

UNIT- 1

NUMBER SYSTEMS & CODES

TOPIC 1.3

CODES

CODES

- In the coding, when numbers, letters or words are represented by a specific group of symbols, it is said that the number, letter or word is being encoded.
- The group of symbols is called as a code.
- The digital data is represented, stored and transmitted as group of binary bits. This group is also called as **binary code**.
- The binary code is represented by the number as well as alphanumeric letter.

Advantages of Binary Code

Following is the list of advantages that binary code offers.

- Binary codes are suitable for the computer applications.
- Binary codes are suitable for the digital communications.
- Binary codes make the analysis and designing of digital circuits if we use the binary codes.
- Since only 0 & 1 are being used, implementation becomes easy.

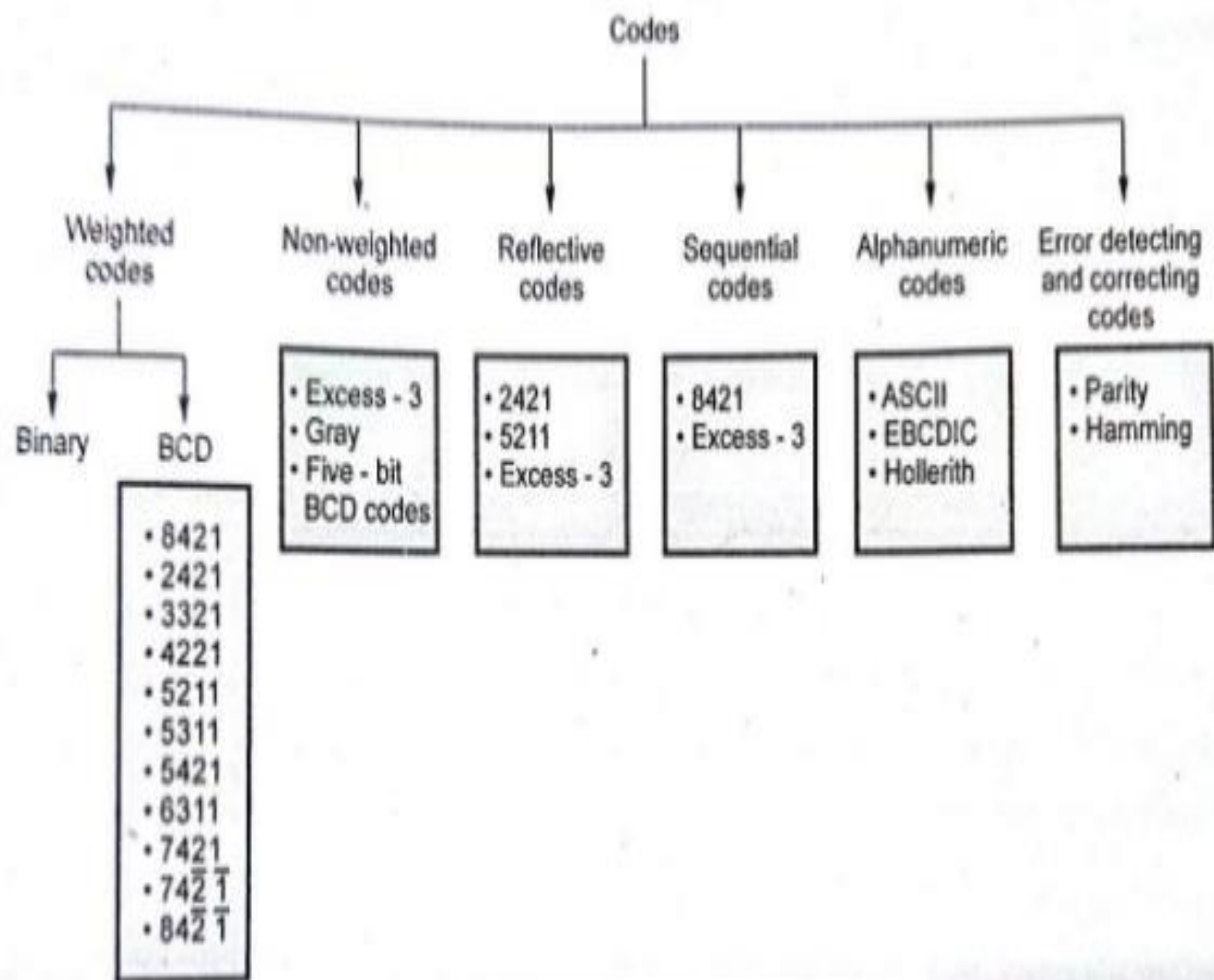
Classification of binary codes

The codes are broadly categorized into following four categories.

- Weighted Codes
- Non-Weighted Codes
- Binary Coded Decimal Code
- Alphanumeric Codes
- Error Detecting Codes
- Error Correcting Codes

CODES

- **Gray Code**
- **BCD Code**
- **Excess -3 Code**
- **ASCII Code**
- **Error Detection and Correction: Hamming Codes**



Weighted Codes

Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the number represents a specific weight. Several systems of the codes are used to express the decimal digits 0 through 9. In these codes each decimal digit is represented by a group of four bits.

Weighted Code

Non-Weighted Codes

In this type of binary codes, the positional weights are not assigned. The examples of non-weighted codes are Excess-3 code and Gray code.

Binary Coded Decimal (BCD) code

In this code each decimal digit is represented by a 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). The remaining six code combinations i.e. 1010 to 1111 are invalid in BCD.

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

Advantages of BCD Codes

- It is very similar to decimal system.
- We need to remember binary equivalent of decimal numbers 0 to 9 only.

Disadvantages of BCD Codes

- The addition and subtraction of BCD have different rules.
- The BCD arithmetic is little more complicated.
- BCD needs more number of bits than binary to represent the decimal number. So BCD is less efficient than binary.

Binary Coded Decimal (BCD)

- Conversion of big Decimal numbers to BCD-
- Decimal Number-98 - 1001 1000
- Decimal Number 164 - 0001 0110 0100
- 78 –binary –(1001110) –(7 bits)
BCD - 0111 1000 (8 bits)

Convert following Decimal to BCD

1. 35 - 0011 0101
2. 174 - 0001 0111 0100
3. 2479 - 0010 0100 0111 1001

Decimal number	Binary number	Binary Coded Decimal(BCD)
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111
8	1000	1000
9	1001	1001
10	1010	0001 0000
11	1011	0001 0001
12	1100	0001 0010
13	1101	0001 0011
14	1110	0001 0100
15	1111	0001 0101

BCD ARITHMETIC

1

- BCD ADDITION

2

- BCD SUBTRACTION

BCD ARITHMETIC

1

- BCD ADDITION

BCD ADDITION

Case 1- Sum is equal to or less than 9 and carry is 0

Case 2- Sum is Greater than 9 and carry is 0

Case 3- Sum is equal to or less than 9 and carry is 1

Case 1- Sum is equal to or less than 9 and carry is 0

Eg.-Addition of 2 and 6 in BCD

Decimal	BCD
2	0 0 1 0
+ 6	+ 0 1 1 0
-----	1 1
8	-----
	0 1 0 0 0

Final Carry zero

sum is a valid BCD Number

---- carry bit

Case 2- Sum is Grater than 9 and carry is 0

-in this case sum is grater than 9 that means it is an invalid BCD Number, but the final carry is zero.

- we have to correct the sum by adding decimal 6 or BCD 0110 to it. after doing this we get the correct BCD sum.

Eg.-Addition of 7 and 6 in BCD

Decimal	BCD
7	0 1 1 1
+ 6	0 1 1 0
-----	<u>1 1</u>
13	1 1 0 1 ← Invalid BCD No. with carry 0
	0 1 1 0 ← Add 6 for correction
	<u>1 0 0 1 1</u> ← Valid BCD no. with Carry =1
	0001 0011 valid BCD Result

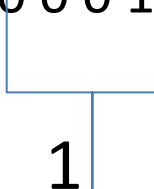
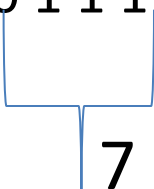
- Case 3-** Sum is equal to or less than 9 and carry is 1
- A nonzero carry indicates that the answer is wrong and needs correction.
 - Add 6 or (0110) BCD to the sum to correct the answer.

Eg.Addition of 9 and 8 in BCD

Decimal	BCD
9	1 0 0 1
+ 8	1 0 0 0

-----	-----	
17	1 0 0 0 1	← Incorrect BCD result
Final Carry one		

Add 6 to Incorrect BCD result-

0 0 0 1	0 0 0 1	
+0 0 0 0	0 1 1 0	
<hr/>		
0 0 0 1	0 1 1 1	sum is a valid BCD Number
		
1	7	

Add 57 and 26 in BCD = ??

BCD ARITHMETIC

2

- BCD SUBTRACTION

- 9's complement method
- 10's complement method

- BCD SUBTRACTION

9's complement method-

-the 9's complement of a BCD number can be obtained by subtracting it from 9.

-for eg. 9's complement of 1 is 8. the 9's complement of various digits are given table

Digital Digit	0	1	2	3	4	5	6	7	8	9
9's Complement	9	8	7	6	5	4	3	2	1	0

- **BCD SUBTRACTION**

Eg.A-B

Step 1-Obtain 9's complement of number B.

Step 2 – Add A and 9's complement of B.

Step 3 – if a carry is generated in step 2 then add it to the sum to obtain the final result. The carry is called as end around carry.

Step 4 – if carry is not produced then the result is negative hence take the 9's complement of the result.

Perform 83 -21 using 9's complement

Step 1- 9's complement of 21-

$$\begin{array}{r} 99 \\ -21 \\ \hline 78 \end{array}$$

Step 2- add 83 and 9's complement of 21

83	1000	0011
+78	0111	1000
<hr/>		
161	1111	1011 – Invalid BCD

Step 3- Add 6 to each invalid BCD number

1111	1011
0110	0110
1111	11
<hr/>	
1	0110 0001

Step 4 –

Add carry to sum obtained in step 3-

0110	0001
+	1
<hr/>	
0110	0010

-- (62)

83-21=62

**Perform 52 -89 using 9's
complement(BCD Subtraction)**

52-89=??

- **BCD SUBTRACTION – 10's Complement Method**

Eg .A-B

Step 1-Obtain 10's complement of number B.

Step 2 – Add A and 10's complement of B.

Step 3 – if a carry is generated in step 2 then Discard carry, then answer is positive and in its true form.

Step 4 – if carry is not produced then the result is negative, hence take the 10's complement of the result.

22 -54=-32

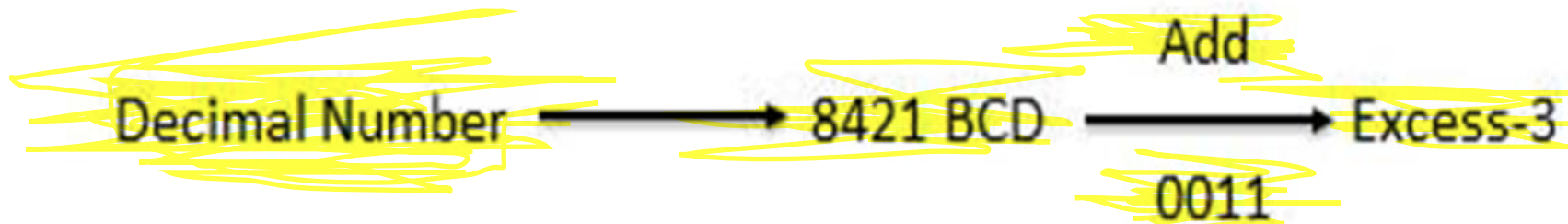
**Perform 3 -8 using 10's complement
(BCD Subtraction)**

3-8 =??

Excess-3 Code

Excess-3 code-

The Excess-3 code is also called as XS-3 code. It is **non-weighted code** used to express decimal numbers. The Excess-3 code words are derived from the 8421 BCD code words adding $(0011)_2$ or $(3)_{10}$ to each code word in 8421. The excess-3 codes are obtained as follows –



Excess-3 Code

Decimal	BCD				Excess-3			
	8	4	2	1	BCD + 0011			
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	1	1	0	0

Sequential code

Eg. 1-Obtain the XS-3 Code for (428)-

	4	2	8	Decimal
	0100	0010	1000	BCD
+3	0011	0011	0011	

	0111	0101	1011	

Eg. 2- 247.6

	2	4	7	.	6	Decimal
	0010	0100	0111	.	0110	BCD
+3	0011	0011	0011	.	0011	

	0101	0111	1010	.	1001	

Gray Code

Gray Code-

- It is the **non-weighted code** and it is not arithmetic codes. That means there are no specific weights assigned to the bit position.
- It has a very special feature that, only one bit will change each time the decimal number is incremented as shown in fig. **As only one bit changes at a time, the gray code is called as a unit distance code.**
- **The gray code is a cyclic code. Gray code cannot be used for arithmetic operation.**

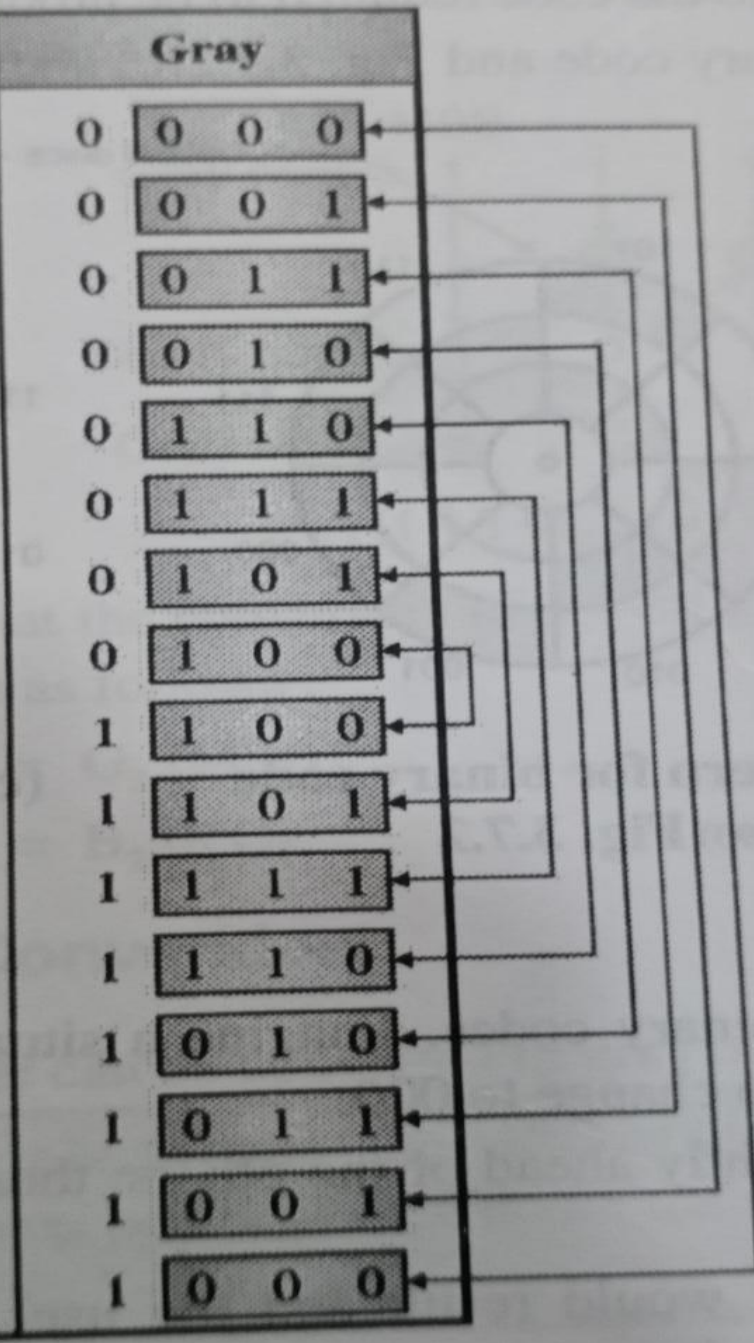
Decimal numbers	Binary code	Gray code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Reflective Property

Decimal	Gray	
0	0 0	0 0
1	0 0	0 1
2	0 0	1 1
3	0 0	1 0
4	0 1	1 0
5	0 1	1 1
6	0 1	0 1
7	0 1	0 0

Reflective Property

Decimal	Gray
0	0 0 0 0
1	0 0 0 1
2	0 0 1 1
3	0 0 1 0
4	0 1 1 0
5	0 1 1 1
6	0 1 0 1
7	0 1 0 0
8	1 1 0 0
9	1 1 0 1
10	1 1 1 1
11	1 1 1 0
12	1 0 1 0
13	1 0 1 1
14	1 0 0 1
15	1 0 0 0



Gray Code

How to Convert Binary to Gray Code

$$0 \oplus 0 = 0$$

$$0 \oplus 1 = 1$$

$$1 \oplus 0 = 1$$

$$1 \oplus 1 = 0$$

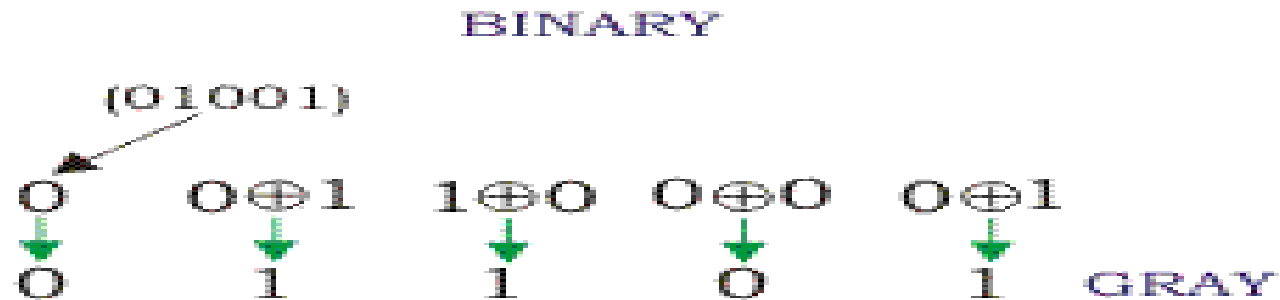
Step 1-

1. The MSB (Most Significant Bit) of the gray code will be exactly equal to the first bit of the given binary number.

How to Convert Binary to Gray Code

Step 2 -The second bit of the code will be exclusive-or (XOR) of the first and second bit of the given binary number, i.e if both the bits are same the result will be 0 and if they are different the result will be 1.

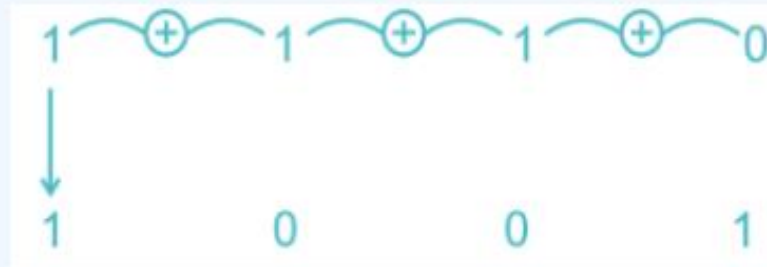
Step 3 -The third bit of gray code will be equal to the exclusive-or (XOR) of the second and third bit of the given binary number. Thus the binary to gray code conversion goes on. An example is given below to illustrate these steps.



Binary number -1110 convert to gray

Example:

Binary Code: $b_3 \ b_2 \ b_1 \ b_0 = 1 \ 1 \ 1 \ 0$ Gray Code: $g_3 \ g_2 \ g_1 \ g_0$



$$g_3 = b_3 = 1$$

$$g_2 = b_3 \oplus b_2 = 1 \oplus 1 = 0$$

$$g_1 = b_2 \oplus b_1 = 1 \oplus 1 = 0$$

$$g_0 = b_1 \oplus b_0 = 1 \oplus 0 = 1$$

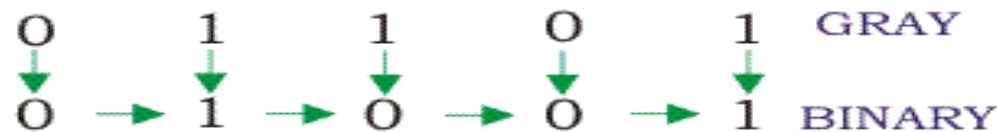
\therefore Final Gray code: 1 0 0 1

Convert $(25)_{10}$ in gray code = ??

Gray Code to Binary Conversion

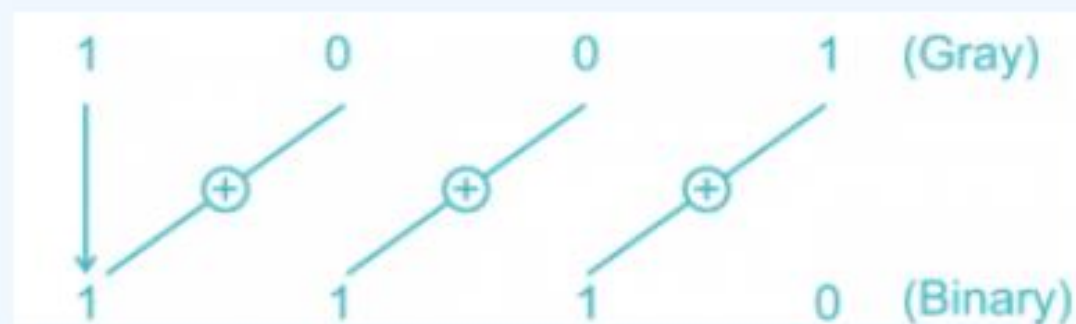
Gray code to binary conversion is again a very simple and easy process. Following steps can make your idea clear on this type of conversions.

1. The MSB of the binary number will be equal to the MSB of the given gray code.
2. Now if the second gray bit is 0, then the second binary bit will be the same as the previous or the first bit. If the gray bit is 1 the second binary bit will alter. If it was 1 it will be 0 and if it was 0 it will be 1.
3. This step is continued for all the bits to do **Gray code to binary conversion**.



Example:

Gray Code: $g_3 \ g_2 \ g_1 \ g_0 = 1 \ 0 \ 0 \ 1$ then **Binary Code:** $b_3 \ b_2 \ b_1 \ b_0$



$$b_3 = g_3 = 1$$

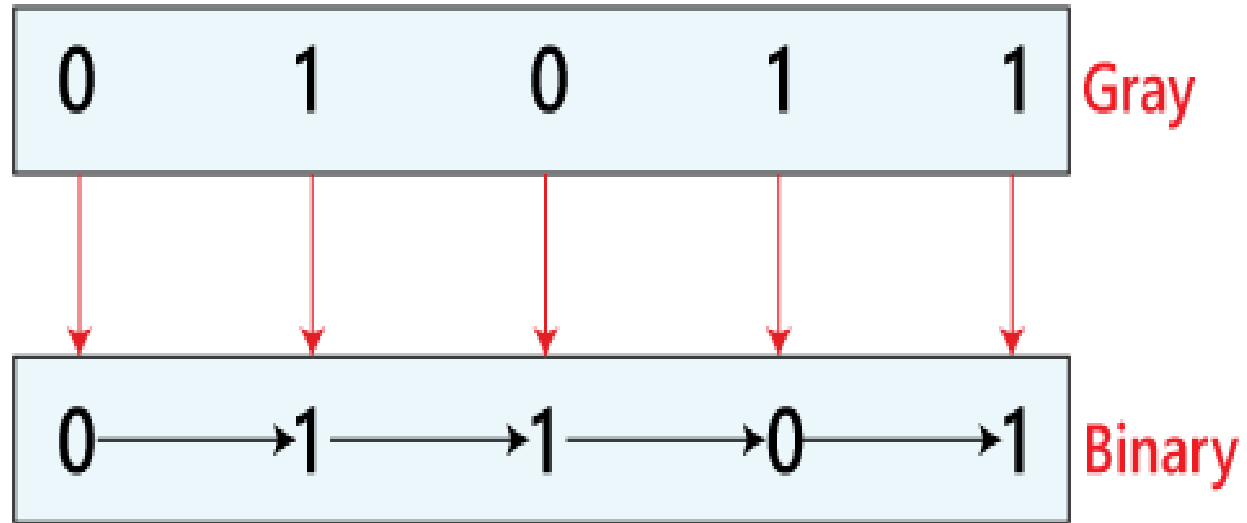
$$b_2 = b_3 \oplus g_2 = 1 \oplus 0 = 1$$

$$b_1 = b_2 \oplus g_1 = 1 \oplus 0 = 1$$

$$b_0 = b_1 \oplus g_0 = 1 \oplus 1 = 0$$

\therefore Final Binary Code: 1 1 1 0

Convert 01011 gray to binary



Convert (1110) gray to binary= ??

Alphanumeric Codes

Alphanumeric codes

A binary digit or bit can represent only two symbols as it has only two states '0' or '1'. But this is not enough for communication between two computers because there we need many more symbols for communication.

These symbols are required to represent 26 alphabets with capital and small letters, numbers from 0 to 9, punctuation marks and other symbols.

- American Standard Code for Information Interchange (ASCII).
- Extended Binary Coded Decimal Interchange Code (EBCDIC).

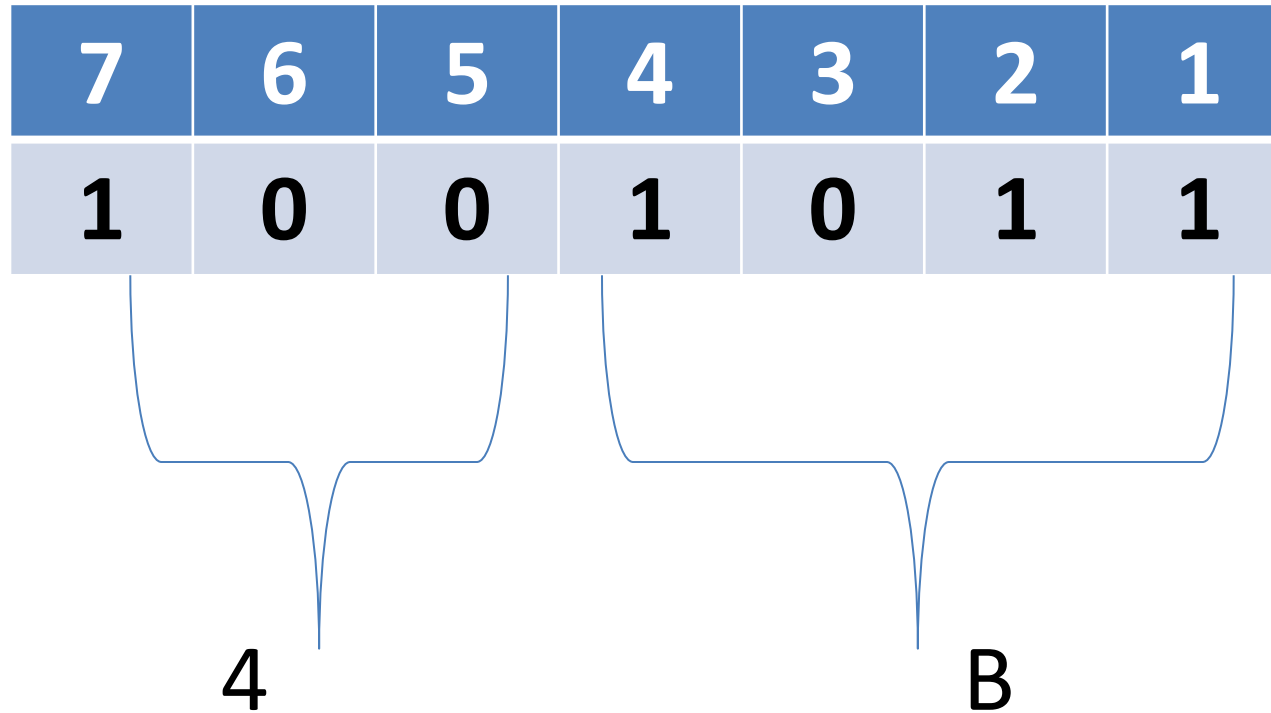
ASCII code is a 7-bit code whereas EBCDIC is an 8-bit code. ASCII code is more commonly used worldwide while EBCDIC is used primarily in large IBM computers.

USASCII code chart

<div> <div> <div>b7</div> <div>b6</div> <div>b5</div> <div>b4</div> <div>b3</div> <div>b2</div> <div>b1</div> </div> <div>Bits</div> </div> <div> <div>Column</div> <div>Row</div> </div>					000	001	010	011	100	101	110	111
					0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	\$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(8	H	X	h	x
1	0	0	1	9	HT	EM)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	
1	1	0	1	13	CR	GS	-	=	M]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Using ASCII Table code word for character “K”

ASCII Code -



Using ASCII Table code word for character “ f ” =??

Extended ASCII Characters

128	Ç	144	É	160	á	176	⌘	193	⊥	209	⌘	225	ß	241	±
129	ü	145	æ	161	í	177	⌘	194	⌘	210	⌘	226	Γ	242	≥
130	é	146	Æ	162	ó	178	⌘	195	⌘	211	⌘	227	π	243	≤
131	â	147	ô	163	ú	179		196	—	212	⌘	228	Σ	244	∫
132	ä	148	ö	164	ñ	180	⌘	197	⌘	213	⌘	229	σ	245	∫
133	à	149	ò	165	Ñ	181	⌘	198	⌘	214	⌘	230	μ	246	÷
134	â	150	û	166	ª	182	⌘	199	⌘	215	⌘	231	τ	247	≈
135	ç	151	ù	167	º	183	⌘	200	⌘	216	⌘	232	Φ	248	°
136	ê	152	—	168	¿	184	⌘	201	⌘	217	⌘	233	Θ	249	·
137	ë	153	Ö	169	—	185	⌘	202	⌘	218	⌘	234	Ω	250	·
138	è	154	Ü	170	¬	186	⌘	203	⌘	219	■	235	δ	251	√
139	ï	156	£	171	½	187	⌘	204	⌘	220	■	236	∞	252	—
140	î	157	¥	172	¼	188	⌘	205	=	221	■	237	φ	253	²
141	ì	158	—	173		189	⌘	206	⌘	222	■	238	ε	254	■
142	Ä	159	ƒ	174	«	190	⌘	207	⌘	223	■	239	∧	255	
143	Å	192	Ł	175	»	191	⌘	208	⌘	224	α	240	≡		

Code Conversions-

1. Binary to BCD
2. BCD to Binary
3. BCD to Excess-3
4. Excess-3 to BCD

Binary to BCD Conversion

Step 1- Convert Binary Number to Decimal

Step 2 – Convert Decimal number into BCD

Eg – Convert $(110101)_2$ into BCD

Step1 = $2^5 + 2^4 + 0 + 2^2 + 0 + 1 = (53)_{10}$

Step 2 = Convert into BCD - 0101 0011

Eg. Convert (11101) into BCD = ??

BCD to Binary Conversion

Step 1- Convert BCD Number to Decimal

Step 2 – Convert Decimal number to Binary

Eg – Convert (0101 0011)BCD into Binary

Step1 =(53)₁₀

Step 2 = Convert into Binary - 110101

Eg. Convert (0011 0101)BCD into Binary = ??

BCD to Excess -3 Conversion

Step 1- Convert BCD Number to Decimal.

Step 2 – Add $(3)_{10}$ to this decimal number.

Step 3 – Convert Decimal number of step2 into Binary, to get the excess -3 code.

Eg. – Convert $(1001)_{\text{BCD}}$ into excess -3

Step1 = $1001 = 9$

Step 2 = $9+3 = 12$

Convert into Binary = $(1100)_2$

$(1001)_{\text{BCD}} = (1100)_{\text{XS-3}}$

Eg. Convert $(0101 \ 0011)_{\text{BCD}}$ into excess -3 = ??

Excess -3 to BCD Conversion

Step 1 – Subtract given number from (0011), so ans is your BCD number

Eg. – Convert $(1001\ 1010)_{XS-3}$ to BCD

$$\begin{array}{r} 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0 \\ -\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1 \\ \hline 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 \\ \hline 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1 \end{array} \quad \text{----BCD}$$

Eg. Convert $(0110\ 0111)_{XS-3}$ into BCD = ??

Q1) Convert $(47.3)_7$ into BCD , Excess -3 and Gray Code. (may 2016 , 3 marks)

=??

Q1) Convert $(47.3)_{10}$ into BCD, Excess -3 and Gray Code.

decimal no = $28 + 7 + 0.4285 = (35.4285)_{10}$

Step 2 – conversion to BCD

3 5 . 4 2 8 5

0011 0101 . 0100 0010 1000 0101 ---BCD

STEP 3-

Conversion to excess 3- Decimal + 3 = 3 5 . 4 2 8 5 = 6 8 . 7 5 11 8

= (0110 1000 . 0111 0101 1011 1000)_{xs -3}

Step 4 – conversion to binary-

$35 = (100011)_2$ and $0.4285 = (0.01101)_2$

$(35.4285)_{10} = (100011.01101)_2$

Step 5 – conversion to gray = binary to gray

1 0 0 0 1 1 . 0 1 1 0 1

1 1 0 0 1 0 . 1 1 0 1 1 gray

Error Detection & Correction Codes

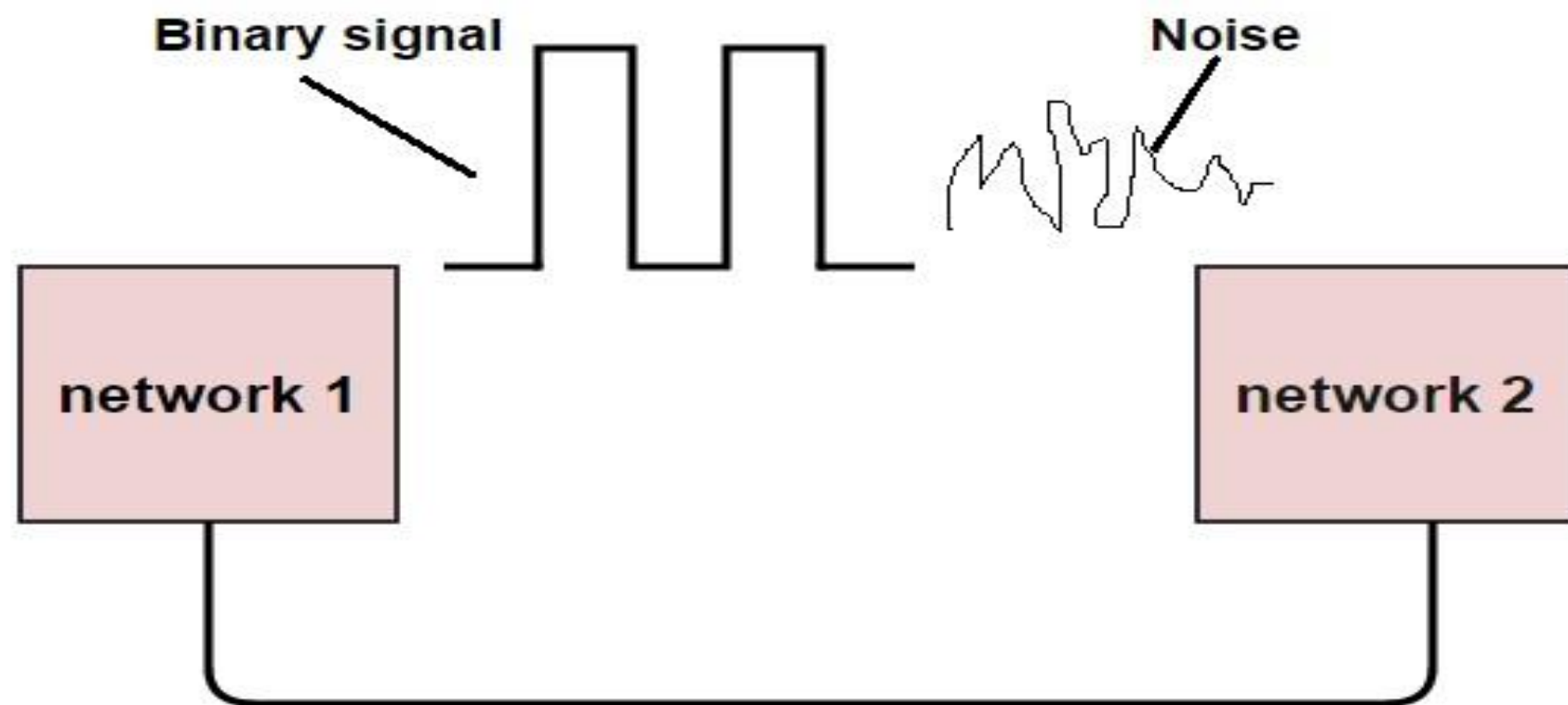
In digital systems, the analog signals will change into digital sequence (in the form of bits). This sequence of bits is called as “Data stream”. The change in position of single bit also leads to catastrophic (major) error in data output. Almost in all electronic devices, we find errors and we use error detection and correction techniques to get the exact or approximate output.

Error Detection & Correction Codes

What is an Error

The data can be corrupted during transmission (from source to receiver). It may be affected by external noise or some other physical imperfections. In this case, the input data is not same as the received output data. This mismatched data is called “Error”.

The data errors will cause loss of important / secured data. Even one bit of change in data may affect the whole system's performance. Generally the data transfer in digital systems will be in the form of ‘Bit – transfer’. In this case, the data error is likely to be changed in positions of 0 and 1 .



Types of Error detection

1. Parity Checking
2. Cyclic Redundancy Check (CRC)
3. Check Sum

Parity Checking

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 or correction of the errors.

The data bits along with the parity bits form a code word.

Parity Checking of Error Detection

It is the simplest technique for detecting and correcting errors. The MSB of an 8-bits word is used as the parity bit and the remaining 7 bits are used as data or message bits. The parity of 8-bits transmitted word can be either even parity or odd parity.



Even parity -- Even parity means the number of 1's in the given word including the parity bit should be even (2,4,6,...).

Odd parity -- Odd parity means the number of 1's in the given word including the parity bit should be odd (1,3,5,...)

. Use of Parity Bit

The parity bit can be set to 0 and 1 depending on the type of the parity required.

For even parity, this bit is set to 1 or 0 such that the no. of "1 bits" in the entire word is even. Shown in fig. (a).

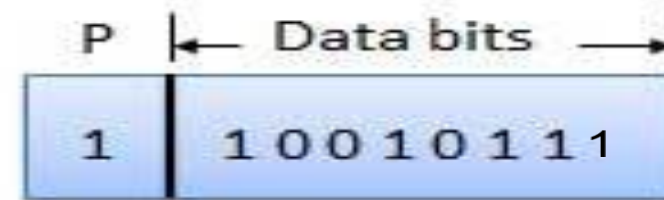
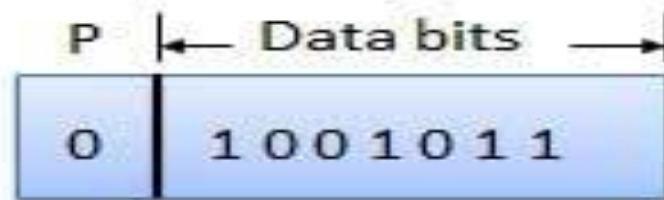


Fig. (a)

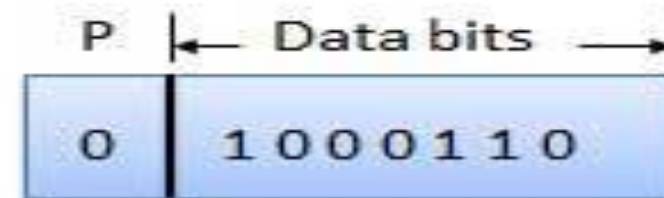
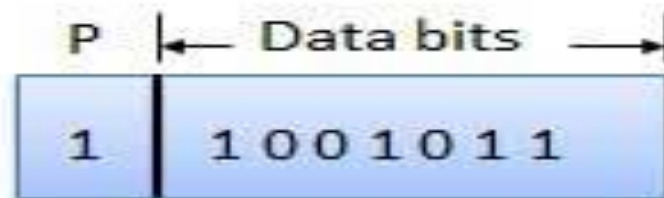
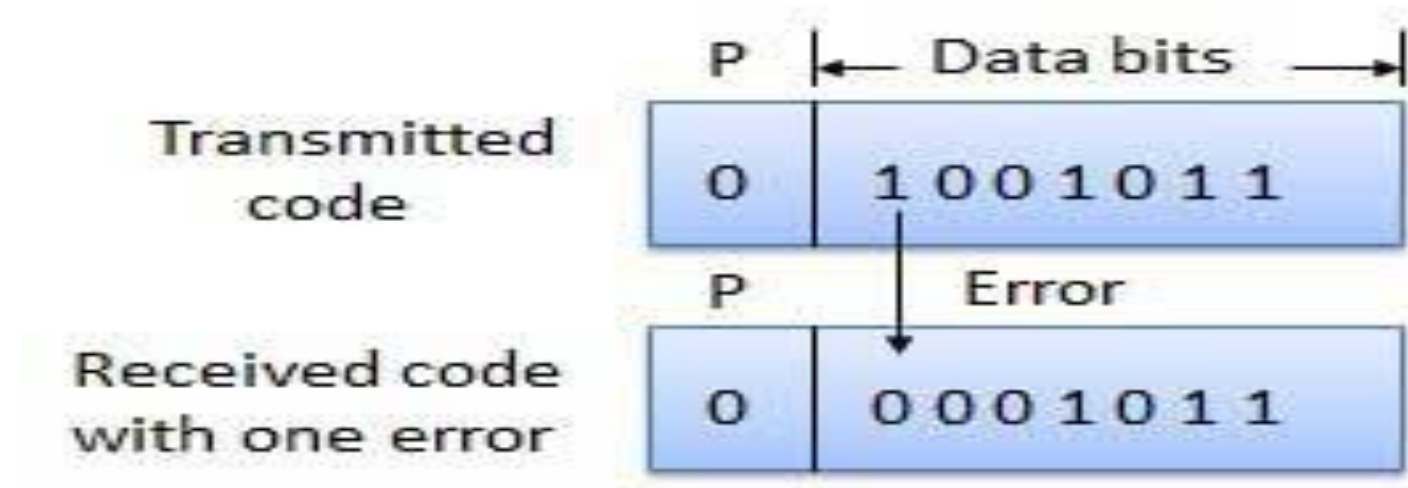


Fig. (b)

How Does Error Detection Take Place?

Parity checking at the receiver can detect the presence of an error if the parity of the receiver signal is different from the expected parity. That means, if it is known that the parity of the transmitted signal is always going to be "even" and if the received signal has an odd parity, then the receiver can conclude that the received signal is not correct. If an error is detected, then the receiver will ignore the received byte and request for retransmission of the same byte to the transmitter.



NOTE: The counting of data bits will include the parity bit also.

3 bit data			Message with even parity		Message with odd parity	
A	B	C	Message	Parity	Message	Parity
0	0	0	000	0	000	1
0	0	1	001	1	001	0
0	1	0	010	1	010	0
0	1	1	011	0	011	1
1	0	0	100	1	100	0
1	0	1	101	0	101	1
1	1	0	110	0	110	1
1	1	1	111	1	111	0

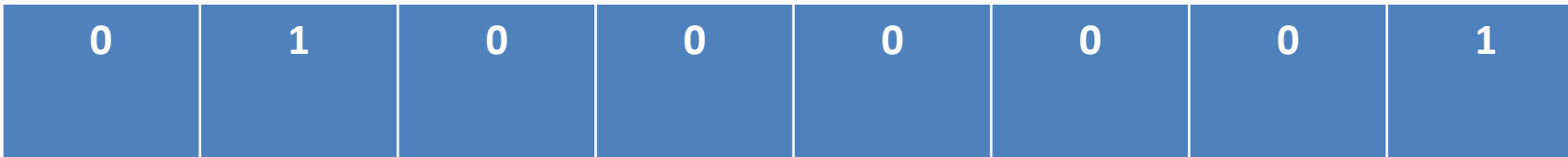
Limitations of parity Checking –

- 1.Simple parity checking method has its limitations. its not suitable for detection of multiple errors (two, four, six etc)**
- 2.It cannot reveal the location of error bit. It can't correct the error either**

Error Detection using ASCII Code

Basically ASCII is a 7 bit code. But in order to detect errors in data communication and processing ,an eight bit is sometimes added to the ASCII character to indicate its parity

A= 41 =1000001(7 bit ascii)(EVEN Parity)

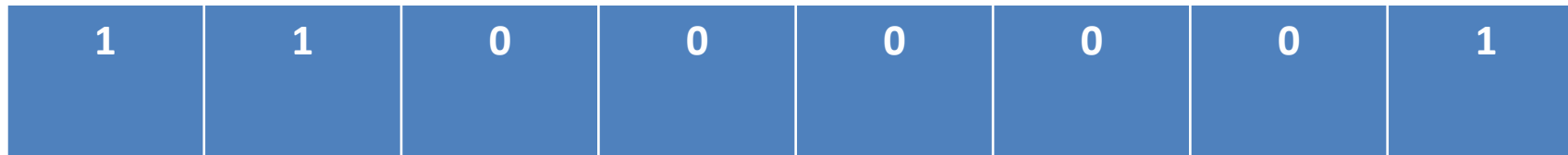


Parity bit

7 bit ASCII

Error Detection using ASCII Code

A= 41 =1000001(7 bit ascii)(ODD Parity)



Parity bit

7 bit ASCII

Error Detection using ASCII Code

H= 1001000(7 bit ascii) (EVEN Parity)

O= 1001111

L= 1001100

E= 1000101

	Parity bit	Data bit 7	Data bit 6	Data bit 5	Data bit 4	Data bit 3	Data bit 2	Data bit 1
H	0	1	0	0	1	0	0	0
O	1	1	0	0	1	1	1	1
L	1	1	0	0	1	1	0	0
E	1	1	0	0	0	1	0	1



Parity bit

Hamming Code

This error detecting and correcting code technique is developed by R.W.Hamming . This code not only identifies the error bit, in the whole data sequence and it also corrects it. This code uses a number of parity bits located at certain positions in the codeword. The number of parity bits depends upon the number of information bits. The hamming code uses the relation between redundancy bits and the data bits and this code can be applied to any number of data bits. (2 types – 7 bit & 15 bit)

What is a Redundancy Bit?

Redundancy means “The difference between number of bits of the actual data sequence and the transmitted bits”. These redundancy bits are used in communication system to detect and correct the errors, if any.

How the Hamming code actually corrects the errors?

In Hamming code, the redundancy bits are placed at certain calculated positions in order to eliminate errors. The distance between the two redundancy bits is called “Hamming distance”.

To understand the working and the data error correction and detection mechanism of the hamming code, let's see to the following stages.

Hamming Code

Number of parity bits

As we learned earlier, the number of parity bits to be added to a data string depends upon the number of information bits of the data string which is to be transmitted. Number of parity bits will be calculated by using the data bits. This relation is given below.

$$2^P \geq n + P + 1$$

Here, n represents the number of bits in the data string.

P represents number of parity bits.

For example, if we have 4 bit data string, i.e. $n = 4$, then the number of parity bits to be added can be found by using trial and error method. Let's take $P = 2$, then

$$2^P = 2^2 = 4 \text{ and } n + P + 1 = 4 + 2 + 1 = 7$$

This violates the actual expression.

So let's try $P = 3$, then

$$2^P = 2^3 = 8 \text{ and } n + P + 1 = 4 + 3 + 1 = 8$$

So we can say that 3 parity bits are required to transfer the 4 bit data with single bit error correction.

Hamming Code

Where to Place these Parity Bits?

After calculating the number of parity bits required, we should know the appropriate positions to place them in the information string, to provide single bit error correction.

In the above considered example, we have 4 data bits and 3 parity bits. So the total codeword to be transmitted is of 7 bits (4 + 3). We generally represent the data sequence from right to left, as shown below.

bit 7, bit 6, bit 5, bit 4, bit 3, bit 2, bit 1, bit 0

The parity bits have to be located at the positions of powers of 2. I.e. at 1, 2, 4, 8 and 16 etc. Therefore the codeword after including the parity bits will be like this

D7, D6, D5, P4, D3, P2, P1

Here P1, P2 and P4 are parity bits & D3,D5,D6,D7 are data bits.

Hamming Code

Constructing a Bit Location Table

In Hamming code, each parity bit checks and helps in finding the errors in the whole code word. So we must find the value of the parity bits to assign them a bit value.

7 bit Hamming code-

Bit Designation	D7	D6	D5	P4	D3	P2	P1
Bit Location	7	6	5	4	3	2	1
Binary Location Number	111	110	101	100	011	010	001
Data Bits (Dn)				---		---	---
Parity Bits (Pn)	---	---	---		---		

Hamming Code

By calculating and inserting the parity bits in to the data bits, we can achieve error correction through Hamming code.

Let's understand this clearly, by looking into an example.

Ex:

Encode the data 1101 in odd parity, by using Hamming code.

Step 1

Calculate the required number of parity bits.

Let $P = 2$, then

$$2^P = 2^2 = 4 \text{ and } n + P + 1 = 4 + 2 + 1 = 7.$$

2 parity bits are not sufficient for 4 bit data.

So let's try $P = 3$, then

$$2^P = 2^3 = 8 \text{ and } n + P + 1 = 4 + 3 + 1 = 8$$

Therefore 3 parity bits are sufficient for 4 bit data.

The total bits in the code word are $4 + 3 = 7$

Step 2

Constructing bit location table

Hamming Code

1	Bit Designation	D7	D6	D5	P4	D3	P2	P1
	Bit Location	7	6	5	4	3	2	1
	Binary Location Number	111	110	101	100	011	010	001
	Data Bits (D_n)	1	1	0		1		
	Parity Bits (P_n)				1		0	1

Step 3

Determine the parity bits.

For P1 : 3, 5 and 7 bits are having two 1's so for odd parity, $P1 = 1$.

For P2 : 3, 6 and 7 bits are having three 1's so for odd parity, $P2 = 0$.

For P4 : 5, 6 and 7 bits are having two 1's so for odd parity, $P4 = 1$.

By entering / inserting the parity bits at their respective positions, codeword can be formed and is transmitted. It is 1101101.

Step 4 – Error detection and correction

NOTE: If the codeword has all zeros (ex: 0000000), then there is no error in Hamming code.

To represent the binary data in alphabets and numbers, we use alphanumeric codes.

Eg.-Write hamming Code for number 0111(Assume Even parity)

Soln. : Assume even parity hamming code.

Step 1 :

D ₇	D ₆	D ₅	P ₄	D ₃	P ₂	P ₁
0	1	1		1		

Step 2 : Select P₁ for P₁ D₃ D₅ D₇ :

D ₇	D ₆	D ₅	P ₄	D ₃	P ₂	P ₁
0	1	1		1		0

→ P₁ = 0 for P₁ D₃ D₅ D₇ = 0110

Step 3 : Select P₂ for P₂ D₃ D₆ D₇ :

D ₇	D ₆	D ₅	P ₄	D ₃	P ₂	P ₁
0	1	1		1	0	0

→ P₂ = 0 for P₂ D₃ D₆ D₇ = 0110

Step 4 : Select P₄ for P₄ D₅ D₆ D₇ :

D ₇	D ₆	D ₅	P ₄	D ₃	P ₂	P ₁
0	1	1	0	1	0	0

→ P₄ = 0 for P₄ D₅ D₆ D₇ = 0110

Hence the complete 7-bit hamming code word is as shown below

0	1	1	0	1	0	0
---	---	---	---	---	---	---

← Complete code word

Hamming Code

Example – Write the Hamming Code – 1010 (May 10(3 marks), May 11(4 marks), May 14, Dec 15 -2 marks)

= ????

Hamming Code

Detection and Correction of Errors-

-After Hamming Code Transmitted , at the receiver it is decoded to get the data back

-The Bits checking for Even parity.

Position 1: check 1 bit, skip 1 bit, check 1 bit, skip 1 bit, etc. (1,3,5,7,9,11,13,15,...)

Position 2: check 2 bits, skip 2 bits, check 2 bits, skip 2 bits, etc. (2,3,6,7,10,11,14,15,...)

Position 4: check 4 bits, skip 4 bits, check 4 bits, skip 4 bits, etc.

(4,5,6,7,12,13,14,15,20,21,22,23,...)

Position 8: check 8 bits, skip 8 bits, check 8 bits, skip 8 bits, etc. (8-15,24-31,40-47,...)

-If all the 4 bit groups mentioned above possess the even parity then the received code word is correct i.e. it does not contain Errors.

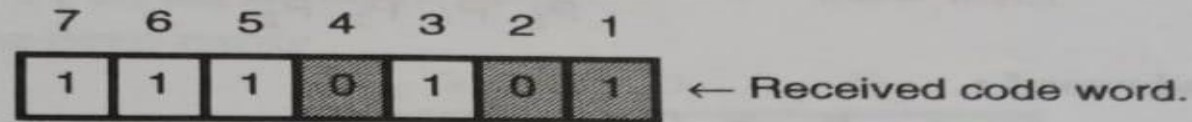
- But if the parity is not even then error exists. Such an error can be located by forming a three bit number out of three parity checks. This process became clear by solving the example

Hamming Code

Detection and Correction of Errors –

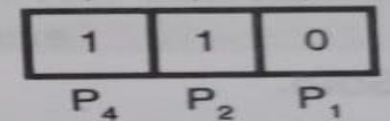
Eg.- A seven bit Hamming Code is received as 1110101. what is the correct code? Assume the parity to be even.

Soln. :



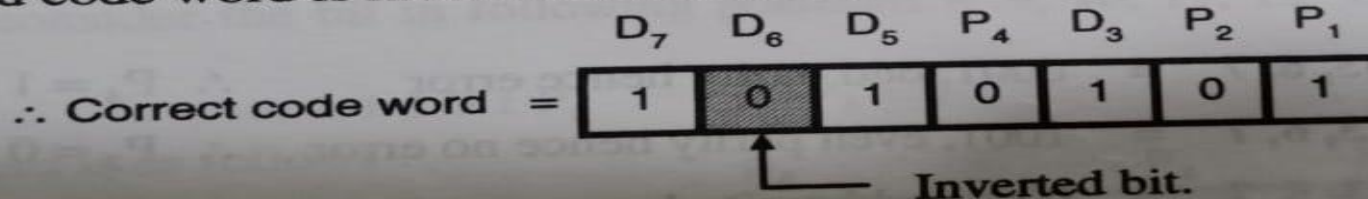
- Step 1 : Check bits 4, 5, 6, 7 : Odd parity hence error $\therefore P_4 = 1$
- Step 2 : Check bits 2, 3, 6, 7 : Odd parity, hence error $\therefore P_2 = 1$
- Step 3 : Check bits 1, 3, 5, 7 : Even parity, hence no error $\therefore P_1 = 0$

\therefore Error word $E =$



Step 4 : Decimal equivalent of $E = 1\ 1\ 0 = (6)_{10}$:

6th bit in the received code word is incorrect. So invert it.



This concept of error detection and correction can be extended to the longer hamming code words as well.

Hamming Code

Example – Suppose that an incoming message 11110101101 is received (Even Hamming Code).

Step 1 – At first the number of redundant bits are calculated using the formula $2^r \geq m + r + 1$. Here, $m + r + 1 = 11 + 1 = 12$. The minimum value of r such that $2^r \geq 12$ is 4.

Step 2 – The redundant bits are positioned as below –

11	10	9	8(r_4)	7	6	5	4(r_3)	3	2(r_2)	1(r_1)
1	1	1	1	0	1	0	1	1	0	1

Step 3 – Even parity checking is done –

$$p_1 = \text{even_parity}(1, 3, 5, 7, 9, 11) = 0$$

$$p_2 = \text{even_parity}(2, 3, 6, 7, 10, 11) = 0$$

$$p_4 = \text{even_parity}(4, 5, 6, 7) = 0$$

$$p_8 = \text{even_parity}(8, 9, 10, 11) = 0$$

Step 4 - Since the value of the check bits $c_1c_2c_3c_4 = 0000 = 0$, there are no errors in this message.

Hamming Code

Example – A 7 bit even parity Hamming code is received as 1000010 , correct it for any Errors and extract 4 bit data is received.

= ????