

## **Number System**

In digital electronics, the number system is used for representing the information. There are two types of number system:

- Non-Positional Number system
- Positional Number system

### **Non-Positional Number System**

- Use symbols such as I for 1, II for 2, III for 3, IIII for 4, IIIII for 5, etc.
- Each symbol represents the same value regardless of its position in the number.
- The symbols are simply added to find out the value of a particular number.
- It is difficult to perform arithmetic with such a number system

## Positional Number System

- Use only a few symbols called digits.
- These symbols represent different values depending on the position they occupy in the number.

**A digital system can understand positional number system only where there are a few symbols called digits and these symbols represent different values depending on the position they occupy in the number.**

A value of each digit in a number can be determined using:

- The digit
- The position of the digit in the number
- The base of the number system (where base is defined as the total number of digits available in the number system).



The positional number system has different bases and the most common of them are :

- Binary
- Decimal
- Octal
- Hexadecimal

### **BINARY NUMBER SYSTEM**

- Has only 2 symbols or digits (0 and 1). Hence its base = 2
- The maximum value of a single digit is 1 (one less than the value of the base)
- Each position of a digit represents a specific power of the base (2)
- This number system is used in computers
- Example :  $(11001)_2$

## DECIMAL NUMBER SYSTEM

- Has 10 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Hence, its base = 10
- The maximum value of a single digit is 9 (one less than the value of the base)
- Each position of a digit represents a specific power of the base (10)
- We use this number system in our day-to-day life
- Example :  $(2586)_{10} = (2 \times 10^3) + (5 \times 10^2) + (8 \times 10^1) + (6 \times 10^0) = 2000 + 500 + 80 + 6$

## OCTAL NUMBER SYSTEM

- Has total 8 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7). Hence, its base = 8
- The maximum value of a single digit is 7 (one less than the value of the base)
- Each position of a digit represents a specific power of the base (8)
- Since there are only 8 digits, 3 bits ( $2^3 = 8$ ) are sufficient to represent any octal number in binary
- Example :  $2057_8$

## HEXADECIMAL NUMBER SYSTEM

- Has total 16 symbols or digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). Hence its base = 16
- The symbols A, B, C, D, E and F represent the decimal values 10, 11, 12, 13, 14 and 15 respectively
- The maximum value of a single digit is 15 (one less than the value of the base)
- Each position of a digit represents a specific power of the base (16)
- Since there are only 16 digits, 4 bits ( $2^4 = 16$ ) are sufficient to represent any hexadecimal number in binary.
- Example :  $(1AF)_{16}$

## CONVERTING A NUMBER OF ANOTHER BASE TO A DECIMAL NUMBER

**Step 1: Determine the column (positional) value of each digit**

**Step 2: Multiply the obtained column values by the digits in the corresponding columns**

**Step 3: Calculate the sum of these products**

Example :  $11101_2 = ( )_{10}$

Step 1	$((1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))$
Step 2	$(16 + 8 + 4 + 0 + 1)$
Step 3	$29_{10}$



## FRACTIONAL NUMBERS IN BINARY NUMBER SYSTEM

						Binary Point				
						↓				
Position	4	3	2	1	0	.	-1	-2	-3	-4
Position Value	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$		$2^{-1}$	$2^{-2}$	$2^{-3}$	$2^{-4}$
Quantity Represented	16	8	4	2	1		$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$

## FRACTIONAL NUMBERS IN OCTAL NUMBER SYSTEM

					Octal Point ↓			
Position	3	2	1	0	.	-1	-2	-3
Position Value	$8^3$	$8^2$	$8^1$	$8^0$		$8^{-1}$	$8^{-2}$	$8^{-3}$
Quantity Represented	512	64	8	1		$1/8$	$1/64$	$1/512$



Q.  $(10110.001)_2 = ( )_{10}$

$$(10110.001)_2 = (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) + (0 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3})$$

$$(10110.001)_2 = (1 \times 16) + (0 \times 8) + (1 \times 4) + (1 \times 2) + (0 \times 1) + (0 \times 1/2) + (0 \times 1/4) + (1 \times 1/8)$$

$$(10110.001)_2 = 16 + 0 + 4 + 2 + 0 + 0 + 0 + 0.125$$

$$(10110.001)_2 = (22.125)_{10}$$

Q.  $(152.25)_8 = ( )_{10}$

$$(152.25)_8 = (1 \times 8^2) + (5 \times 8^1) + (2 \times 8^0) + (2 \times 8^{-1}) + (5 \times 8^{-2})$$

$$(152.25)_8 = 64 + 40 + 2 + (2 \times 1/8) + (5 \times 1/64)$$

$$(152.25)_8 = 64 + 40 + 2 + 0.25 + 0.078125$$

$$(152.25)_8 = 106.328125$$

## CONVERTING A DECIMAL NUMBER TO A NUMBER OF ANOTHER BASE

Step 1: Divide the decimal number to be converted by the value of the new base

Step 2: Record the remainder from Step 1 as the rightmost digit (least significant digit) of the new base number

Step 3: Divide the quotient of the previous divide by the new base

Step 4: Record the remainder from Step 3 as the next digit (to the left) of the new base number

Repeat Steps 3 and 4, recording remainders from right to left, until the quotient becomes zero in Step 3.

- Note that the last remainder thus obtained will be the most significant digit (MSD) of the new base number

Q.  $(25)_{10} = ( )_2$

2	25	
2	12	1 ← First remainder
2	6	0 ← Second Remainder
2	3	0 ← Third Remainder
2	1	1 ← Fourth Remainder
	0	1 ← Fifth Reaminder

Read Up

Binary Number = 11001

Q.  $(569)_{10} = ( )_8$

8	569
8	71
8	8
8	1
	0

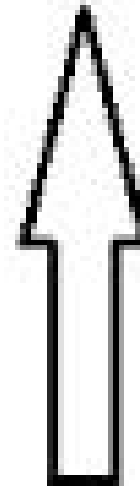
Remainders

1

7

0

1



Read in  
reverse order

Therefore,  $(569)_{10} = (1071)_8$

Q.  $(3479)_{10} = ( )_{16}$

16		3479			
16		217			
16		13	7	→	7
			9	→	9
			13	→	D
					↑
					First Remainder
					Second Remainder
					Third Remainder
					Read up

## CONVERTING A NUMBER FROM ONE BASE TO ANOTHER BASE

Step 1: Convert the original number to a decimal number (base 10)

Step 2: Convert the decimal number so obtained to the new base number

*Example:*  $(545)_6 = ( )_4$

Step 1: Convert from base 6 to base 10

$$\begin{aligned}(545)_6 &= 5 \times 6^2 + 4 \times 6^1 + 5 \times 6^0 \\ &= 5 \times 36 + 4 \times 6 + 5 \times 1 \\ &= 180 + 24 + 5 \\ &= (209)_{10}\end{aligned}$$

Step 2: Convert  $(209)_{10}$  to base 4

$$\text{Hence, } (209)_{10} = (3101)_4$$

$$\text{So, } (545)_6 = (209)_{10} = (3101)_4$$

$$\text{Thus, } (545)_6 = (3101)_4$$

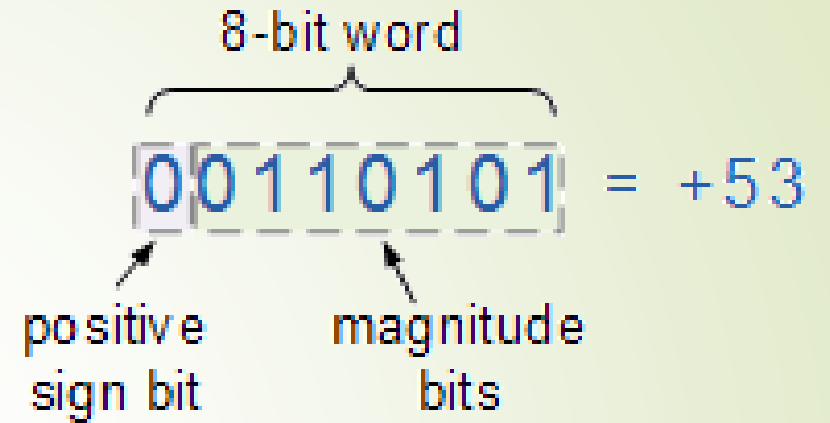
4	209	Remainders
	52	1
	13	0
	3	1
	0	3

## Signed Binary Numbers

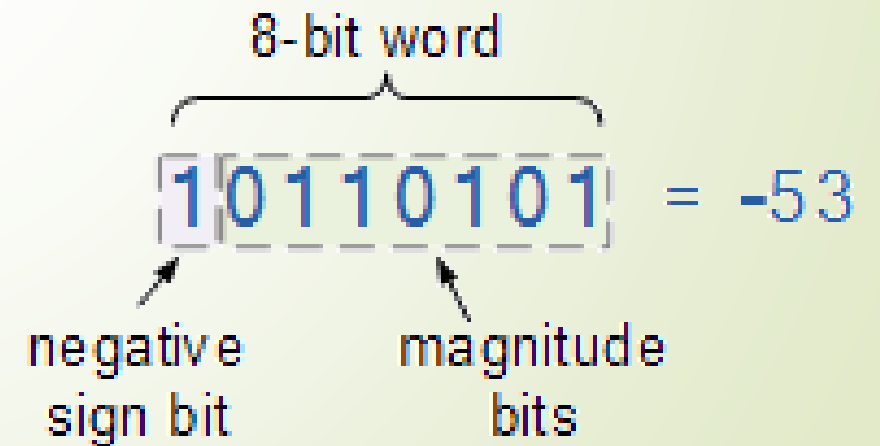
- In digital circuits there is no provision to put a plus or even a minus sign to a number, since digital systems operate with binary numbers that are represented in terms of “0’s” and “1’s”.
- But Digital Systems and computers must also be able to use and to manipulate negative numbers as well as positive numbers.
- For signed binary numbers the most significant bit (MSB) is used as the sign bit. If the sign bit is “0”, this means the number is positive in value. If the sign bit is “1”, then the number is negative in value. The remaining bits in the number are used to represent the magnitude of the binary number in the usual unsigned binary number format way.
- Thus the Sign-and-Magnitude (SM) notation stores positive and negative values by dividing the “n” total bits into two parts: 1 bit for the sign and n–1 bits for the value which is a pure binary number.



## Positive Signed Binary Numbers



## Negative Signed Binary Numbers



# BINARY ARITHMETIC

Arithmetic operations in a computer are done using binary numbers and not decimal numbers and these take place in its arithmetic unit.

## BINARY ADDITION

Case	A	+	B	Sum	Carry
1	0	+	0	0	0
2	0	+	1	1	0
3	1	+	0	1	0
4	1	+	1	0	1

## BINARY SUBTRACTION

Case	A	-	B	Subtract	Borrow
1	0	-	0	0	0
2	1	-	0	1	0
3	1	-	1	0	0
4	0	-	1	0	1

## BINARY MULTIPLICATION

Case	A	x	B	Multiplication
1	0	x	0	0
2	0	x	1	0
3	1	x	0	0
4	1	x	1	1

## BINARY DIVISION

Binary division is similar to decimal division.  
It is called as the long division procedure.

$$101010 / 000110 = 000111$$

$$\begin{array}{r} 111 = 7_{10} \\ 000110 \overline{) 101010} = 42_{10} \\ \underline{-110} = 6_{10} \\ 1001 \\ \underline{-110} \\ 110 \\ \underline{-110} \\ 0 \end{array}$$

**Binary addition for two binary numbers 10001001 and 10010101**

$$\begin{array}{r} 1 \\ 10001001 \\ + 10010101 \\ \hline 10011110 \end{array}$$

**Subtracting 10100010 from 10101010**

$$\begin{array}{r} 10101010 \\ - 10100010 \\ \hline 00001000 \end{array}$$

**Multiply 1001 by 101**

$$\begin{array}{r} 1001 \\ \times 101 \\ \hline 1001 \\ 0000 \\ 1001 \\ \hline 101101 \end{array}$$

**Divide 11010 by 101**

$$\begin{array}{r} 101 \overline{) 11010} \\ \underline{101} \phantom{0} \\ 110 \\ \underline{101} \\ 1 \end{array} \quad \left( \begin{array}{l} 101 \rightarrow \text{Quotient} \\ 1 \rightarrow \text{Remainder} \end{array} \right)$$

## 1's and 2's complement

**1's complement** of a binary number is another binary number obtained by toggling all bits in it, i.e., transforming the 0 bit to 1 and the 1 bit to 0.

1010	→	0101
1111		0000
11110000	→	00001111
10100011		01011100

**2's complement** of a binary number is 1 added to the 1's complement of the binary number.

0 1 1 0 1 1 1 0 ← Original binary value

1 0 0 1 0 0 0 1 ← 1's complement

1 0 0 1 0 0 0 1
+ 1

1 0 0 1 0 0 1 0 ← 2's complement

## SUBTRACTION USING 1'S COMPLEMENT

### SUBTRACTION OF A SMALLER NUMBER FROM A LARGER NUMBER

1. DETERMINE THE 1'S COMPLEMENT OF THE SMALLER NUMBER.
2. ADD THIS TO THE LARGER NUMBER.
3. REMOVE THE CARRY AND ADD IT TO THE RESULT.

Subtract  $(1010)_2$  from  $(1111)_2$

**Direct Subtraction**

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

**1's complement method**

1's complement  $\rightarrow$

$$\begin{array}{r} 1111 \\ + 0101 \\ \hline 10100 \end{array}$$

Carry  $\rightarrow$

$$\begin{array}{r} 10100 \\ + 1 \\ \hline 0101 \end{array}$$



## SUBTRACTION OF A LARGER NUMBER FROM A SMALLER NUMBER

1. DETERMINE THE 1'S COMPLEMENT OF THE LARGER NUMBER.
2. ADD THIS TO THE SMALLER NUMBER.
3. THE ANSWER IS THE 1'S COMPLEMENT OF THE TRUE RESULT AND IS OPPOSITE IN SIGN. THERE IS NO CARRY.

**Example 1.14** Subtract  $(1010)_2$  from  $(1000)_2$  using the 1's complement method. Also subtract by direct method and compare.

### Solution

Direct subtraction

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

1's Complement  $\rightarrow$

1's Complement method

$$1\ 0\ 0\ 0 (+)$$

$$0\ 1\ 0\ 1$$

$$\hline 1\ 1\ 0\ 1$$

No carry is obtained. The answer is the 1's complement of 1101 and is opposite in sign, i.e.  $-0010$ .

## SUBTRACION USING 2'S COMPLEMENT

### SUBTRACTION OF A SMALLER NUMBER FROM A LARGER NUMBER

1. DETERMINE THE 2'S COMPLEMENT OF THE SMALLER NUMBER.
2. ADD THIS TO THE LARGER NUMBER.
3. OMIT THE CARRY.

*Example 1.15* Subtract  $(1010)_2$  from  $(1111)_2$  using the 2's complement method. Subtract by direct method also and compare.

**Solution**

Direct subtraction

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

2's Complement  $\rightarrow$

Carry  $\rightarrow$

2's Complement method

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

The carry is discarded. Thus, the answer is  $(0101)_2$ .



## SUBTRACTION OF LARGER NUMBER FROM A SMALLER NUMBER

1. DETERMINE THE 2'S COMPLEMENT OF THE LARGER NUMBER.
2. ADD THE 2'S COMPLEMENT TO THE SMALLER NUMBER.
3. THERE IS NO CARRY. THE RESULT IS IN THE 2'S COMPLEMENT FORM AND IS NEGATIVE.

Direct subtraction

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

2's Complement method

$$1\ 0\ 0\ 0 (+)$$

$$2's\ Complement \rightarrow \begin{array}{r} 0\ 1\ 1\ 0 \\ \hline \end{array}$$

$$No\ carry \rightarrow \begin{array}{r} 1\ 1\ 1\ 0 \\ \hline \end{array}$$

No carry is obtained. Thus, the difference is negative and the true answer is the 2's complement of  $(1110)_2$ , i.e.  $(0010)_2$ .

## Codes

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

Binary codes can be classified into two types.

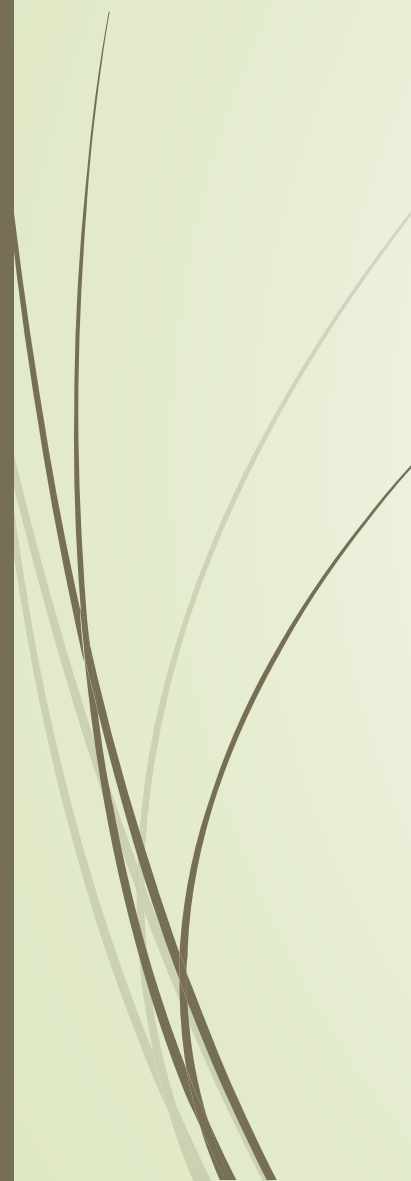

- Weighted codes
- Non weighted codes

If the code has positional weights, then it is said to be weighted code. Otherwise, it is a non weighted code.

## Weighted Binary Codes

### Binary Coded Decimal (BCD)

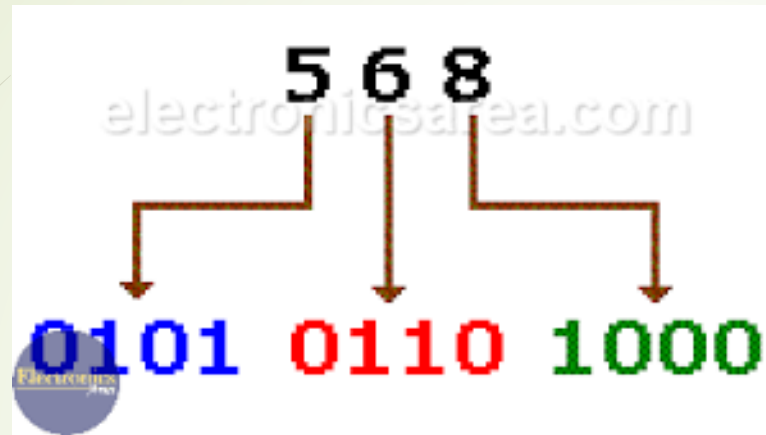
- The BCD is a combination of 4 binary digits that represent decimal numbers.
- The 8421 code is a type of binary coded decimal.
- The weights of this code are 8, 4, 2 and 1.
- This code has all positive weights. So, it is a positively weighted code.
- This code is also called as natural BCD (Binary Coded Decimal) code.



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

BY : Ms. KANIKA DHINGRA





Hence BCD code for  $(568)_{10} = 010101101000$



## Non Weighted Codes

Non weighted codes are codes that are not positionally weighted. This means that each position within a binary number is not assigned a fixed value.

- Excess-3 Code
- Gray Code

### Excess-3 Code

- Excess-3 code represents a decimal number in binary form, as a number greater than 3.
- An excess-3 code is obtained by adding 3 to a decimal number.
- It is a self – complementing code.

Example : Calculate Excess-3 code for  $(643)_{10}$

Step1 : Add 3 to each bit of the decimal number. We get (976)


Step 2 : Converting the above sum into its BCD code

Hence Excess-3 code for  $(643)_{10}$  is 100101110110

## Gray Code

The Gray code belongs to a class of codes called minimum-change codes, in which only one bit in the code group changes when moving from one step to the next.

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



**Conversion of a binary number to gray code** A binary number can be converted to its Gray code when

- (i) the first bit (MSB) of the Gray code is the same as the first bit of the binary number;
- (ii) the second bit of the Gray code equals the exclusive-OR, of the first and second bits of the binary number, i.e. it will be 1 if these binary code bits are different and 0 if they are the same;
- (iii) the third Gray code bit equals the exclusive-OR of the second and third bits of the binary number, and so on.

## Convert $(10110)_2$ to Gray code

**Step 1** The first bit MSB of the Gray code is the same as the first bit of the binary number.

1 0 1 1 0 Binary

↓

1 Gray

**Step 2** Add the first bit of the binary digit to the second bit of the binary. The addition of 1 and 0 is 1. The result is the second bit of the Gray code.

1 ⊕ 0 1 1 0 Binary

↓

1 1 Gray

**Step 3** Add the second bit 0 to the third bit 1 of the binary.

1 0 ⊕ 1 1 0 Binary

↓

1 1 1 Gray

**Step 4** Add the third bit 1 to the fourth bit 1 and omit the carry. The exclusive OR addition of 1 and 1 is 0. This is the fourth bit of the Gray code.




$$\begin{array}{ccccccc}
 1 & 0 & \boxed{1 \oplus 1} & 0 & \text{Binary} \\
 & & \downarrow & & \\
 1 & 1 & 1 & 0 & \text{Gray}
 \end{array}$$

**Step 5** Add the fourth bit 1 to the last bit of the binary.

$$\begin{array}{ccccccc}
 1 & 0 & 1 & \boxed{1 \oplus 0} & \text{Binary} \\
 & & & \downarrow & \\
 1 & 1 & 1 & 0 & 1 & \text{Gray}
 \end{array}$$

As the conversion is complete, the Gray code of the binary 10110 is 11101.



**Conversion from gray code to binary** Conversion of a Gray code into its binary form involves the reverse of the procedure given above:

- (i) The first binary bit (MSB) is the same as that of the first Gray code bit.
- (ii) If the second Gray bit is 0, the second binary bit is the same as that of the first binary; if the second Gray bit is 1, the second binary bit is the inverse of its first binary bit.
- (iii) Step 2 is repeated for each successive bit.

**Step 1** Write the first binary bit 1 which is the MSB of the Gray code.

1 1 0 1 0 1 Gray

↓

1

Binary

**Step 2** The second bit of the Gray code is 1 and therefore the second bit of the binary is 0, i.e. inverse of the first binary bit '1'.

1      1 0 1 0 1 Gray

↓

1 → 0

Binary

**Step 3** The third bit of the Gray code is 0 and therefore the third bit of the binary is same as that of the second binary bit, i.e. 0.

1 1      0 1 0 1 Gray

↓

1 0 → 0

Binary



#### Step 4

1 1 0      1 0 1    Gray  
                 ↓  
1 0 0 → 1      Binary

#### Step 5

1 1 0 1 0 1    Gray  
                 ↓  
1 0 0 1 → 1      Binary

#### Step 6

1 1      0 1 0 1    Gray  
                 ↓  
1 0      0 1    1 → 0    Binary

Therefore,  $[110101]_G = [100110]_2$ .

## ERROR DETECTING AND ERROR CORRECTING CODES

- Error detection and correction code plays an important role in the transmission of data from one source to another.
- The noise also gets added into the data when it transmits from one system to another, which causes errors in the received binary data at other systems. The bits of the data may change(either 0 to 1 or 1 to 0) during transmission.
- It is impossible to avoid the interference of noise, but it is possible to get back the original data. For this purpose, we first need to detect either an error  $z$  is present or not using error detection codes. If the error is present in the code, then we will correct it with the help of error correction codes.

## **ERROR DETECTING CODES**

- **PARITY CHECK**
- **CHECK SUMS**

### **PARITY CHECK**

- If a single error transforms a valid code word into an invalid one, it is said to be a single error-detecting code.
- The Parity check is the most simple and commonly used error detecting method in which an extra parity bit is included with the binary message, to make the total number of 1's either odd or even, resulting in two methods :
  - Even parity method
  - Odd parity method



- In the even - parity method, the total number of 1's in the code group ( including the parity bit ) must be an even number.
- In odd – parity method, the total number of 1's ( including the parity bit ) must be an odd number.

Message xyz	Even-parity code xyz p	Odd-parity code xyz p
000	000 0	000 1
001	001 1	001 0
010	010 1	010 0
011	011 0	011 1
100	100 1	100 0
101	101 0	101 1
110	110 0	110 1
111	111 1	111 0

## CHECK SUMS

- In this method a checksum is created based on the data values in the data blocks to be transmitted using some algorithm and appended to the data.
- When the receiver gets this data, a new checksum is calculated and compared with the existing checksum. A non-match indicates an error.
- For error detection by checksums, data is divided into fixed sized frames or segments.
  - **Sender's End** – The sender adds the segments using 1's complement arithmetic to get the sum. It then complements the sum to get the checksum and sends it along with the data frames.
  - **Receiver's End** – The receiver adds the incoming segments along with the checksum using 1's complement arithmetic to get the sum and then complements it.
- If the result is zero, the received frames are accepted; otherwise they are discarded.

### Sender's End

Frame 1:	11001100
Frame 2:	+ 10101010
Partial Sum:	1 01110110
	+ 1
	01110111
Frame 3:	+ 11110000
Partial Sum:	1 01100111
	+ 1
	01101000
Frame 4:	+ 11000011
Partial Sum:	1 00101011
	+ 1
Sum:	00101100
Checksum:	11010011

### Receiver's End

Frame 1:	11001100
Frame 2:	+ 10101010
Partial Sum:	1 01110110
	+ 1
	01110111
Frame 3:	+ 11110000
Partial Sum:	1 01100111
	+ 1
	01101000
Frame 4:	+ 11000011
Partial Sum:	1 00101011
	+ 1
Sum:	00101100
Checksum:	11010011
Sum:	11111111
Complement:	00000000

Hence accept frames.



# ERROR CORRECTING CODES

- Error-correcting codes (ECC) are a sequence of numbers generated by specific algorithms for detecting and removing errors in data that has been transmitted over noisy channels.
- Error correcting codes ascertain the exact number of bits that has been corrupted and the location of the corrupted bits, within the limitations in algorithm.

## Hamming Code

- Hamming code is a block code that is capable of detecting up to two simultaneous bit errors and correcting single-bit errors. It was developed by R.W. Hamming for error correction.
- In this coding method, the source encodes the message by inserting redundant bits within the message. These redundant bits are extra bits that are generated and inserted at specific positions in the message itself to enable error detection and correction.
- When the destination receives this message, it performs recalculations to detect errors and find the bit position that has error.



## Encoding a message by Hamming Code

Following procedure is used by the sender to encode the message :

1. Calculation of the number of redundant bits.

$$2^r \geq m + r + 1$$

where  $m$  is the number of data bits and  $r$  is the number of redundant bits.

2. Positioning the redundant bits.

The  $r$  redundant bits placed at bit positions of powers of 2, i.e. 1, 2, 4, 8, 16 etc. They are referred to as  $r_1$  (at position 1),  $r_2$  (at position 2),  $r_3$  (at position 4),  $r_4$  (at position 8) and so on.

### 3. Calculating the values of each redundant bit.

Each redundant bit,  $r_i$ , is calculated as the parity, generally even parity, based upon its bit position. It covers all bit positions whose binary representation includes a 1 in the  $i$ th position except the position of  $r_i$ . Thus –

- $r_1$  is the parity bit for all data bits in positions whose binary representation includes a 1 in the least significant position excluding 1 (3, 5, 7, 9, 11 and so on)
- $r_2$  is the parity bit for all data bits in positions whose binary representation includes a 1 in the position 2 from right except 2 (3, 6, 7, 10, 11 and so on)
- $r_3$  is the parity bit for all data bits in positions whose binary representation includes a 1 in the position 3 from right except 4 (5-7, 12-15, 20-23 and so on)
- $r_4$  is the parity bit for all data bits in positions whose binary representation includes a 1 in the position 4 from right except 8 (9, 10, 11 and so on)

## Decoding a Message in Hamming Code

Following procedure is used by receiver to detect errors and correct them:

1. Calculation of the number of redundant bits.

$$2^r \geq m + r + 1$$

where  $m$  is the number of data bits and  $r$  is the number of redundant bits.

2. Positioning the redundant bits.

The  $r$  redundant bits placed at bit positions of powers of 2, i.e. 1, 2, 4, 8, 16 etc.

### 3. Parity Checking

Parity bits are calculated based upon the data bits and the redundant bits using the same rule as during generation of  $c_1, c_2, c_3, c_4$  etc. Thus

$c_1 = \text{parity}(1, 3, 5, 7, 9, 11 \text{ and so on})$

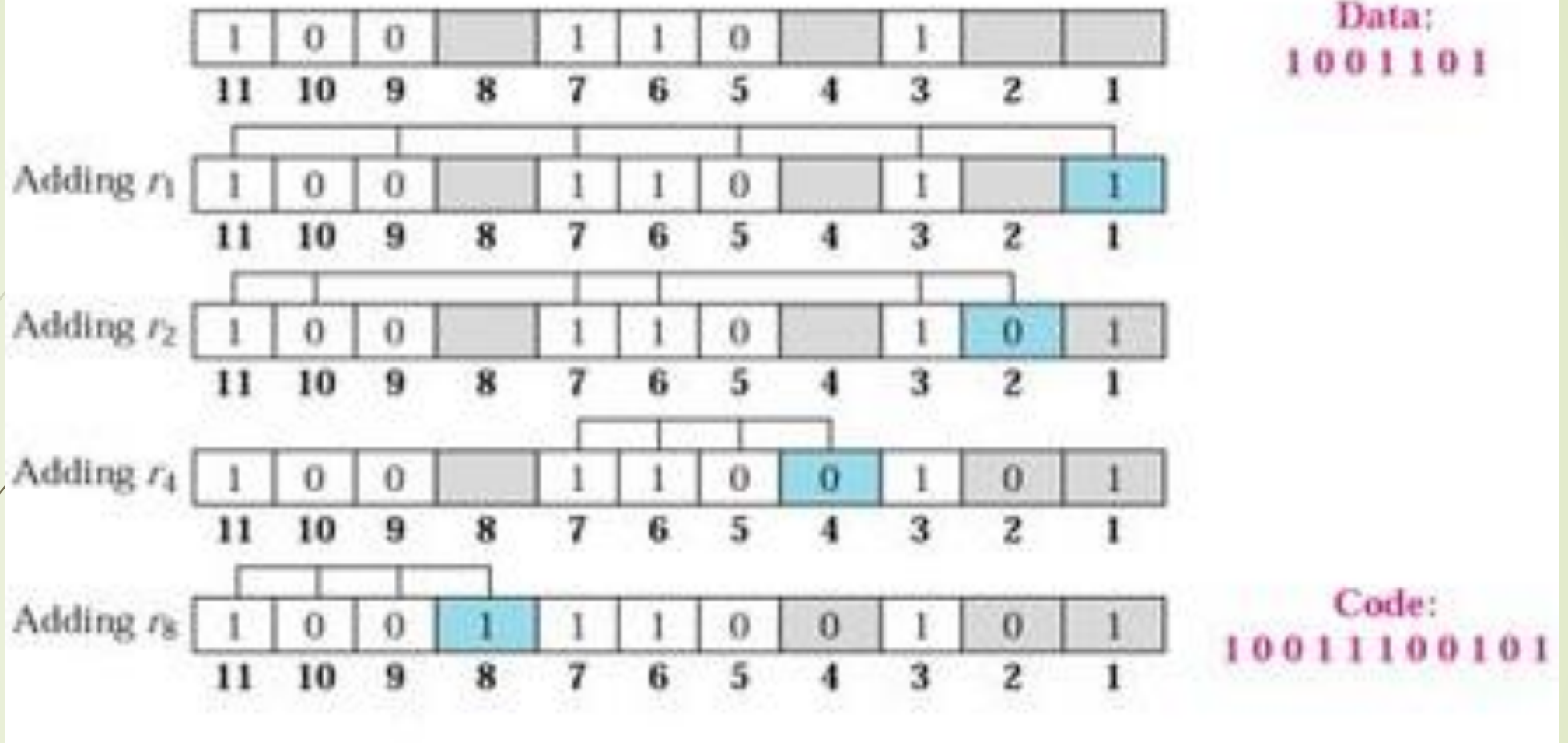
$c_2 = \text{parity}(2, 3, 6, 7, 10, 11 \text{ and so on})$

$c_3 = \text{parity}(4-7, 12-15, 20-23 \text{ and so on})$

$C_4 = \text{parity}(8, 9, 10, 11 \text{ and so on})$

### 4. Error Detection and Correction

The decimal equivalent of the parity bits binary values is calculated. If it is 0, there is no error. Otherwise, the decimal value gives the bit position which has error. For example, if  $c_4c_3c_2c_1 = 1001$ , it implies that the data bit at position 9, decimal equivalent of 1001, has error. The bit is flipped to get the correct message.



Here  $m = 7$ , therefore according to formula  $2^r \geq m + r + 1$  the value  $r = 4$ .

## Case 1: If message received is correct

Message Sent : 10011100101

Message Received : 10011100101

Here since  $m = 7$  and  $r = 4$ , therefore recalculating the parity bits we get

$c_1 = 0$      $c_2 = 0$      $c_3 = 0$  and  $c_4 = 0$

Hence the decimal equivalent of parity bits binary value ( $c_4 c_3 c_2 c_1$ ) is 0 indicating that there is no error.



## Case 2 : If message received is corrupted

Message Sent : 10011100101

Message Received : 10011000101

Here since  $m = 7$  and  $r = 4$ , therefore recalculating the parity bits we get

$c_1 = 0$      $c_2 = 1$      $c_3 = 1$  and  $c_4 = 0$

Hence the decimal equivalent of parity bits binary value ( $c_4 c_3 c_2 c_1$ ) is 6 indicating that there is an error at position 6. The bit at position 6 is flipped to get the correct answer.

Thus correct message is 10011100101.