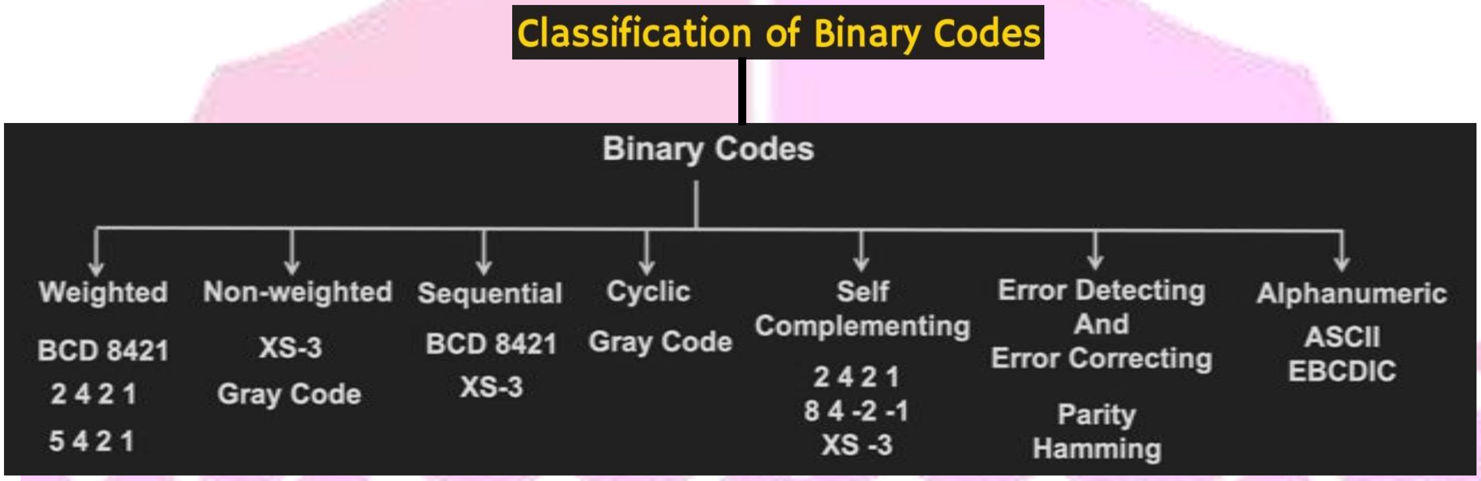
Binary Codes

- 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.



Weighted code

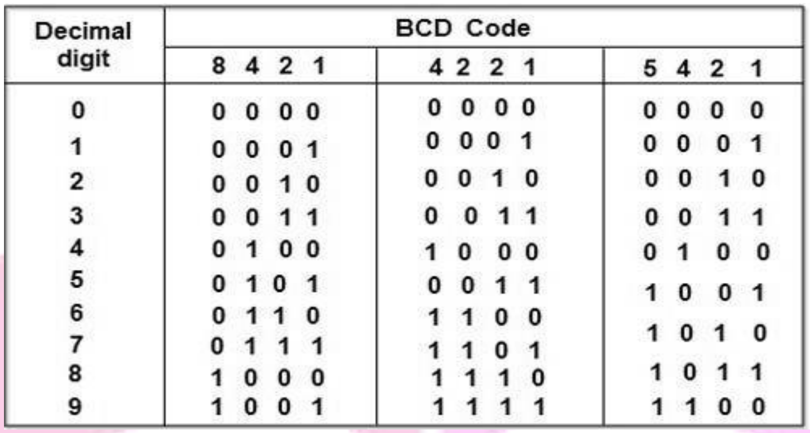
- Each binary bit is assigned by a "weight" and values depend on the position of the binary bit.
- The sum of the weights of these binary bits, whose value is 1 is equal to the decimal digit which they represent.
- In other words, if w1, w2, w3 and w4 are the weights of the binary digits, and x1, x2, x3 and x4 are the corresponding bit values, then the decimal digit N=w4x4 + w3x3+w2x2+w1x1 is represented by the binary sequence x4x3x2x1.
- **Example of Weighted codes:** BCD, 8421, 2421, 7421, 5421, 4221, 5211, 3321 etc.

- **Weighted codes are used in:**
  a) Data manipulation during arithmetic operation.
  b) For input/output operations in digital circuits.
  c) To represent the decimal digits in calculators, volt meters etc.

Binary coded Decimal Code (8421)

- Binary-coded decimal is a code to represent a given decimal number in an equivalent binary form.
- Generally BCD assigns a four-digit binary code to each digit 0 to 9 in a decimal (base 10) number.
- BCD number which is representing the decimal number greater than 9 is called packed BCD. Ex:- $(12)_{10}$=(10010), $(148)_{10}$=(101001000) etc
- There are several BCD codes like 8421, 2421, 3321, 4221, 5211, 5311, 5421, etc.
- The most common BCD code is the 8421 BCD code.
- We can convert a binary number to BCD by using "Shift ADD-3" method.

- BCD code is a Sequential code. In sequential codes, each succeeding code is one binary number greater than its preceding code,



Decimal digit	BCD Code		
	8 4 2 1	4 2 2 1	5 4 2 1
0	0 0 0 0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 0	0 0 1 0
3	0 0 1 1	0 0 1 1	0 0 1 1
4	0 1 0 0	1 0 0 0	0 1 0 0
5	0 1 0 1	0 0 1 1	1 0 0 1
6	0 1 1 0	1 1 0 0	1 0 1 0
7	0 1 1 1	1 1 0 1	1 0 1 1
8	1 0 0 0	1 1 1 0	1 0 1 1
9	1 0 0 1	1 1 1 1	1 1 0 0

Conversion of decimal number to BCD

- Convert each digit to 4 bit binary code

Example: **$(49)_{10}$ = $(1001001)_{BCD}$**

Conversion of BCD to Decimal

- Pair 4 bits of given binary and convert in equivalent Decimal.

Example: **$(10011101011001)_{BCD}$ = $(2759)_{10}$**

BCD Addition

Perform BCD addition by following the binary addition and check result with some following cases:-

Case1:- If the resultant sum is less than or equal to 9 and final carry is 0 then the result is correct.

Example: 0100 + 0101 = 1001 (<=9 and no carry occurs) correct

Case2:- If the resultant sum is less than 9 and final carry is 1 then the result is incorrect. To find the correct result we need to add 6 (0110) in the resultant sum.

Example: 1001 + 1001 = 10010 (<=9 and carry occurs) incorrect

Therefore add 0110 (6) in the result

10010+ 0110 =11000 (Correct)

Case3:- If the resultant sum is greater than 9 and final carry is 0 then the result is incorrect. To find the correct result we need to add 6 (0110) in the resultant sum.

Example: 1001 + 0101 = 1110 (>=9 and no carry occurs) incorrect

Therefore add 0110 (6) in the result

1110+ 0110 =10100 (Correct)

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 more complicated than binary number.
- BCD needs more number of bits than binary to represent the decimal number. So BCD is less efficient than binary.

Non-Weighted Codes

- In non-weighted or un-weighted codes, the digit value does not depend upon their position i.e., each digit position within the number is not assigned fixed value.
- Examples of non-weighted codes: Excess-3 code and gray code.
- Non weighted codes are used in:
  a) To perform certain arithmetic operations.
  b) Shift position encodes.
  c) Used for error detecting purpose.

Excess-3 code

- The excess-3 code of a decimal number is achieved by adding the number 3 to the 8421 code.
- In other words we can say excess-3 code is a 4 bit code which can be achieved by adding 3 in 8421 code.
- Excess-3 code is non weighted code.
- It is also a reflective or self complementing and sequential code.
- It is represented as xs-3 or x-3 code.

DECIMAL NUMBER	BINARY NUMBER	8421 CODE	EXCESS-3
0	0	0000	0011
1	1	0001	0100
2	10	0010	0101
3	11	0011	0110
4	100	0100	0111
5	101	0101	1000
6	110	0110	1001
7	111	0111	1010
8	1000	1000	1011
9	1001	1001	1100

GRAY Code

- This code doesn't have any weights. So, it is an un-weighted code.
- In the gray code the successive values are differed in one bit position only. Hence, this code is called as unit distance code.
- It is also called as reflected binary code or minimum error code or cyclic code.
- Binary code is converted to gray code to reduce switching operation.

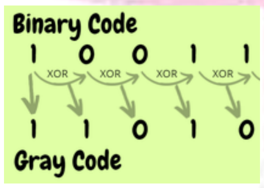
Decimal Number	Binary Code	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Binary to Gray code conversion

Step1:- Write MSB bit as given.
Step2:- add the MSB to the next bit, write the sum and ignore the carry.
Step3:- Repeat step2 till last bit.

Or use Xor for conversion

Example:

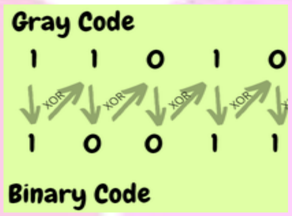


Gray to Binary conversion

Step1:- Write MSB bit as given.
Step2:- add the MSB to the next bit, write the sum and neglect the carry.
Step3:- Repeat step2 till last bit.

Or use Xor for conversion

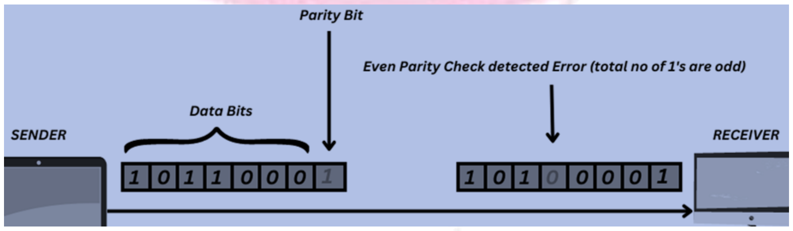
Example:



Parity

- Parity is used to check error in data transmission. Parity bit is an extra bit added to the message to detect error during data transmission.
- During data transmission the external noise can change bits from 1 to 0 or 0 to 1 which changes the meaning of the actual message and is called error.
- For efficient data transfer, there should be an error detection and correction codes.
- There are three error detection techniques Parity Bit, CheckSum and Cyclic Redundancy Check (CRC).
- A Parity Generator is a combinational logic circuit that generates the parity bit in the transmitter.
- On the other hand, a circuit that checks the parity in the receiver is called Parity Checker.
- A combined circuit or device of parity generators and parity checkers are commonly used in digital systems to detect the single bit errors in the transmitted data.
- There are two parity bit checker:-
  - ✓ Even parity checks
  - ✓ Odd parity checks

Even Parity:- In even parity, the parity bit is set (1) or reset (0) to ensure that the total number of 1's in the code, including the parity bit, is an even number. If any single bit error occurs during transmission, the parity check will detect it. However, it cannot detect errors where an even number of bits are flipped (e.g., two bits).

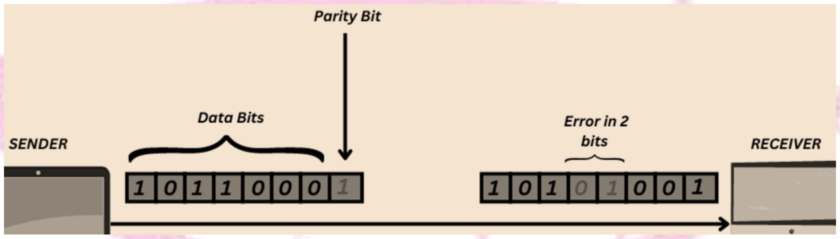


Odd Parity:- In odd parity, the parity bit is set (1) or cleared (0) to ensure that the total number of 1s in the code, including the parity bit, is an odd number. Similar to even parity, it can detect single bit errors but not errors where an odd number of bits are flipped.

4-bit received message				Odd	Even
A	B	C	D		
0	0	0	0	0	1
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	1

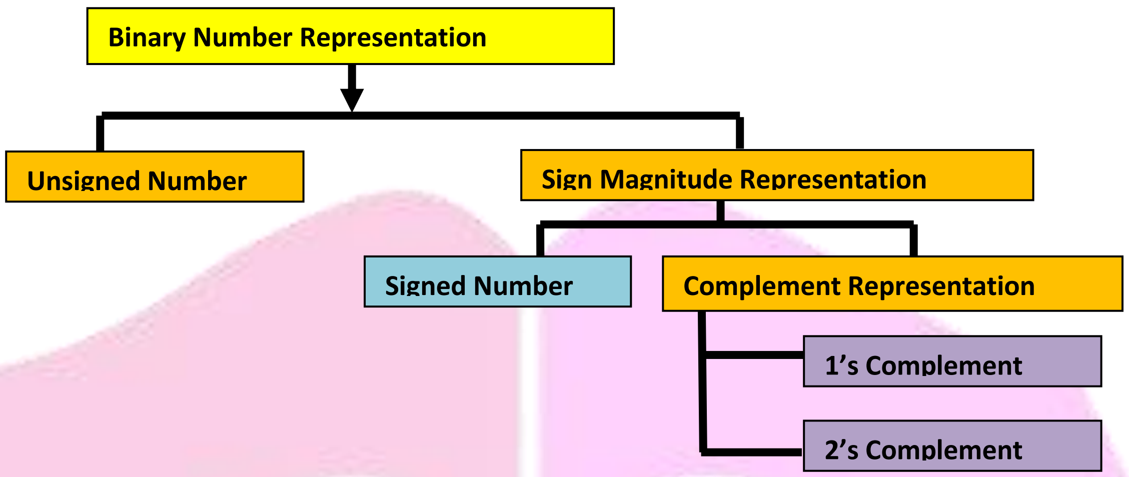
Disadvantages of parity bit check

1. It detect error in a single bit only and it also cannot determine the exact location of error in the data bit.
2. If the number of bits in even or odd parity check increase or decrease (data changed) but remained to be even or odd respectively then it won't be able to detect error as the number of bits are still even or odd.



Hamming code- error detection and correction

- Richard W. hamming invented Hamming codes to detect and correct the single bit error in data. later on it is extended up to 2-error detecting codes.
- Hamming codes are created because parity check cannot correct error in the data. The Hamming codes are inserted to any block length of data between actual data and redundancy bits.
- In the hamming code we add some redundant bits with the original message (data bits) to detect and correct the error.
- First we encode the data by finding and adding the value of redundant bits.
- After that we decode the data to find the single bit error and if error occurs we can correct it.


Binary Number Representation
Unsigned Number
Sign Magnitude Representation
Signed Number
Complement Representation
1's Complement
2's Complement

Unsigned Binary Numbers

- Only positive binary numbers can be represented.
- For n-bit unsigned binary numbers, all n-bits are used to represent the magnitude of the number. Need not to take extra bit.
- In unsigned binary number representation, using n-bits, we can represent the numbers from 0 to $2^n - 1$.

For example:
Represent $(18)_{10}$ in 6-bit unsigned number form.
$(18)_{10} = (010010)_2$

Sign Magnitude Representation

- In sign-magnitude representation, the Most Significant bit of the number is a sign bit and the remaining bit represents the magnitude of the number in binary form.
- For positive numbers, the sign bit is 0 and for negative number, the sign bit is 1.
- Using signed binary number representation both positive and negative numbers can be represented.

For example: (+10 and -10) 00001010 and 10001010
Representation of 8-bit sign-magnitude number, the MSB is a sign bit and the remaining 7 bits represent the magnitude.

There are three different ways the signed binary numbers can be represented.

a) Signed Magnitude Representation
b) 1's Complement Representation
c) 2's Complement Representation

Signed Number

Using n-bits, the range of numbers that can be represented in Sign Magnitude Representation is from –$(2^{n-1} - 1)$ to $(2^{n-1} – 1)$.

1’s Complement Representation

- We represent negative binary number using 1’s complement form.
- 1’s complement of a binary number is obtained by inverting the binary bits from 0 to 1 and 1 to 0.

For example: $(-24)_{10}$ to represent a negative number, first write the binary representation of its positive parts.
$(11000)_2 = (00111)_2$ (1’s complement representation of -24)

Subtraction Using 1’s complement

Step 1: Represent the Numbers in Binary
Step 2: Find the 1's Complement of the Subtrahend
Step 3: Add the Minuend to the 1's Complement of the Subtrahend.
Step 4: If the result has a carryover, then add that carry over in the least significant bit
Step 5: If there is no carryover, then take the 1’s complement of the resultant, and it is negative.

2’s Complement Representation

- In 2’s complement representation also, the representation of the positive number is same as 1’s complement and sign-magnitude form.
- But the representation of the negative number is different.

For example: Representation of -25 in 2’s complement form
Rules:

- Write the number corresponding to +25.
- Find 1’s complement of the +25 then add 1 to get 2’s complement.

Or

- After the first ‘1’ is encountered, invert all the 1s in the number with 0s and 0s in the number with 1s (including the sign bit)
- The resultant number is 2’s complement representation of the number -25.

Subtraction Using 2's complement

Step 1: Represent the Numbers in Binary
Step 2: Find the 2's Complement of the Subtrahend
Step 3: Add the Minuend to the 2's Complement of the Subtrahend.
Step4: If the final carry over of the sum is 1, then it is dropped and the result is positive.
Step5: If there is no carry over, then 2's complement of the sum is the final result and it is negative.

Difference between 1's complement and 2's complement

S.No.	1's complement	2's complement
1.	To get 1's complement of a binary number, simply invert the given number.	To get 2's complement of a binary number, simply invert the given number and add 1 to the least significant bit (LSB) of given result.
2.	1's complement of binary number 110010 is 001101	2's complement of binary number 110010 is 001110
3.	Simple implementation which uses only NOT gates for each input bit.	Uses NOT gate along with full adder for each input bit.
4.	Can be used for signed binary number representation but not suitable as ambiguous representation for number 0.	Can be used for signed binary number representation and most suitable as unambiguous representation for all numbers.
5.	0 has two different representation one is -0 (e.g., 1 1111 in five bit register) and second is +0 (e.g., 0 0000 in five bit register).	0 has only one representation for -0 and +0 (e.g., 0 0000 in five bit register). Zero (0) is considered as always positive (sign bit is 0)
6.	For k bits register, positive largest number that can be stored is $(2^{(k-1)} -1)$ and negative lowest number that can be stored is $-(2^{(k-1)} - 1)$.	For k bits register, positive largest number that can be stored is $(2^{(k-1)} -1)$ and negative lowest number that can be stored is $-(2^{(k-1)})$.
7.	end-around-carry-bit addition occurs in 1's complement arithmetic operations. It added to the LSB of result.	end-around-carry-bit addition does not occur in 2's complement arithmetic operations. It is ignored.
8.	1's complement arithmetic operations are not easier than 2's complement because of addition of end-around-carry-bit.	2's complement arithmetic operations are much easier than 1's complement because of there is no addition of end-around-carrybit.
9.	Sign extension is used for converting a signed integer from one size to another.	Sign extension is used for converting a signed integer from one size to another.