Signed Binary NO. → in signed binary NO. we require one extra bit to represent a +ve or -ve NO.. Signed binary NO. is represented in the following ways.

1. Sign - Magnitude
2. Sign - 1's complement
3. Sign - 2's "

* A positive NO. in any representation has a 0 as the left most bit for +, followed by positive binary number.
* A negative NO always has a '1' as the leftmost bit for negative followed by Magnitude bits as.

→ +ve NO. in sign Magnitude form.
→ 1's compliment of binary No. in 1's comp. form
→ 2's " " " " " 2's " " "

⇒ Table for +ve and -ve for bit pattern (one bit for sign.

	Sign Mag.	1's complt	2's complent
0 →	0,000		
1 →	0,001		
2 →	0,010		
3 →	0,011		
4 →	0,100		
5 →	0,101		
6 →	0,110		
7 →	0,111		
-8 →	Impossible	Impossible →	1000
-7 →	1,111 →	1,000 →	1,001
-6 →	1,110 →	1,001 →	1,010
-5 →	1,101 →	1,010 →	1,011
-4 →	1,100 →	1,011 →	1,100
-3 →	1,011 →	1,100 →	1,101
-2 →	1,010 →	1,101 →	1,110
-1 →	1,001 →	1,110 →	1,111
-0 →	1,000 →	1,111 →	Impossible

⇒ using 7 bit representation.

	+11	−11
Sign Mag.	0, 0 0 1 0 1 1	1, 0 0 1 0 1 1 (comp only sign bit)
Sign-1's comp.	0, 0 0 1 0 1 1	1, 1 1 0 1 0 0 (comp. each bit including sign bit)
Sign 2's "	0, 0 0 1 0 1 1	1, 1 1 0 1 0 1 (taking 2's comp. of +ve No. including sign bit).

Addition of Signed No. →

* Addition of two signed binary numbers in 2's complement form is performed by adding two No. including their sign bit. A carry in MSB (sign bit) ignored
* If −ve No. in 1's complement form the carry out of MSB (sign bit) is added to the LSB of sum and then ignored.

⇒ 7 bit representation. (in 2's comp. form).

+11 → 0, 0 0 1 0 1 1
+6 → 0, 0 0 0 1 1 0
+17 → 0 0 1 0 0 0 1

+11 = 0 0 0 1 0 1 1
−6 = 1 1 1 1 0 1 0
ignore → (1), 0 0 0 0 1 0 1
EAC
+5

−11 → 1 1 1 0 1 0 1
+6 → 0 0 0 1 1 0
NO → 1 1 1 1 0 1 1 → −5 in 2's comp. form.
EAC

−11 → 1 1 1 0 1 0 1
−6 → 1 1 1 1 0 1 0
ignore → (1) 1 1 0 1 1 1 1
EAC
−17 in 2's comp. form.

⇒ Addition using (1's comp. form).

+11 0 0 0 1 0 1 1
+6 0 0 0 0 1 1 0
+17 0 0 1 0 0 0 1

−11 → 1 1 1 0 1 0 0
−6 → 1 1 1 1 0 0 1
EAC → (1) 1 1 0 1 1 0 1
1
−17 ← 1 1 0 1 1 1 0 → in 1's comp. form

⇒ Note: * A zero with sign bit in different rep.

8-bit pattern.	+0	−0
Sign Mag	0 0000000	1 0000000
Sign 1's Comp.	0 0000000	1 1111111
Sign 2's Comp	0 0000000	

* The Range of Binary Integer NO. that can be stored in n bit register.
  → 1 bit is reserved for sign
  → k bits are used to binary NO.

$k = n-1$

⇒ In Sign Magnitude and 1's Comp' the Range of NO. is

$-(2^k-1)$ to $+(2^k-1)$

⇒ In 2's Compliments the range is

-2^k to $+(2^k-1)$

for example — 8 bit register Can Store.

	Sign-Mag	1's comp'	2' Comp'
+127	0 1111111	0 1111111	01111111
−127	1 1111111	1 0000000	10000001
−128	Impossible	Impossible	1.0000000

Arthematic Substraction — using 2's Compliment. (using 7 bit pattern). Do: (+9) − (+6) = +3

+9 → 0001001
+6 → 0000110
−9 → 1ϕϕ0111
−6 → 1111010

Do. (+9) − (−6) = +9+6

```
  0001001
  0000110
---------
  0001111 → +15
```

```
  0001001 → +9 minuend
  1111010 → 2's comp of Subtrahend
+ ---------
 (1)0000011 ← +3
 Ignore
 EAC
```

(−9) − (+6)

```
  1110111
+ 1111010
---------
 (1)1110001 ← −15 in 2's complement
 ignore
 EAC
```

Encoding and Conversion → Encoding requires More bits compared to conversion. on the Average $\log_2^{10} = 3.32$ bits are required when decimal Number are converted in to binary while 4 bits per digit are required in encoding. The ratio $\frac{4}{3.32} = 1.2$ is measure of extra bits required this consequently requires extra storage as well.

* Conversion from decimal to binary is a slower process An algorithm involving successive division is needed for Conversion While encoding is straight forward table look up.

* There are (3×10^{10}) ways of developing Codes. only few of them are chosen from viewpoint of
  i) ease in Arithmetic
  ii) error detection property
  iii) ease in coding.

classification of codes →

* weighted codes — $d = \Sigma w_i b_i$ w_i = weight of i^{th} bit
  $b_i = 0/1$

* Self Complementy code :- on Replacing 0 by 1 and 1 by 0 in a code for decimal digit d, one obtained the code for decimal dgit 9-d. it is called Self Complementy Code

* cyclic / Reflected / Gray code — A code in which each code differs from its Neighbour by only one bit

* Error detecting codes. The error detecting codes are ~~used~~ constructed by using redundant bits in the codes. One such Method is:

→ Introduce fifth bit such that total No. of 1's in 5 bit group is odd. This is called parity bit. If in a code total No. of 1's is not odd, one can conclude that exist a single error

* Error detecting codes :-

Transmitted

Information bits →					Row Parity ↓
	0	0	0	0	1
	0	0	0	1	0
	0	0	1	0	0
	0	0	1	1	0
	0	1	0	0	0
→	1	0	1	1	

Column Parity

Received

0	0	0	0	1	
0	0	0	1	0	← Parity failure
0	0	1	0	0	
0	0	1	1	1	
0	1	0	0	0	
1	0	1	1		

↳ Parity failure

→ single error detecting / correcting code.

1	2	3	4	5	6	7	← bit Position.
P_1	P_2	I_3	P_4	I_5	I_6	I_7	
0	0	0	0	0	0	0	
1	1	0	1	0	0	1	
1	0	1	0	0	1	0	
0	1	1	0	0	1	1	
1	0	0	1	1	0	0	
0	1	0	1	1	0	1	
0	0	1	1	1	1	0	
1	1	1	1	1	1	1	

1, 2, 4 → Parity bit
3, 5, 6, 7 → Information bit

bit at position 1 – Even Parity for 1, 3, 5, 7

bit at position 2 – Even parity for 2, 3, 6, 7.

bit at position 4 – Even Parity For 4, 5, 6, 7.

⇒ The following procedure are used to detect and correct the error.

* The position 1,3,5,7 passes even parity test → $C_1 = 0$ else $C_1 = 1$
* The " 2,3,6,7 " " " " → C_2 is 0 or else C_2 = One
* The position 4,5,6,7 " " " " → $C_4 = 0$ else $C_4 = 1$

→ The decimal equivalent of $C_4 C_2 C_1$ gives position of incorrect bit if $C_4 C_2 C_1 = 0$ it indicates no error

→ example : Received code is 0110110 that means.

$C_1 = 0$, $C_2 = 1$, $C_4 = 0$, decimal $[C_4 C_2 C_1] = de[010] = 2$

Correct code is 0010110

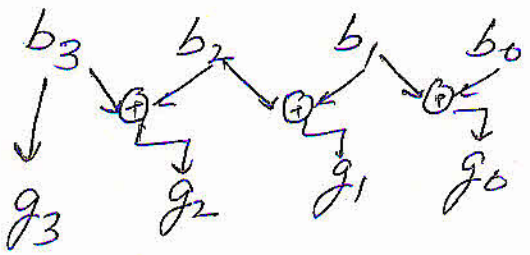
Codes →

d	weights 8421	8-4-2-1 / 8421	2421	X-3
0 →	0 0 0 0	0 0 0 0	0 0 0 0	0 0 1 1
1 →	0 0 0 1	0 1 1 1	0 0 0 1	0 1 0 0
2 →	0 0 1 0	0 1 1 0	0 0 1 0	0 1 0 1
3 →	0 0 1 1	0 1 0 1	0 0 1 1	0 1 1 0
4 →	0 1 0 0	0 1 0 0	0 1 0 0	0 1 1 1
5 →	0 1 0 1	1 0 0 0	1 0 1 1	1 0 0 0
6 →	0 1 1 0	1 0 1 0	1 1 0 0	1 0 0 1
7 →	0 1 1 1	1 0 0 1	1 1 0 1	1 0 1 0
8 →	1 0 0 0	1 1 1 0	1 1 1 0	1 0 1 1
9 →	1 0 0 1	1 1 1 1	1 1 1 1	1 1 0 0

* A necessary condition for self complementry weighted code is that the sum of its weights be ~~one~~ Nine (9)
* A self complementry code need not necessary be weighted e.g. X-3 non weighted self complementry

←

Conversion from binary to gray →

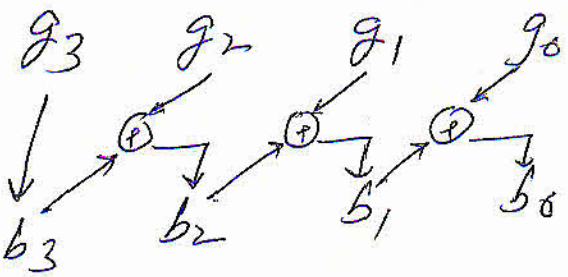


Rule
* Retain MSB
* Ex-OR every bit with next bit and place the Result or add two bits and ignore carry

dec	Binary	Gray
0 →	0 0 0 0	→ 0 0 0 0
1 →	0 0 0 1	→ 0 0 0 1
2 →	0 0 1 0	→ 0 0 1 1
3 →	0 0 1 1	→ 0 0 1 0
4 →	0 1 0 0	→ 0 1 1 0
5 →	0 1 0 1	→ 0 1 1 1
6 →	0 1 1 0	→ 0 1 0 1
7 →	0 1 1 1	→ 0 1 0 0

gray code
↓
0 1 0 1 1 0 1

⇒ Gray to Binary



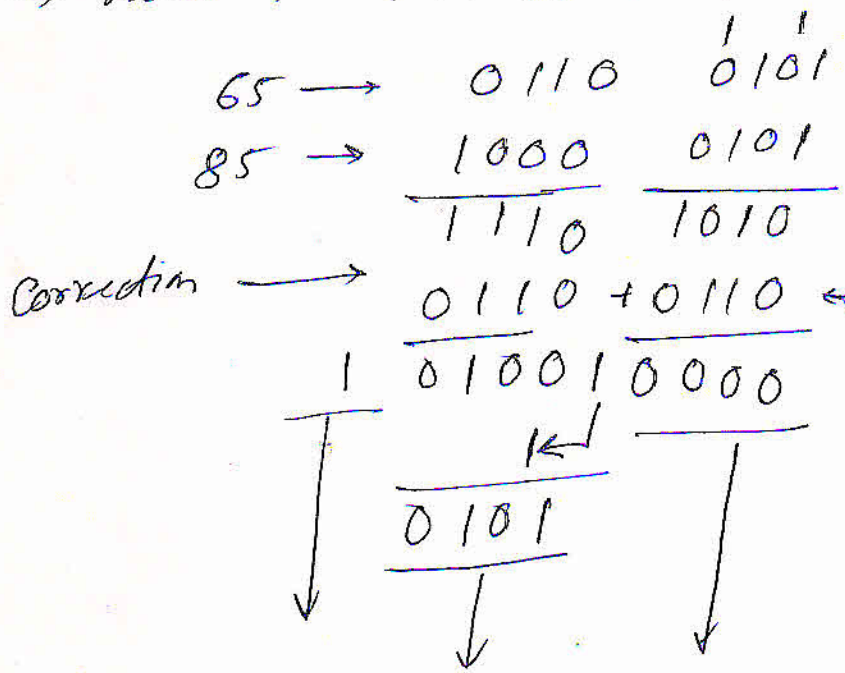
Rule
* Retain MSB
* Ex-OR this bit with next bit from gray code
* The Result is then Ex-ORed with next bit from gray code and so, on,

Gry code is used in A/D Converter.

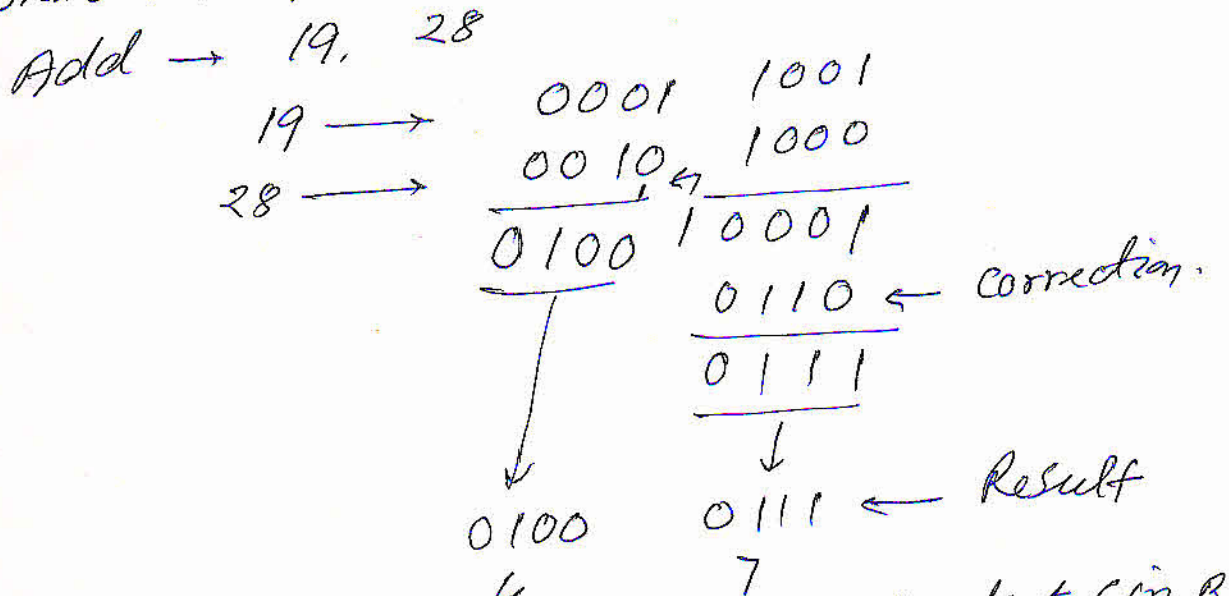
BCD – Binary coded decimal – 0-9 with four binary bit 25 → 0010, 0101

BCD Addition → If Binary sum of pair of digits >10, get correct BCD code by adding 0110(6) to sum else leave it alone.

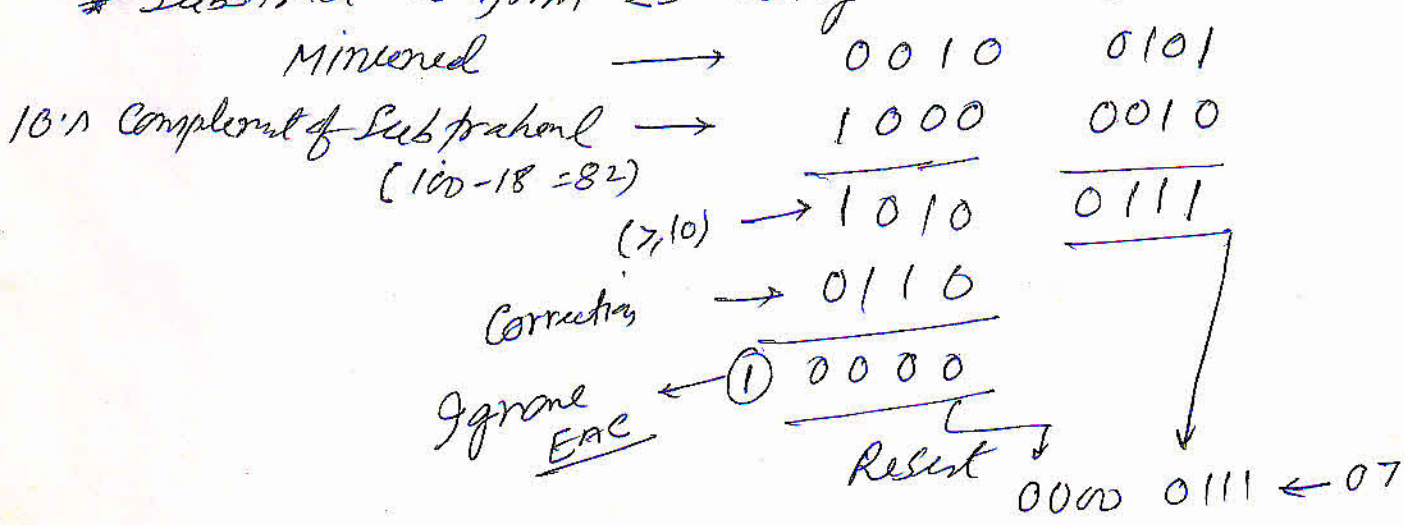
⇒ Add 65 and 85



* In BCD addition, correction is also required when sum yields a carry



* Subtract 18 from 25 using 10's complement (in BCD).



* Subtract 25 from 18

→ 0001 1000

→ 0111 0101

1001 | 1101

0100 ← correction.

↓ 9   10011 ↓3

NO EAC result = −(10's comp of 93) = −07

Addition of Xs − 3 Code →

* If a carry is generated add 0011 (3) to the sum and if no carry is generated substract 0011 (3) from sum.

⇒ Add 525 and 639 (in Xs-3 code)

Augend → 1000 0101 1000

Addend → 1001 0110 1100

1 0001 1100 10100

+11 +11 −11 +11

[0110 + BCD] → 0100 0100 1001 0111
X's-3.

result = 1 1 6 4

⇒ Subtract 863 − 425 (using 10's complement) (in Xs-3 code)

Minuend 863 → 1011 1001 0110

10's comp of Subtrahend (575) → 1000 1010 1000

Ignore EAC ← ① 0100 10011 1110 ← correction.

+11 +11 −11

0111 0110 1011

result = 4 3 8