'S' controls the operation.

when S=0, the circuit is an adder

when S=1, the circuit becomes subtractor.

Each XOR gate receives input S and one of the inputs of B. When S=0; we have $B_i \oplus 0 = B_i$. If full adders receives the value of B, and i/p carry is '0', the circuit performs A plus B. When S=1; we have $B_i \oplus 1 = \overline{B_i}$ and $C_0 = 1$. The circuit performs the operation A plus 2's Complement of B.