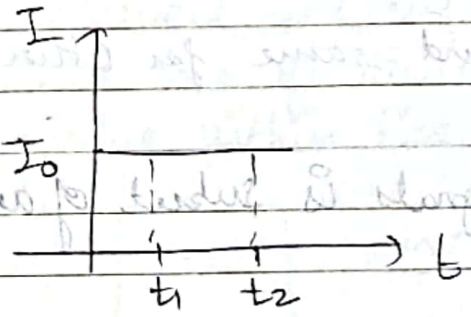
⇒ BOOLEAN ALGEBRA AND COMBINATIONAL CIRCUITS

Signal: A signal is a function that represents the variation of a physical quantity with respect to any parameter

In ~~electro~~ electrical & electronics, usually signal is variation of electrical quantity (generally I or V) with time.

- If I is same for different time then it is direct current (DC)

$dI = 0$

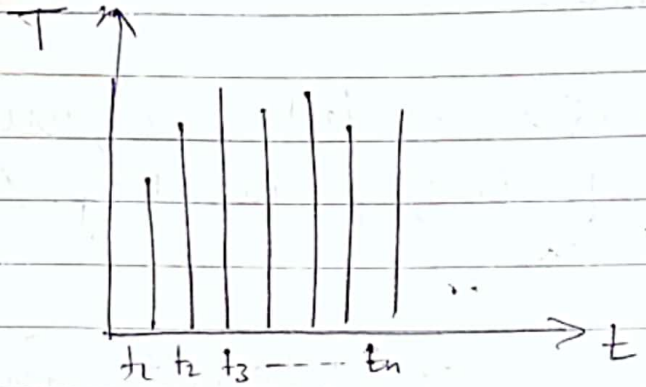


Transducers → Device used to convert non-electrical signals to electrical signals.
Reverse transducers → Device used to convert electrical signals to non-electrical signals.

System: A physical device that performs an operation a signal.
Eg: Amplifier.

Analog Signals & Digital Signals:

Discrete time signals: The signal which is defined for discrete time intervals is called discrete time signals.

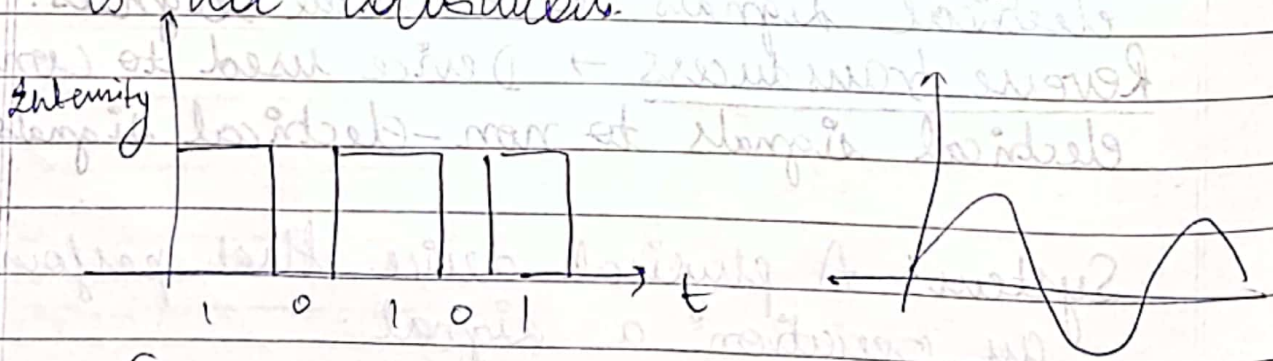


$f(t_0)$, $f(t_1)$ - -

We know these values but not anything b/w time t_1 & t_0 and same for other intervals.

#Discrete time signals is subset of analog signals.

Continuous time signals: Often referred to as continuous signals even when signal function is not continuous.



(Abrupt amplitude variations)

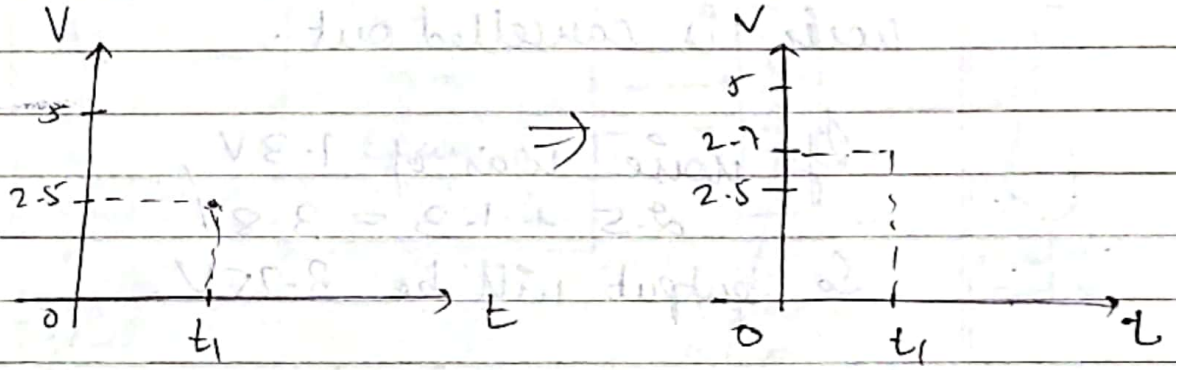
Need of Digital Signals:

→ All real life signals are analog in nature.

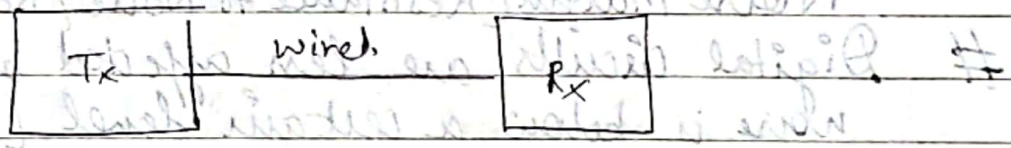
→ However, digital signal is used in communication process to minimise effect of noise.

noise
↓
unwanted signal.

Suppose we have analog signal. At some time t_1, we want to transmit 2.5 V

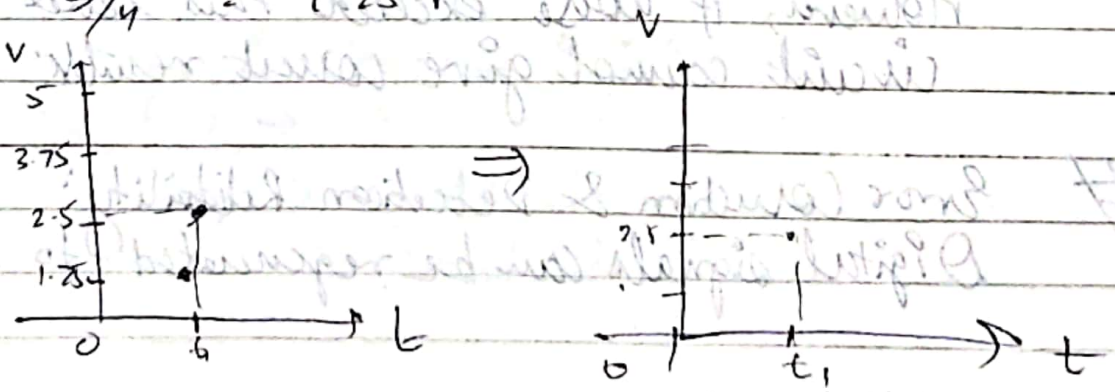


Now if a noise of 0.2 V is added, at the receivers end we get a distorted signal. As, in case of analog signal voltage can take any value within the given limit.



Now suppose we use digital signals and as we have levels in digital signal, suppose we take 4 levels. from 0 - 5 V.

5/4 = 1.25 V



Now, suppose again noise of 0.2V is added. So we will have 0.2+2.5V at receiver end. But as we can see we do not have any level corresponding to 2.7V. So we will use the level just below it which in this case is 2.5V.

So at transmitter's end we transmitted 2.5V which is what we receive, the effect of noise is cancelled out.

If noise was of 1.3V,

$2.5 + 1.3 = 3.8V$

So output will be 3.75V.

∴ Digital signal is good for noise rejection when noise is smaller. But when value of noise is high we have to look for more advanced methods.

Noise Margin (Resistance to Noise / Robustness)

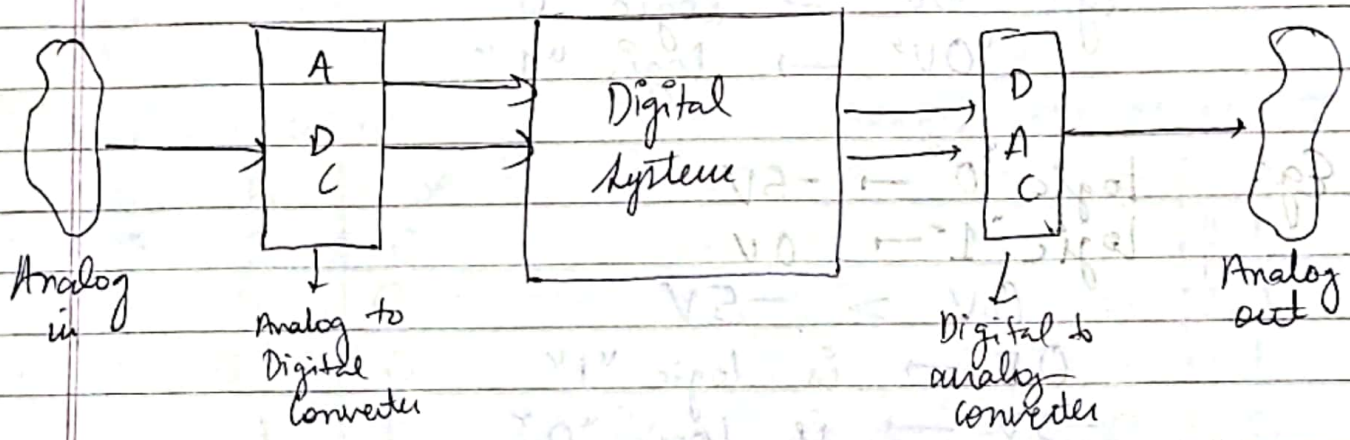
Digital circuits are less affected by noise. If the noise is below a certain level (the noise margin) a digital circuit behaves as if there was no noise at all. The stream of bits can be constructed into a perfect replica of the original source.

However, if noise exceeds this level, the digital circuit cannot give correct results.

Error Correction & Detection Reliability:

Digital signals can be regenerated to achieve

lossless data transmission, within certain limits. Analog signal transmission & processing, by contrast, always introduces noise. Digital systems are highly reliable one of the reasons for that is use of error correction codes.



Advantages:

- → Noise immunity
- → Uses less bandwidth
- → Encryption
- → Efficiency ↑ for long dist. transmission
- → Design is easy.
- → High speed
- → Flexibility
- → Cheap electronic circuits

Positive & Negative logic

1. Positive logic →
   - Higher voltage corresponds to logic "1".
   - Lower voltage corresponds to logic "0".

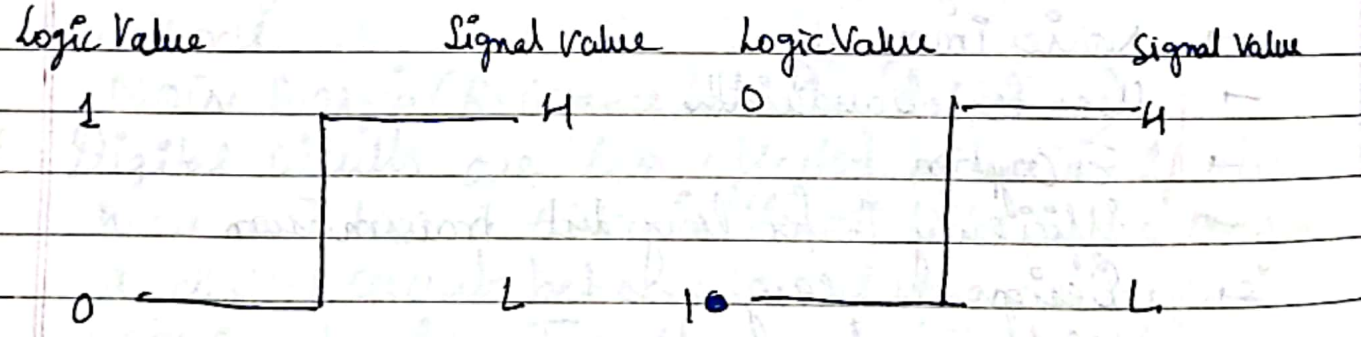
Eg: "5V" ⇒ logic "1"
"0V" ⇒ logic "0"

2. Negative logic:

- Higher voltage corresponds logic "0"
- Lower voltage corresponds logic "1"

Eg: "5V" ⇒ logic "0"
"0V" ⇒ logic "1"

Eg: logic "0" → -5V
logic "1" → 0V
0V > -5V
0V → is logic "1"
-5V → is logic "0"
∴ This is Positive logic



a) Positive logic b) Negative logic

DUAL FORM:

i) Positive & Negative logic AND GATE

A	B	Y
L	L	L
L	H	L
H	L	L
H	H	H

(+)ve logic AND GATE

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

(−)ve logic AND GATE

A	B	Y
1	1	1
1	0	1
0	1	1
0	0	0

ii) Positive and Negative OR GATE:

A	B	Y
L	L	L
L	H	H
H	L	H
H	H	H

(+)ve logic

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

(−)ve logic

A	B	Y
1	1	1
1	0	0
0	1	0
0	0	0

✓ (+)ve logic AND GATE = (−)ve logic OR GATE

✓ (−)ve logic AND-GATE = (+)ve logic OR GATE

Dual form: Dual expression is used to convert +ve logic to -ve logic & -ve logic to +ve logic

$\underbrace{A \cdot B}_{(+)ve} \xrightarrow{\text{Dual}} \underbrace{A + B}_{(-)ve}$

$A + B \xrightarrow{\text{Dual}} A \cdot B$

→ AND ⟷ OR
→ 0 ⟷ 1
→ Variables as it is

} Dual form

Self Dual:

→ For any logical expression, two times dual gives the same expression
→ In self dual exp, one time dual gives the same expression.

Eg) $F = A \cdot B \cdot \bar{C} + \bar{A} \cdot B \cdot C \rightarrow A \cdot B \cdot C$
$F' = (A + B + \bar{C}) \cdot (\bar{A} + B + C) \cdot (A + B + C)$
$F'' = A \cdot B \cdot \bar{C} + \bar{A} \cdot B \cdot C + A \cdot B \cdot C$

Eg) $G = A \cdot B + B \cdot C + C \cdot A$
$G' = (A + B) \cdot (B + C) \cdot (C + A)$

$\Rightarrow (B + A\cdot C)(A + C)$ $\quad [(X+Y)(X+Z) = X + (Y\cdot Z)]$

$\Rightarrow A\cdot B + A\cdot A\cdot C + B\cdot C + A\cdot C\cdot C$

$\Rightarrow A\cdot B + A\cdot C + B\cdot C + A\cdot C \quad (X+X = X)$

$G' \Rightarrow A\cdot B + A\cdot C + B\cdot C$

No. of self dual : for n variables $= 2^{2^{n-1}}$

Eg $\Rightarrow n = 2$

$\Rightarrow 2^{2^1} \Rightarrow 2^2 = 4$

A & B :

$A \xrightarrow{\text{Dual}} A$, $\bar{A} \xrightarrow{\text{Dual}} \bar{A}$, $\bar{B} \xrightarrow{\text{Dual}} \bar{B}$, $B \xrightarrow{\text{Dual}} B$

R's Complement:

Complement

- r's Complement [Radix Complement]
- (r-1)'s complement [Diminished radix complement]

Eg: Comp of $(7)_{10} = 3 \quad (\because 10-7=3)$

Comp of $(6)_{10} = 4 \quad (\because 10-6=4)$

Generalising: N = 7

n = 1 (no. of digits in N)

r = 10 (Base)

r's Complement $= r^n - N$

Q. N = 5690, determine 10's complement:

$r = 10$, $N = 5690$, $n = 4$

$\Rightarrow 10^4 - 5690 = 10000 - 5690$

$= 4310$

Q. N = 1101. Find 2's complement:

$r = 2$ $N = 1101$ $n = 4$

$= (2^4)_{10} - (1101)_2$

$= (16)_{10} - (1101)_2$

$= 10000 - 1101$

$$\begin{array}{r} {}^{0}\cancel{1}^{10}\,\cancel{0}^{1}\,\cancel{0}^{1}\,\cancel{0}^{1}\,\cancel{0}\;10 \\ 1\;1\;0\;1 \\ \hline 0\;0\;0\;1\;1 \end{array}$$

$\Rightarrow (11)_2 = (3)_{10}$

(r−1)'s Complement

	r's Compl	(r−1)'s Comp
r = 10	10's comp	9's comp
r = 2	2's	1's
r = 8	8's	7's
r = 16	16's	15's ⇒ F's

r's complement = $r^n - N$

(r-1)'s comp = $[r^n - N] - 1$

(r-1)'s comp = r's comp − 1

(r-1)'s comp + 1 = r's comp

No borrow operation involved in (r-1)'s comp

borrow operation involved in r's comp

Eg: 7's comp of Octal No. 5674

$= (8^4)_{10} - 5674 - 1$

$= (4096)_{10} - 5674 - 1$

$= 10000 - 5674 - 1$

$= 7777 - 5674$

$= 7777$
$\quad - 5674$ (No borrow involved)
$\Rightarrow \quad 2103$

$10000 - 1 = 7777$

Eg: 8's complement of 5674

7's comp → 2103
$\quad +1$
$\quad 2104$

8's complement

Eg: 1's complement of 1101

$= 1111 - 1101$
$\quad 0010$
$\quad +1$
$\quad 0011 \rightarrow$ 2's comp

For $n=4$:

$(r-1)$'s comp $= r^4 - N - 1$

$= (r^4 - 1) - N$

if $r = 10$

$(r^4 - 1) = 9999$

if $r = 8$

$(r^4 - 1) = 7777$

if $r = 16$

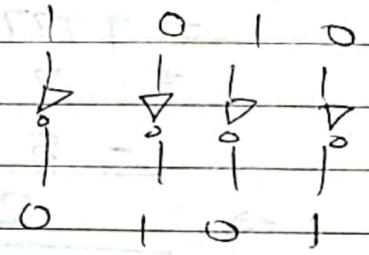
$(r^4 - 1) = FFFF$

if $r = 2$,

$(r^4 - 1) = 1111$

Eg: 1's comp of $(1010)_2$

$$\begin{array}{r} 1111 \\ -\ 1010 \\ \hline 0101 \end{array}$$



Reverse the digits: $(1010)_2 \xrightarrow{1's} (0101)_2$

Q Represent 10111000 in 2's comp.

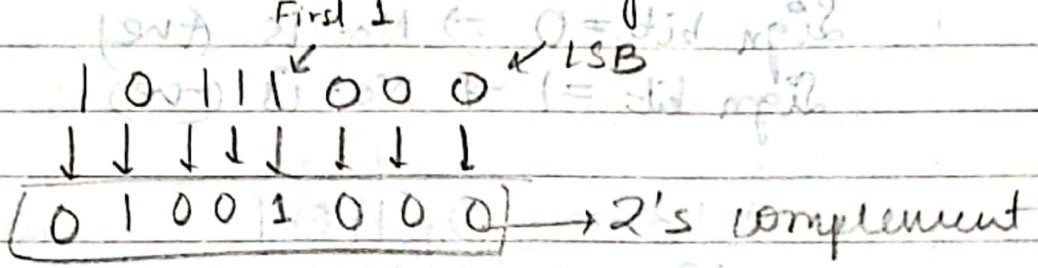
1's comp ⇒ 01000111

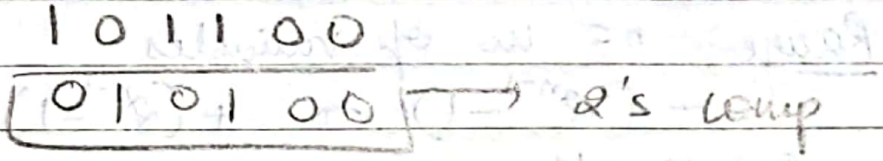
$+\ 1$

2's comp ⇒ 01001000

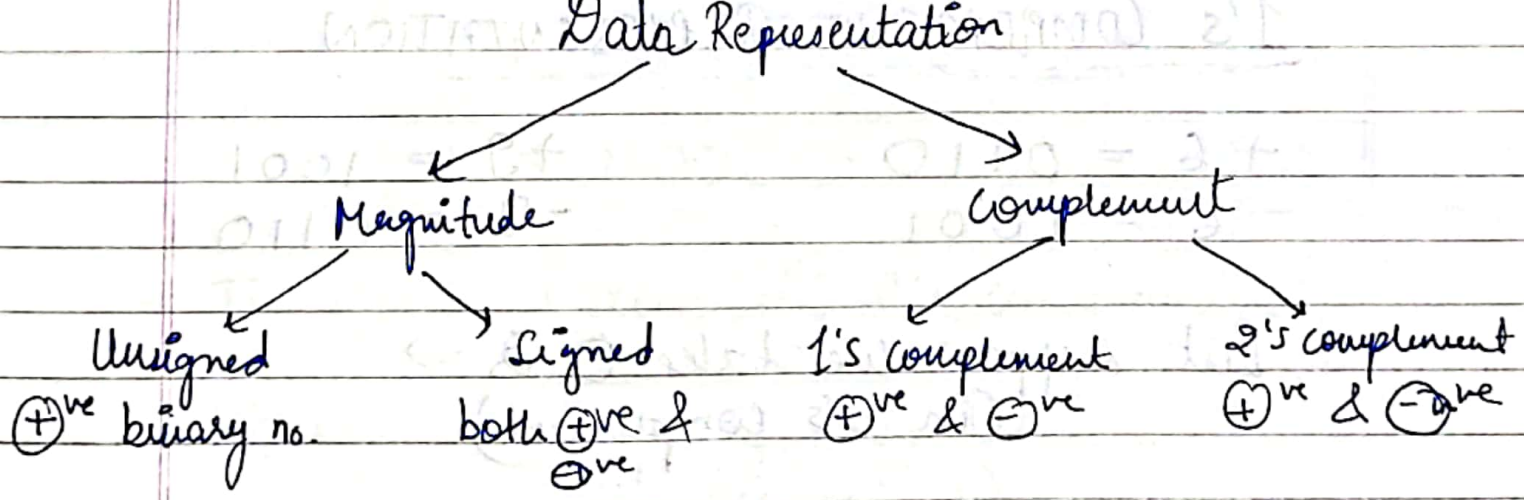
Shortcut for 2's comp:

1) Write down the given number
2) Starting from LSB, copy all zeroes till first 1
3) Copy the first 1.
4) Complement all the remaining bits



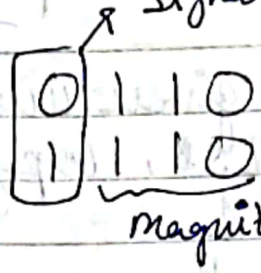


Data Representation for Signed Magnitude



Unsigned mag →
+6 = 110
-6 ⇒ can't represent

Signed mag:



Sign bit = 0 ⇒ No is (+ve)
Sign bit = 1 ⇒ No. is (-ve)

+13 = 0 1101
-13 = 1 1101

Range: n = no. of variables

$-(2^{n-1}-1)$ to $+(2^{n-1}-1)$

Eg: n = 4

−7 to +7

→ Here we have 2 zeroes → (+0) 0000, (-0) 1000

1's COMPLEMENT REPRESENTATION

+6 = 0110 +9 = 1001
−6 = 1001 -9 = 0110

But suppose we take 0 ⇒ (in 1's complement)

+0 = 0000 (positive zero)
−0 = 1111 (negative zero)

→ here we are getting a positive zero and a negative zero.

Range : n = no. of variables

$-(2^{n-1}-1)$ to $+(2^{n-1}-1)$

$n=4 \Rightarrow -7$ to $+7$

2's COMPLEMENT REPRESENTATION

$+6 = 0110$

1's comp : 1001

$+1$

2's comp : 1010

$-6 = 1010$

$+4 = 0100$

1's comp : 1011

$+1$

2's comp : 1100

$-4 = 1100$

→ There's only 1 zero in 2's complement : 0000

Range : n = no of variables

-2^{n-1} to $(2^{n-1}-1)$

Eg : $n = 4$

$\Rightarrow -8$ to $+7$

8 (ve) integers — 1 zero — 7 (ve) integers

MSB indicates sign → in 2's comp

1 → ⊖ve

0 → ⊕ve

BINARY SUBTRACTION USING 1's COMPLEMENT:

1) Convert number to be subtracted to its 1's complement form
2) Perform the addition
3) If the final carry is 1; then add it to the result obtained in step 2. If final carry is 0, result obtained in step 2 is negative and in the 1's complement form.

$$A - B = A + (-B)$$

↑ 1's complement of B = −B

Eg: Perform $(1100)_2 - (0101)_2$

A = 1100 $(12)_{10}$ B = 0101 $(5)_{10}$

1's comp of B = 1010

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

carry

128 64 32 1 = 225

$$\begin{array}{r} 0\ 1\ 1\ 0 \\ +\ \ \ \ 1 \\ \hline 0\ 1\ 1\ 1 \end{array} \quad (7)_{10}$$

Q. Perform $(0101)_2 - (1100)_2$

$A = 0101\ (+5) \quad B = 1100\ (+12)$

$-B = 0011$

$$\begin{array}{r} 0\ 1\ 0\ 1 \\ 0\ 0\ 1\ 1 \\ \hline 1\ 0\ 0\ 0 \end{array}$$

carry. (Ans is negative)

1's comp of result

$0111 \Rightarrow$ Ans

$(-7)_{10} \rightarrow$ Ans

BINARY SUBTRACTION USING 2'S COMPLEMENT:

1) Find 2's complement of number to be subtracted
2) Perform the addition
3) If final carry is generated then the result is positive and in its true form. If final carry is not produce then the result is negative and in 2's complement form.

$$A - B = A + (-B)$$

↑

2's comp of $B = -B$

Note: We neglect final carry in two's complement method.

Eg: $(1001)_2 - (0100)_2$ $(9)_{10} - (4)_{10}$

A = 1001 B = 0100

1's comp of B = 1011

+ 1

2's comp of B ⇒ 1100

(-B)

```
  1 0 0 1
  1 1 0 0
---------
1 0 1 0 1
---------
```

carry

Ans ⇒ $(0101)_2$ $(5)_{10}$

Eg: $(0110)_2 - (1011)_2$

A = 0110

B = 1001

0110

+1

0111

B = 1011

0100

+ 1

0101

```
  0 1 1 0
  0 1 0 1
---------
  1 0 1 1
---------
```

⟶ (-ve) ans

1's comp ⇒ 0100

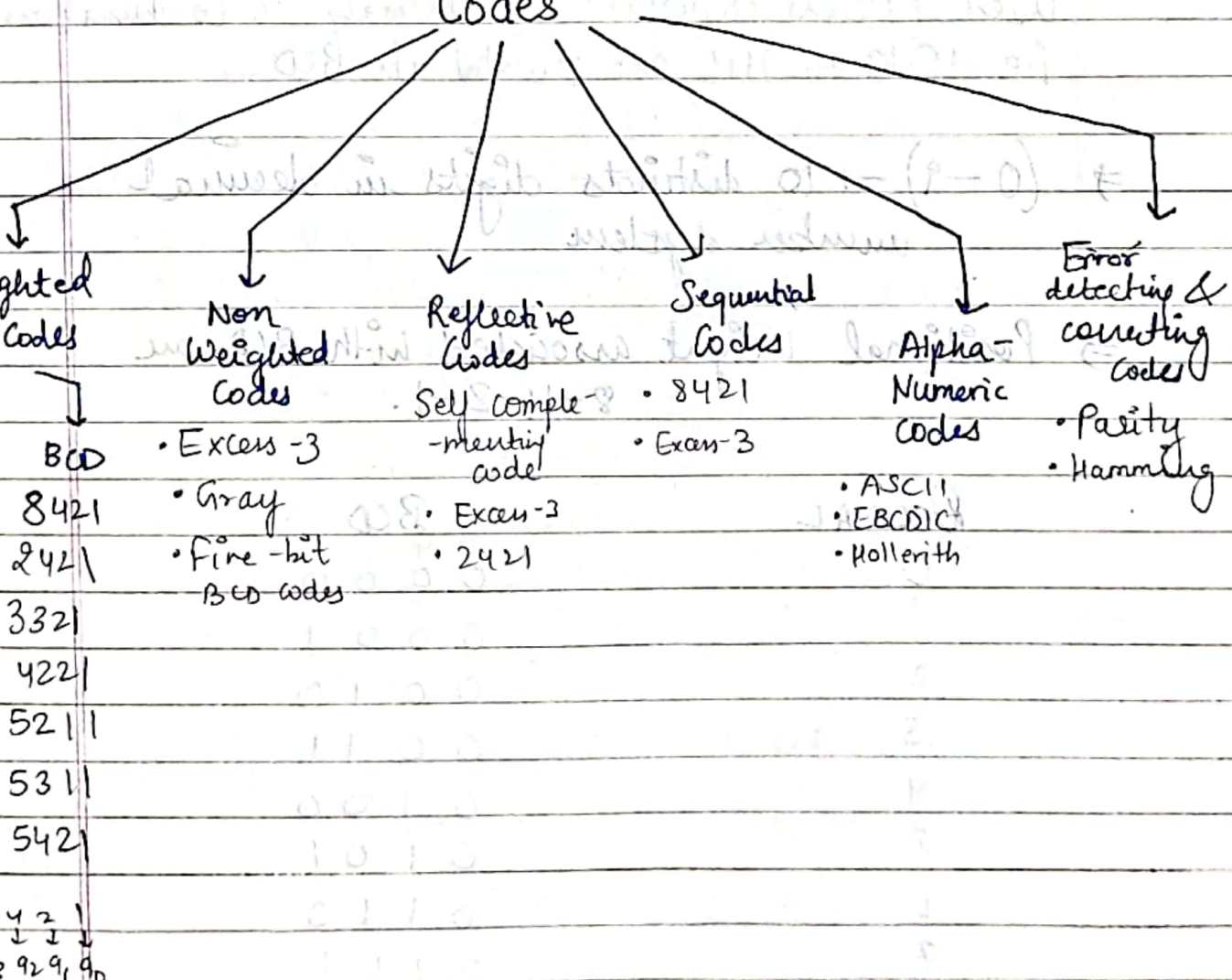
+1

2's comp ⟵ 0101 ⇒ $(+5)_{10}$

$\therefore (-5)_{10}$ is answer

BINARY CODES

- When numbers, alphabets or words are represented by a specific group of symbols, i.e they are encoded.
- The group of symbols used to encode them are called codes
- Digital data is represented, stored and transmitted as groups of binary digits (bits)



BINARY CODED DECIMAL:

- These codes obey positional weight principle
- Each position represents a specific weight.
- In this code each decimal digit is represented by 4 bit binary number
- BCD is a way to express each of the decimal digits with a binary code. In the BCD, with four bits we can represent sixteen numbers (0000 to 1111)
- But in BCD code only first ten of these are used (0000 to 1001), remaining 6 combinations i.e 1010 to 1111 are invalid in BCD.

⇒ (0–9) → 10 distincts digits in decimal number system

⇒ Positional weight associated with BCD are 8–4–2–1.

DECIMAL	BCD (8 4 2 1)
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
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

10, 11 -- 15 → They are not decimal digits, they are decimal numbers.

Convert decimal to BCD:

1) $(17)_{10}$ → 1 7

0001 0111

BCD → 0001 0111

2) $(156)_{10}$ → 1 5 6

0001 0101 0110

BCD → 0001 0101 0110

Convert BCD to decimal:

1) 10100 BCD

⇒ 0001 0100

⇒ $(14)_{10}$

2) 1001001 BCD

⇒ 0100 1001

⇒ $(49)_{10}$

Comparison b/w Binary & BCD:

1) $(10)_{10}$ → Binary: 1010 BCD: 0001 0000

	binary	BCD
$(12)_{10}$ →	1100	00010010

* BCD is less efficient than binary as it is using more bits than binary.
* BCD arithmetic is little more complicated
* The addition, sub has different rules.

BCD Addition:

i) Sum ≤ 9, Final Carry = 0 : Answer is correct
ii) Sum ≤ 9, Final carry = 1 : Answer is incorrect and we need to add (0110): 6
iii) Sum > 9, Final carry = 0 : Answer is incorrect, add 0110

Ex: $(2)_{10} + (6)_{10}$ BCD addition

```
  0 0 1 0
  0 1 1 0
  -------
  1 0 0 0
```

Sum < 9, final carry = 0
Ans: 1000

Ex: $(3)_{10} + (7)_{10}$ BCD

```
  0 0 1 1
  0 1 1 1
  -------
  1 0 1 0
```

Sum > 9, final carry = 0

We have to add 6 ⇒

```
  1 0 1 0
  0 1 1 0
-------------
0 0 0 1 0 0 0 0
```

= $(10)_{10}$

Ex: $(8)_{10} + (9)_{10}$

```
  1 0 0 0
  1 0 0 1
-----------
1 0 0 0 1
```

Sum < 9 ↙ , carry = 1

```
    1 0 0 0 1
      0 1 1 0
-----------------
0 0 0 1 0 0 1 1 1
```

= $(17)_{10}$

Ex: $(57)_{10} + (26)_{10}$

```
  0 1 0 1 0 1 1 1
  0 0 1 0 0 1 1 0
-----------------
  0 1 1 1 1 1 0 1
```

7 < 9 ↙ carry = 0 ; 13 > 9 ↘

```
    0 1 1 0
-----------------
1 0 0 0 0 0 1 1
```

8 3

= $(83)_{10}$

Eg: $(93)_{10} + (42)_{10}$

```
  1 0 0 1   0 0 1 1
+ 0 1 0 0   0 0 1 0
-------------------
  1 1 0 1   0 1 0 1
```

⇒ 9 & c = 0

```
  1 1 0 1   0 1 0 1
+ 0 1 1 0
-----------------
1 0 0 1 1   0 1 0 1
```

⇒ 0001 0011 0101

Eg: $(49)_{10} + (23)_{10}$

```
  0 1 0 0   1 0 0 1
+ 0 0 1 0   0 0 1 1
-------------------
  0 1 1 0   1 1 0 0
```

> 9

```
  0 1 1 0   1 1 0 0
            +
      0 1 1 0
-----------------
  0 1 1 1   0 0 1 0
```

Ans ⇒ (0111 0010)

BCD Subtraction

(Using 9's Complement)

1. Take 9's complement for BCD subtrahend
2. Add it to the minuend using BCD Addition
3. If the result is invalid BCD, correct by adding 6.
4. Shift the carry to the next bits
5. If end around carry is generated then add it to the result.

Eg. $(1234)_{10} = (1000)_{10}$

Eg: 9's complement of 1000:

$10^n - 1 \Rightarrow 10^4 - 1 = 9999$

$$\begin{array}{r} 9999 \\ -1000 \\ \hline 8999 \end{array}$$

Q1) BCD Sub for: $(79)_{10} - (26)_{10}$

9's comp of 26 $\Rightarrow$ 99 − 26 = 73

$$\begin{array}{rcl} 79 & \Rightarrow & 0111\ \ 1001 \\ 73 & \Rightarrow & +\ 0111\ \ 0011 \\ & & \hline 1110\ \ 1100 \\ & & \text{(add 6)} \end{array}$$

12 > 9

1110 1100
+ 0110
1111 0010
15>9 2<9
(add 6)

1111 0010
+ 0110
carry ← 1 0101 0010
→ +0110 +1
0101 0011
⇒ $(53)_{10}$

Q2) $(89)_{10} - (54)_{10}$
9's comp of 54 ⇒ 99 − 54 = 45

1000 1001
+ 0100 0101
1100 1110
>9

1100 1110
+ 0110
1101 0100
>9

1101 0100
+ 0110
carry ← 1 0011 0100
→ +1
0011 0101
⇒ $(35)_{10}$

(Using 10's Complement)

1. Take 10's complement for subtrahend
2. Add it to the minuend by BCD addition
3. If error BCD digits are there correct by adding
4. Follow up the carry to next bits.
5. If end around carry is there discard the carry. If not, say result is (-ve) & is in its 10's comp. Then take 10's comp to get the actual result.

Q. $(79)_{10} - (26)_{10}$

$$\begin{array}{r} 99 \\ 26 \\ \hline 73 \\ +1 \\ \hline 74 \end{array}$$

9's → 73

10's → 74

$$\begin{array}{r} 0111\ 1001 \\ 0111\ 0100 \\ \hline 1110\ 1101 \end{array}$$

(1101 → -9)

$$\begin{array}{r} 1110\ 1101 \\ +\ 0110 \\ \hline 1111\ 0011 \\ +0110 \\ \hline 1\ 0101\ 0011 \end{array}$$

discard the carry

$= -(53)_{10}$

Date / /
Page No.

Signed BCD →

- Sign is represented in 4 bits
- Plus sign is denoted by four zeros
- Minus sign with BCD equi of 9

+52 = 0000 0101 0010
-52 = 1001 0101 0010

Non-weighted code:-

In non-weighted code, there is not positional weight i.e. each position within the binary number is not assigned a prefixed value. No specific weights are assigned to bit position in non-weighted code.

Excess-3 Code

Decimal ⟶ 8-4-2-1 code (BCD) $\xrightarrow[0011]{\text{Add}}$ Excess-3 code (3)

Eg: 5 ⟶ 0101 ⟶ 1000 (8)

$$\begin{array}{r} 0101 \\ +0011 \\ \hline 1000 \end{array}$$

Decimal	BCD	XS-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Self complement

0011 ⟷ 1100

(0) (9)

Eg $(24)_{10}$

0010	0100
+0011	0011
0101	0111

$\Rightarrow$ XS-3 code

Eg) $(658)_{10}$

6	5	8	
0110	0101	1000	
+0011	0011	0011	
1001	1000	1101	XS-3

- XS-3 code is self complementing
- It is the only unweighted code that is self comp.

Finding if code is self complementing

$w_3 w_2 w_1 w_0$

If $w_3 + w_2 + w_1 + w_0 = 9$ (self comp)

$\neq 9$ (Not self comp.)

Eg: 2-4-2-1

$= 2+4+2+1 = 9$ ✓ self comp.

8-4-2-1

$= 8+4+2+1 = 15 \neq 9$

Other eg: 3-3-1-1, 4-3-1-1

Q1) $(2)_{10} + (5)_{10}$

$(2)_{10} \rightarrow 0010 \text{ (BCD)} \rightarrow 0101$

$(5)_{10} \rightarrow 0101 \text{ (BCD)} \rightarrow 1000$

XS-3

0101 (XS-3)
1000 (XS-3)
1101 (∴ XS-6)

(So we will subtract 0011 [3])

1X01
−0011
1010 ⇒ Ans (XS-3)

Q2) $(27)_{10} + (39)_{10}$

$(27)_{10} \rightarrow 0010\ 0111 \text{ (BCD)} \rightarrow 0101\ 1010$ (XS-3)

$(39)_{10} \rightarrow 0011\ 1001 \rightarrow 0110\ 1100$ (XS-3)

C_2 → 1 1 ← C_1

0101 1010
0110 1100
1100 0110
−0011 +0011
1001 1001
9 9

C_1 = Final carry = 1 : Add 0011
C_1 = Final carry = 0 : Sub 0011

(1)
2
+39
61
66
+33 (XS-3)
99

GRAY CODE

- Also known as reflected binary code
- Unweighted code
- Unit dist. code & minimum error code
- Two successive values differ in only 1 bit
- Binary is converted to gray code in switching operation.

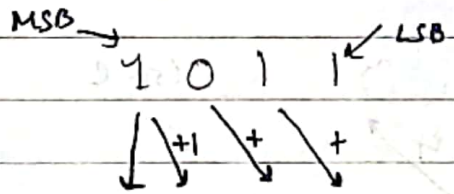
Decimals	Binary	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000
16	10000	0000

To convert from 7 to 8 in binary all 4 bits have to be changed, while in gray code only 1 bit has to be changed. This is true for any two successive values.
This is why gray code is called unit dist code & switching operation is reduced in gray code.

Binary to Gray Conversion:

1. Record the MSB as it is
2. Add the MSB to the next bit, record the sum & neglect the carry → XOR
3. Repeat the process

Eg: Convert 1011 to Gray Code.



→ Ans.

XOR		
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

grey code ⟷ binary

M.S.B remains same

$b_3\ b_2\ b_1\ b_0$

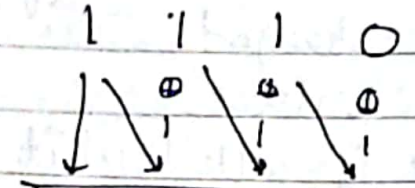
$g_3 = b_3$

$g_2 = b_2 \oplus b_3$

$g_1 = b_1 \oplus b_2$

$g_0 = b_0 \oplus b_1$

Q- 1 1 1 0

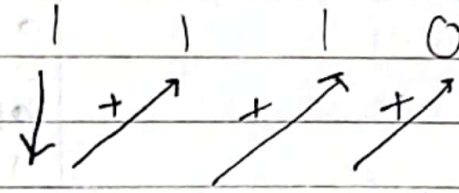


→ G.C

GRAY TO BINARY CONVERSION:

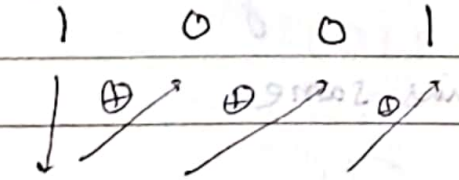
- Record MSB as it is
- Add MSB to next bit of gray code, record the sum, neglect carry ⇒ or XOR
- Repeat the process.

Eg:



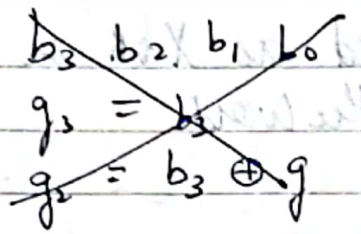
→ Binary

Q.



→ Binary

→ Generalising →



$$\begin{cases} g_3\ g_2\ g_1\ g_0 \\ b_3 = g_3 \\ b_2 = b_3 \oplus g_2 \\ b_1 = b_2 \oplus g_1 \\ b_0 = b_1 \oplus g_0 \end{cases}$$

ERROR DETECTING & CORRECTING CODES:-

PARITY :→

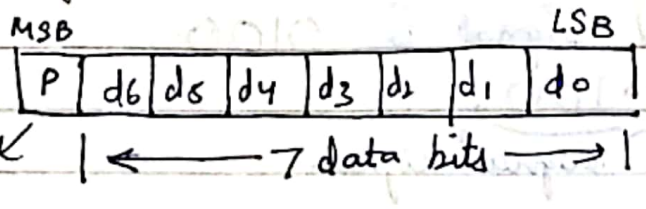
→ To detect and correct the errors additional bits are added to the data bits, at the time of transmission

→ The additional bits are called parity bits. They allow detection and correction of errors.

→ The data bits along with parity bits forms code word

→ PARITY is used to detect errors.

→ Single bit error is detected by it.
(This extra bits tells us about the total number of 1's in the data bits.)



- Parity bit can be computed as X-OR or X-NOR of all the other bits in the word

- Even Parity:
The resulting word after adding parity should have even number of 1's.

If transmitted signal is : 0100
To number of 1 is one, which is odd so parity bit will be 1.

$$\underbrace{1}_{\text{parity bit}}\ \underbrace{0100}_{\text{original signal}}$$

If transmitted signal is : 1100
There are 2 → 1's which means there are even number of 1's. So parity bit will be odd.

$$\underbrace{0}_{\text{parity bit}}\ \underbrace{1100}_{\text{original signal}}$$

- Odd Parity:
The resulting word after adding parity should have odd number of 1's

If transmitted signal is : 0100

$$\underbrace{0}_{\text{parity}}\ \underbrace{0100}_{\text{original signal}}$$

If the transmitted signal is : 1100

1 1100
parity original signal

Error detection:

Suppose transmitted signal is : 1 0100 (even parity)

Now if any of the bits get corrupted say,

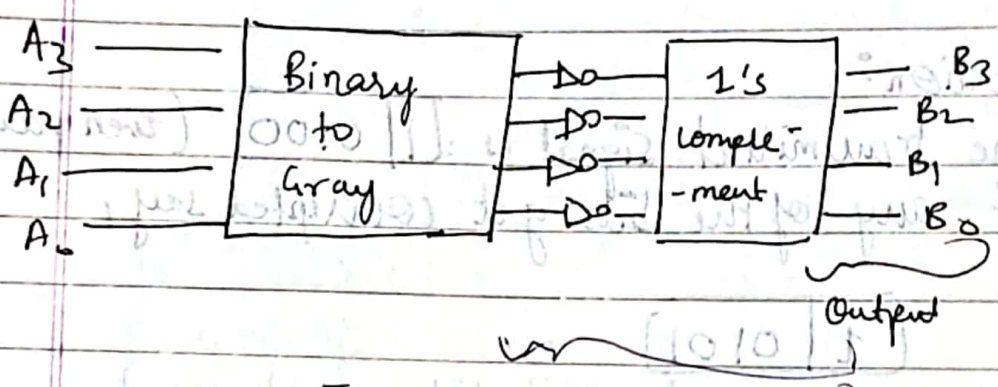
1 0101
↑ This bit got changed.

The data signal has become odd parity. But the receiver knows there is an error because the received signal is different from the expected parity which was even.

→ Parity checking at the receiver can detect the presence of an error, if the parity of the receiver signal is different from expected parity. That means, if it is known that the parity of the transmitted signal is going to be "even" and if the received signal has odd parity, the the receiver can conclude that the received sign is not correct.

- If an error is detected, then the receiver will ignore the received byte and request for re-transmission of the same byte to the transmitter.

Q. If the input binary number $A_3A_2A_1A_0$ is equal to the output $B_3B_2B_1B_0$ & both are non-zero, then the decimal equivalent of $A_3A_2A_1A_0$ is?



Effect of not gates is nullified by the 1's complement. So the answer would be those numbers whose binary is equal to gray code. The are:

0	0000	0000
1	0001	0001
	(Binary)	(Gray)

But since we can't take 0, as stated in Q, answer is $\boxed{1}$

Hamming Code:

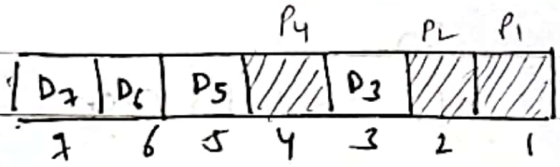
→ Easy to implement
→ 7-bit hamming code is used normally.
→ Given by R.W. Hamming

Rules:

1) Data bits ⇒ 4 (In case of 7 bit hamming code)
2) Parity bits ⇒ 3

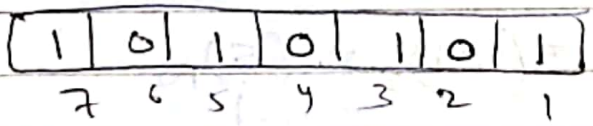
Position of Parity bits: 2^n (where $n = 0, 1, 2 \cdots n$)

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$



P_1 depends on : $D_3 D_5 D_7$
(Depending on even or odd parity)
P_2 : $D_3 D_6 D_7$
P_4 : $D_5 D_6 D_7$

Eg: Data: 1011 by Even Parity



Suppose received data is: 1110101

$P_1 = 1$ ✓
$P_2 = 0$ ✗ Error in 3, 6, 7
$P_4 = 0$ ✗ Error in 5, 6, 7

Q. If the 7-bit hamming code word received by a receiver is 1011011. Assuming then even parity State whether the received code word is correct or wrong. If wrong locate the bit having error.

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1	1	0	1	1

P_4	D_5	D_6	D_7	
1	1	0	1	→ 1's odd

∴ error

$P_4 = 1$ (Contradiction)

P_2	D_3	D_6	D_7	
1	0	0	1	→ 1's even

∴ No error

$P_2 = 0$ (∵ No contradiction)

P_1	D_3	D_5	D_7	
1	0	1	1	→ 1's odd

∴ error

$P_1 = 1$

$P_4 P_2 P_1 = (101)_2 = (5)_{10}$

↓

5th bit is having error

D_5 should be 0

∴ 1001011 → Correct code

Date / /
Page No.

Calculation of number of redundant (Parity) Bits :→

No. of information bits = D
No. of Parity Bits = P

Eqn should satisfy:

$$D + P + 1 \leq 2^P$$

Eg: D = 4

Taking P = 2

$4 + 2 + 1 \leq 2^2 = 4$

$7 \not\leq 4$

Taking P = 3

$4 + 3 + 1 \leq 2^3$

$8 \leq 8$

∴ For D = 4, P = 3 is taken