Chapter 1
Data Representation
(4 Hrs)

Prepared By: Rolisha Sthapit

CONTENTS

1.1 Data Representation: Binary Representation, BCD, Alphanumeric Representation, Complements ((r-1)’s Complement and r’s complement), Fixed Point representation, Representing Negative Numbers, Floating Point Representation, Arithmetic with Complements (Subtraction of Unsigned Numbers, Addition and Subtraction of Signed Numbers), Overflow, Detecting Overflow

1.2 Other Binary Codes: Gray Code, self Complementing Code, Weighted Code (2421 and 8421 codes), Excess-3 Code, EBCDIC

1.3 Error Detection Codes: Parity Bit, Odd Parity, Even parity, Parity Generator & Checker

Computer Organization vs Architecture

- **Computer Architecture** refers to those attributes of a system that have a direct impact on the logical execution of a program. Examples:

- the instruction set
- the number of bits used to represent various data types
- I/O mechanisms
- memory addressing techniques

- **Computer Organization** refers to the operational units and their interconnections that realize the architectural specifications. Examples are things that are transparent to the programmer:

- control signals
- interfaces between computer and peripherals
- the memory technology being used.

- So, for example, the fact that a multiply instruction is available is a computer architecture issue. How that multiply is implemented is a computer organization issue.
- **Architecture** is those attributes visible to the programmer **-** Instruction set, number of bits used for data representation, I/O mechanisms, addressing techniques. e.g. Is there a multiply instruction?
- **Organization** is how features are implemented **-** Control signals, interfaces, memory technology. e.g. Is there a hardware multiply unit or is it done by repeated addition?
- **Computer architecture** is concerned with the structure and behavior of computer system as seen by the user.
- **Computer organization** is concerned with the way the hardware components operate and the way they are connected together to form a computer system.

1.1 Data Representation: Data Types:

- The term “data” refers to factual information used for analysis or reasoning.
- The data types found in the registers of digital computers may be classified as being one of the following categories:

(1) Numbers used in arithmetic computation,

(2) Letters of the alphabet used in data processing,

(3) Other discrete symbols used for specific purposes.

- Data are numbers and other binary-coded information that are operated on to achieve required computational results.
- The Binary Number System is the most natural system used in digital computer but sometimes it is convenient to employ different number system especially decimal number system since it is used by people for performing arithmetic operations.

Number System

- A number system of base or radix is a system that uses distinct symbols for **‘r’** digit symbols to determine the quantity that the number represents. It is necessary to multiply each digit by an integer power of **‘r’** and the sum all the weighted digits.
- We can categorize number system as below:

- Binary number system (r = 2)
- Octal Number System (r = 8)
- Decimal Number System (r = 10)
- Hexadecimal Number system (r = 16)

Number System

i) Decimal

→ Base, or radix (r) = 10

→ 10 symbols: 0,1,2,3,4,5,6,7,8,9

→ E.g. $630.5 = 6*10^2 + 3*10^1 + 0*10^0 + 5*10^{-1}$

ii) Binary

→ r = 2

→ 2 symbols: 0,1

→ E.g. $110010.01 = 1*2^5 + 1*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 0*2^0 + 0*2^{-1} + 1*2^{-2}$

iii) Octal

→ r = 8

→ 8 symbols: 0,1,2,3,4,5,6,7

→ E.g. $(736.4)_8 = 7*8^2 + 3*8^1 + 6*8^0 + 4*8^{-1}$

iv) Hexadecimal

→ r = 16

→ 16 symbols: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

→ E.g. $A9D = A*16^2 + 9*16^1 + D*16^0$

Exercise

1. Convert binary 101101 to decimal
2. Convert Hexadecimal number F3 to decimal
3. Convert decimal 41 to Binary
4. Convert decimal 153 to octal.
5. Convert decimal 235 to hexadecimal
6. Convert decimal (0.6875) to binary

(Answer 1: 45 2: 243 3: 101001 4:231 5: EB 6: 0.1011)

Complements

- Complements are used in digital systems for simplifying the subtraction operation and for logic manipulation, simplifying operations leads to simpler, less expensive to implement.
- Two types of complements are:

i) r’s complement (Radix Complement)

ii) (r-1)’s complement (The Diminished Radix Complement)

i. r-1's Complement:

→ (r-1)'s complement of N (positive number with base r) is defined as

$(r^n - r^{-m}) - N$, for number with fraction

$(r^n - 1) - N$, for number with integer only

where,

n →number of digits in integer part of N

m→number of digits in fraction part of N

r→base or radix

For 1's Complement of 1101100
Solution: Make 0 to 1 and 1 to 0
0010011

For 9's Complement of 246700
Solution: Subtract from 999999
(999999-246700) = 753299
(The number of 9 should be equal to the number given)

ii. r's Complement:

→ r's complement of N (positive number with base r) is defined as

$r^n - N$, for $N \neq 0$

and 0, for $N = 0$

where,

n=number of digits (integer parts only)

For2's Complement of 1101100

Solution: First find 1's complement

0010011

Add 1

= 0010100

For 10's Complement of 246700

Solution: Subtract from 999999

(999999-246700) = 753299

Add 1

= 753300

- To find (r-1)’s complement, subtract all digit from (r-1).
- E.g. for 1’s complement, subtract each digit from 1.
- for 7’s complement, subtract each digit from 7.
- for 9’s complement, subtract each digit from 9.
- for 15’s complement, subtract each digit from 15.

And,

- To find r’s complement, subtract each digit from (r-1), and then add 1 to the last digit.
- E.g. for 2’s complement, subtract each digit from 1, and then add 1 to the last digit.
- for 8’s complement, subtract each digit from 7, and then add 1 to the last digit.
- for 10’s complement, subtract each digit from 9, and then add 1 to the last digit.

Subtraction with r’s complement

- Subtraction of two positive numbers (M - N), both of base r, is done as follows:

i) Add M to r’s complement of N.

ii) If an end carry occurs, discard it. What is left is result.

iii) If an end carry does not occur, take r’s complement of the number obtained in step (i) and place a negative sign in front. That is the result.

- **Subtraction with (r-1)’s complement**
- Subtraction of two positive numbers (M - N), both of base r, is done as follows:

i) Add M to (r-1)’s complement of N.

ii) If an end carry occurs, add 1 to LSB. What is received is result.

iii) If an end carry does not occur, take (r-1)’s complement of the number obtained in step (i) and place a negative sign in front. That is the result.

Fixed Point Representation

- There may be signed and unsigned numbers. In ordinary arithmetic, a negative number is represented by a ‘-’ sign and positive number by a ‘+’ sign.
- Because of hardware limitations, computers must represent everything in 1’s and 0’s, including the sign of a number.
- As a consequence it is necessary to represent the sign with a bit placed in the left most position of the number. The convention is to make the sign bit equal to 0 for positive and 1 for negative.
- In addition to the sign, a number may have a binary point. The position of the binary point is needed to represent fractions, integers or mixed integer fraction numbers.
- There are two ways of specifying the position of the binary point: by giving it a fixed point or by employing a floating point representation.

- The fixed point method assumes that the binary point is always fixed in one position. Fixed point position usually uses one of the two following positions:

❑ A binary point in the extreme left of the register to make the stored number a fraction.

❑ A binary point in the extreme right of a register to make the stored number an integer.

Integer Representation

- When an integer is positive, the MSB or signed bit is 0 and the remaining bits represents the magnitude.
- When an integer is negative, the MSB or signed bit is1 but the rest of the number can be represented in one of the three ways:

- Signed Magnitude Representation
- Signed 1’s Complement Representation
- Signed 2’s Complement Representation

Suppose, we have 8-bit register.

- For +ve number,

i) Signed-magnitude representation

When an integer binary number is positive, the sign is represented by 0 and the magnitude by a positive binary number.

E.g. +14 = 0 0001110

- For –ve number,

i) Signed-magnitude representation

1 for the -ve sign followed by binary equivalent of that number. E.g. -14 = 1 0001110

ii) Signed-1’s complement representation

1 for the -ve sign followed by 1’s complement of that number. E.g. -14 = 1 1110001

iii) Signed-2’s complement representation

1 for the -ve sign followed by 2’s complement of that number. E.g. -14 = 1 1110010

Arithmetic addition and subtraction of signed numbers

Addition

- Mostly signed 2's complement system is used. So, in this system only addition and complement is used.
- Procedure: add two numbers including sign bit and discard any carry out of the sign bit position. (note: negative numbers initially be in the 2's complement and that if the sum obtained after the addition is negative, it is in 2's complement form)

+6	00000110	−6	11111010
+13	00001101	+13	00001101
+19	00010011	+7	00000111
+6	00000110	−6	11111010
−13	11110011	−13	11110011
−7	11111001	−19	11101101

In each of the 4 cases, the operation performed is always addition, including the sign-bits. Any carry out of the sign bit is discarded and negative results are automatically in 2's complement form.

Subtraction:

- Subtraction of two signed binary numbers is done as: take the 2's complement of the subtrahend (including the sign bit) and add it to the minuend (including the sign-bit). The carry out of the sign bit position is discarded.
- Procedure: subtraction operation can be changed to the addition operation if the sign of the subtrahend is changed:

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

Example: (-6)-(-13) = +7, in binary with 8-bits this is written as:

-6 → 11111010
-13 → 11110011 (2's complement form)

Subtraction is changed to addition by taking 2's complement of the subtrahend (-13) to give (+13).

-6 → 11111010
+13 → 00001101

+7 → 100000111 (discarding end carry).

Overflow

- When two numbers of n digits each are added and the sum occupies n + 1 digits, we say that an overflow occurred.
- An overflow is a problem in digital computers because the width of registers is finite. A result that contains n + 1 bits cannot be accommodated in a register with a standard length of n bits. For this reason, many computers detect the occurrence of an overflow, and when it occurs, a corresponding flip-flop is set which can then be checked by the user.
- The detection of an overflow after the addition of two binary numbers depends on whether the numbers are considered to be signed or unsigned.
- When two unsigned numbers are added, an overflow is detected from the end carry out of the most significant position.
- In the case of signed numbers, the leftmost bit always represents the sign, and negative numbers are in 2's complement form. When two signed numbers are added, the sign bit is treated as part of the number and the end carry does not indicate an overflow.

- **An overflow cannot occur after an addition if one number is positive and the other is negative, since adding a positive number to a negative number produces a result that is smaller than the larger of the two original numbers.**
- An overflow may occur if the two numbers added are both either positive or both negative.

For example: two signed binary numbers +70 and +80 are stored in two 8-bit registers.

carries:	0	1	carries:	1	0
+70	0	1000110	−70	1	0111010
+80	0	1010000	−80	1	0110000
+150	1	0010110	−150	0	1101010

Since the sum of numbers 150 exceeds the capacity of the resister (since 8-bit register can store values ranging from +127 to -127), hence the overflow.

Overflow Detection

- An overflow condition can be detected by observing two carries: carry into the sign bit position and carry out of the sign bit position.
- Consider example of above 8-bit register, if we take the carry out of the sign bit position as a sign bit of the result, 9-bit answer so obtained will be correct.
- Since answer can not be accommodated within 8-bits, we say that an overflow occurred.
- If these two carries are equal ==> no overflow
- If these two carries are not same ==> overflow condition is produced.
- If two carries are applied to an exclusive-OR gate, an overflow will be detected when output of the gate is equal to 1.

Decimal Representation

- We can normally represent decimal numbers in one of following two ways

- By converting into binary

- By using BCD codes

- **By converting into binary**

Advantage

- Arithmetic and logical calculation becomes easy. Negative numbers can be represented easily.

Disadvantage

- At the time of input conversion from decimal to binary is needed and at the time of output conversion from binary to decimal is needed.
- Therefore this approach is useful in the systems where there is much calculation than input/output.

- **By using BCD codes**

Decimal number	Binary-coded decimal (BCD) number	
0	0000	Code for one decimal digit
1	0001	
2	0010	
3	0011	
4	0100	
5	0101	
6	0110	
7	0111	
8	1000	
9	1001	
10	0001 0000	
20	0010 0000	
50	0101 0000	
99	1001 1001	
248	0010 0100 1000	

Advantage : At the time of input conversion from decimal to binary and at the time of output conversion from binary to decimal is not needed.

Disadvantage: Arithmetic and logical calculation becomes difficult to do. Representation of negative numbers is tricky.

Therefore, this approach is useful in the systems where there is much input/output than arithmetic and logical calculation

Alphanumeric Representation

- Alphanumeric character set is a set of elements that includes the 10 decimal digits, 26 letters of the alphabet and special characters such as $, %, + etc. The standard alphanumeric binary code is ASCII(American Standard Code for Information Interchange) which uses 7 bits to code 128 characters (both uppercase and lowercase letters, decimal digits and special characters).

Character	Binary code	Character	Binary code
A	100 0001	0	011 0000
B	100 0010	1	011 0001
C	100 0011	2	011 0010
D	100 0100	3	011 0011
E	100 0101	4	011 0100
F	100 0110	5	011 0101
G	100 0111	6	011 0110
H	100 1000	7	011 0111
I	100 1001	8	011 1000
J	100 1010	9	011 1001
K	100 1011		
L	100 1100		
M	100 1101	space	010 0000
N	100 1110	.	010 1110
O	100 1111	(	010 1000
P	101 0000	+	010 1011
Q	101 0001	$	010 0100
R	101 0010	*	010 1010
S	101 0011	)	010 1001
T	101 0100	−	010 1101
U	101 0101	/	010 1111
V	101 0110	,	010 1100
W	101 0111	=	011 1101
X	101 1000		
Y	101 1001		
Z	101 1010		

NOTE: Decimal digits in ASCII can be converted to BCD by removing the three higher order bits, 011.

Floating Point Representation

- The floating point representation of a number has two parts:
- The first part represents a signed fixed point number called **Mantissa.**
- The second part designates the position of decimal or binary point is called the **Exponent.**
- The fixed point mantissa may be a fraction or an integer. For example: The decimal number +6132.789 is represented in floating point with a fraction and an exponent as follows:

Fraction	**Exponent**
0.6132789	4

The value of the exponent indicates the actual position of the decimal point is 4 position to the right of the indicated decimal point in the fraction.

- Floating point is represented by $\mathbf{M*r^e}$
- Only the mantissa ‘M’ and the exponent ‘e’ are physically represented in the register including their sign. The radix ‘r’ and the radix point position of the Mantissa are always assumed.
- A floating point is said to be normalized if the MSB of the mantissa is non-zero.

Normalization

- A floating-point number is said to be normalized if the most significant digit of the mantissa is nonzero. For example, decimal number 350 is normalized but 00035 is not.

Examples

1. 525.360

Solution:

Fraction	Exponent
0.525360	3

M= 0.525360

e = 3

$M*r^e = 0.525360*10^3$

Normalization = $5.25360 *10^2$

2. -000845.67

Solution:

Fraction	Exponent
-0.84567	3

M= -0.84567

e = 3

$M*r^e = -0.84567*10^3$

Normalization = $- 8.4567 *10^2$

Examples

3. 0.000245

Solution:

Fraction	Exponent
0.245	- 3

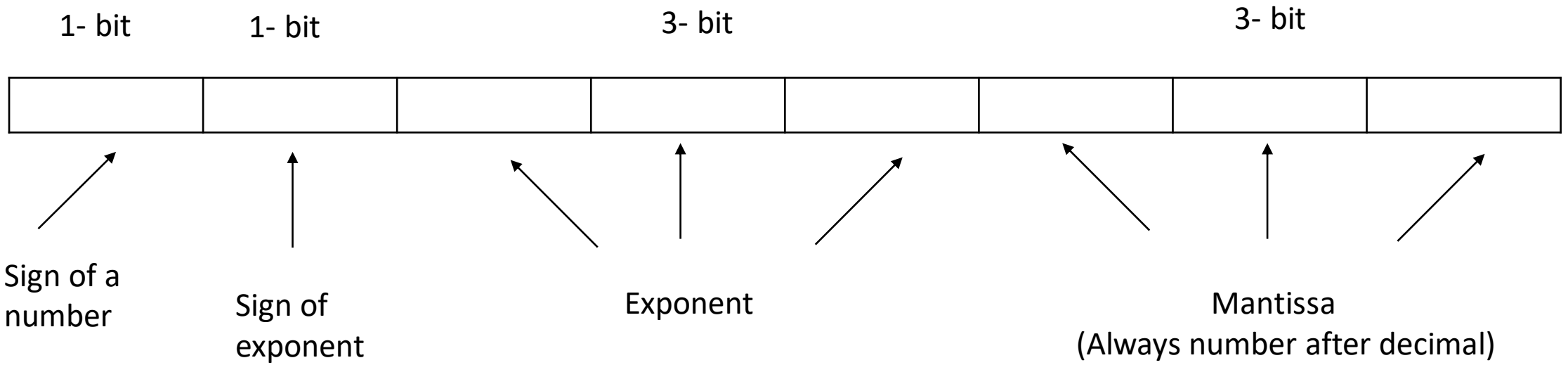
M= 0.245

e = - 3

$M*r^{e} = 0.245*10^{-3}$

Normalization = $2.45 *10^{-4}$

Convert decimal to binary : 8-bit word



Example:
Decimal (-13.9) into binary 8- bits

- Solution:

$(13)_{10} = (1101)_2$

$0.9 = 0.9 * 2 = 1.8 \quad 1$

$0.8 = 0.8 * 2 = 1.6 \quad 1$

$0.6 = 0.6 * 2 = 1.2 \quad 1$

$0.2 = 0.2 * 2 = 0.4 \quad 0$

$0.4 = 0.4 * 2 = 0.8 \quad 0$

$0.8 = 0.8 * 2 = 1.6 \quad 1$

$(0.9)_{10} = (0.111001)$

$(-13.9)_{10} = 1101.111001$

Normalized Form $= 1.101111001 * 2^3$

Exponent $= 3 = 011$

Sign = -ve = 1

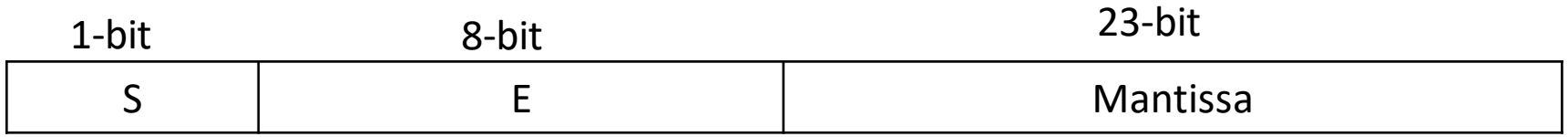
8-bit word

1	0	0	1	1	1	0	1

Single Precision Floating Point (32-bit)

- It is computer number format that occupies 4-bytes i.e. 32 bits in computer memory and represents a wide dynamic range of values by using a floating point. Single Precision binary floating point is used due to wider range over fixed point (of the same bit and width).

- Sign : 1- bit
- Exponent : 8-bits
- Significant precision = 24 bits (23 explicitly stored)
- Sign bit determined the sign of the number, which is the sign of the significand as well. Exponent is either an 8-bit signed integer from -128 to +127 or an 8-bit unsigned integer from 0 to 255.
- The true significand includes 23 fraction bits to the right of the binary point.



Sign bit

Exponent
Excess 127 binary integer added
Actual exponent is:
e= E -127
E = e + 127

Manitssa: normalized binary significand with a hidden integer bit: 1.M

Examples

1. Convert decimal $(0.75)_{10}$ to 32- bit single precision format.

Solution:

0.75 * 2 = 1.5 1

0.5 * 2 = 1.0 1

$(0.75)_{10} = (0.11)_2$

Normalized form = $1.1 * 2^{-1}$

The Mantissa is positive so S = 0

Exponent (e) = -1

E = -1 +127 = 126 = 01111110

Fractional Part of Mantissa is

M = 10000000000000000000000

The Single Precision Format is

0	01111110	10000000000000000000000

2. $(-2345.125)_{10}$

Solution:

2345 = 100100101001

0.125 * 2 = 0.25 - 0

0.25 * 2 = 0.5 - 0

0.5 * 2 = 1 - 1

$(-2345.125)_{10} = (100100101001.001)_2$

Normalized form = $1.00100101001001 * 2^{11}$

The Mantissa is negative so S = 1

Exponent (e) = 11

E = 11 +127 = 138 = 10001010

Fractional Part of Mantissa is

M = 00100101001001000000000

The Single Precision Format is

1	10001010	00100101001001000000000

$(5.2)_{10}$

Solution:

5 = 101

0.2 * 2 = 0.4 - 0

0.4 * 2 = 0.8 - 0

0.8 * 2 = 1.6 - 1

0.6 * 2 = 1.2 -1

0.2 * 2 = 0.4 - 0 (repetition starts)

$(5.2)_{10} = (101.0011)_2$

Normalized form = $1.010011 * 2^2$

The Mantissa is positive so S = 0

Exponent (e) = 2

E = 2 +127 = 129 = 10000001

Fractional Part of Mantissa is

M = 01001100110011001100110

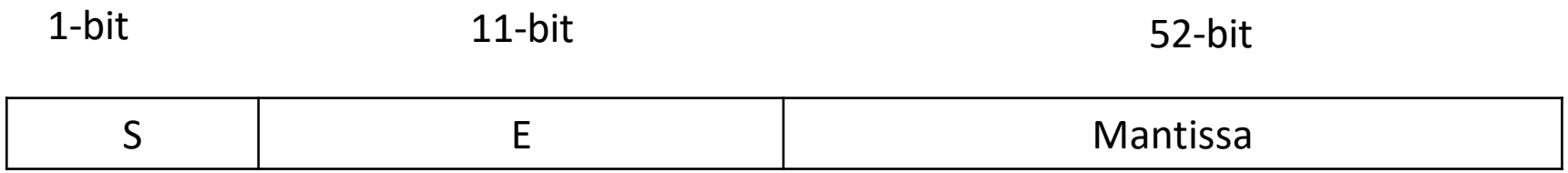
The Single Precision Format is

0	10000001	01001100110011001100110

Double Precision Floating Point (64-bit)

- It consists of 1 sign bit for mantissa, 11 bit of exponent including sign bit and remaining 52 bits for mantissa data / fraction.

$$\textbf{Value} = N = (-1)^{S} * 2^{E-1023} * (1.M)$$



Sign bit

Actual exponent is:
e= E -1023
E = e + 1023

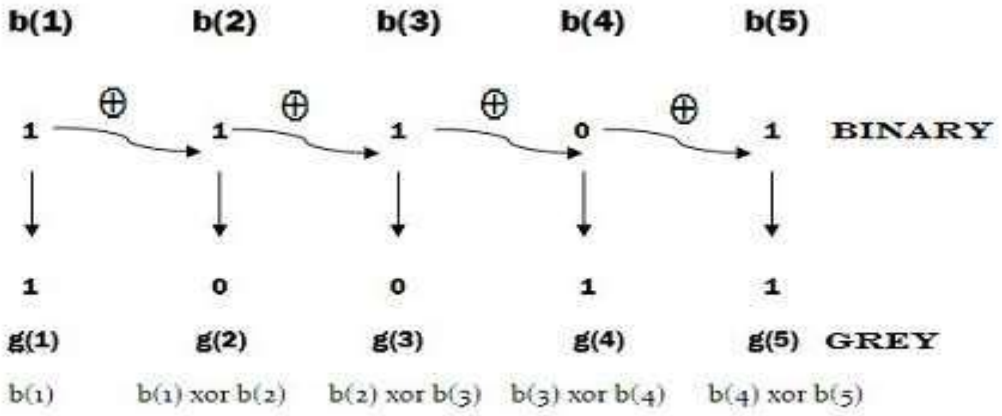
Manitssa: normalized binary significand with a hidden integer bit: 1.M

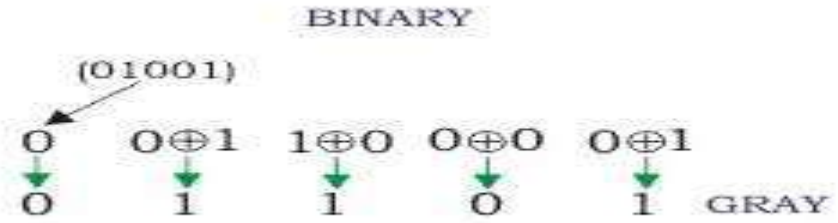
1.2 Other Binary Codes: Gray Code

- The reflected binary or Gray code is used to represent digital data converted from analog information.
- Gray code changes by only one bit as it sequences from one number to the next.
- Gray codes are very useful in the normal sequence of binary numbers generated by the hardware that may cause an error or ambiguity during the transition from one number to the next.
- So, the Gray code can eliminate this problem easily since only one bit changes its value during any transition between two numbers.

Convert Binary to Gray Code

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

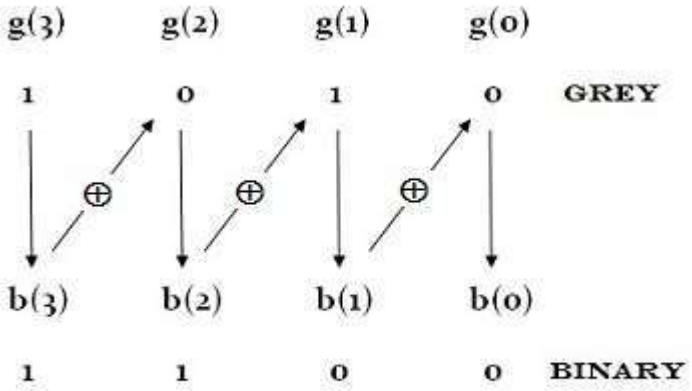




Gray Code to Binary Conversion

- **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**.



i.e

$$b(3) = g(3)$$

$$b(2) = b(3) \oplus g(2)$$

$$b(1) = b(2) \oplus g(1)$$

$$b(0) = b(1) \oplus g(0)$$

Converting Binary to Gray Code

Binary Code (Input)				Gray Code (Output)			
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

CD / AB	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

A

(a) *k-map for W*

CD / AB	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

$\bar{A}B$

$A\bar{B}$

(b) *k-map for X*

CD / AB	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	0	0

$\bar{B}C$

$B\bar{C}$

(c) *k-map for Y*

CD / AB	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

$\bar{C}D$

$C\bar{D}$

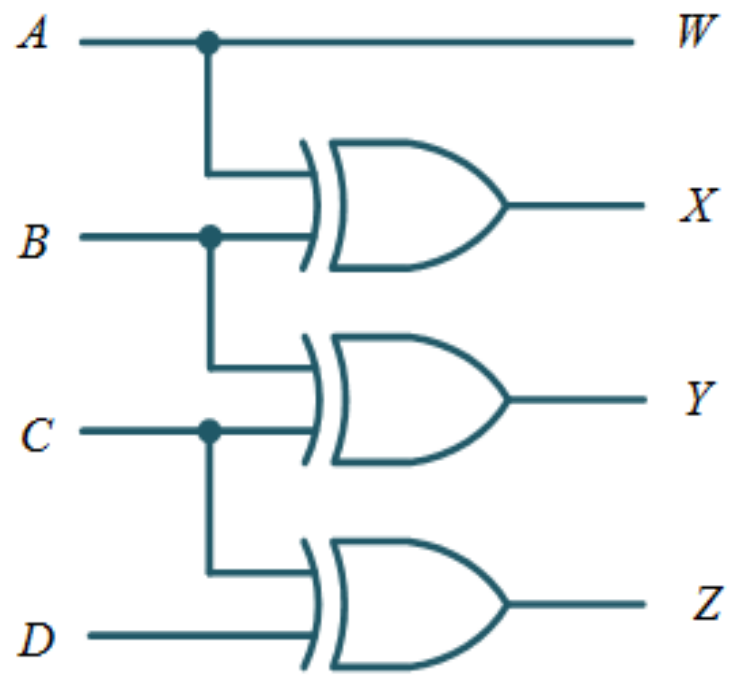
(d) *k-map for Z*

$$W = A$$

$$X = \bar{A}B + A\bar{B} = A \oplus B$$

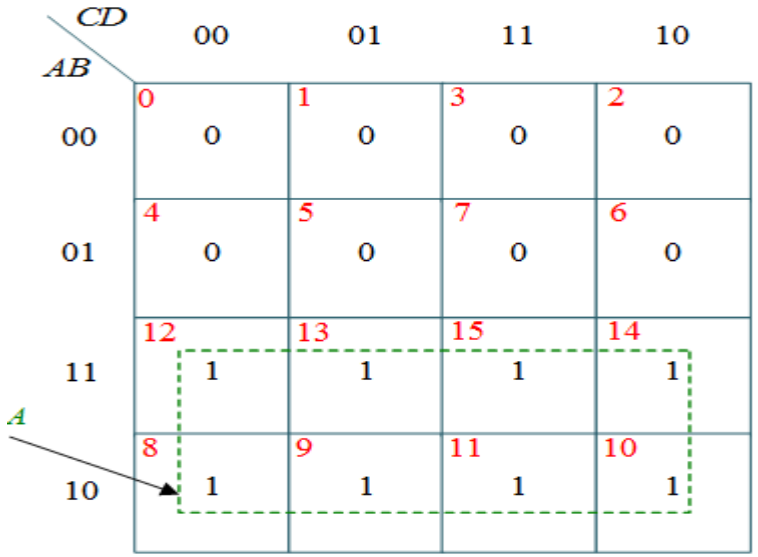
$$Y = \bar{B}C + B\bar{C} = B \oplus C$$

$$Z = \bar{C}D + C\bar{D} = C \oplus D$$

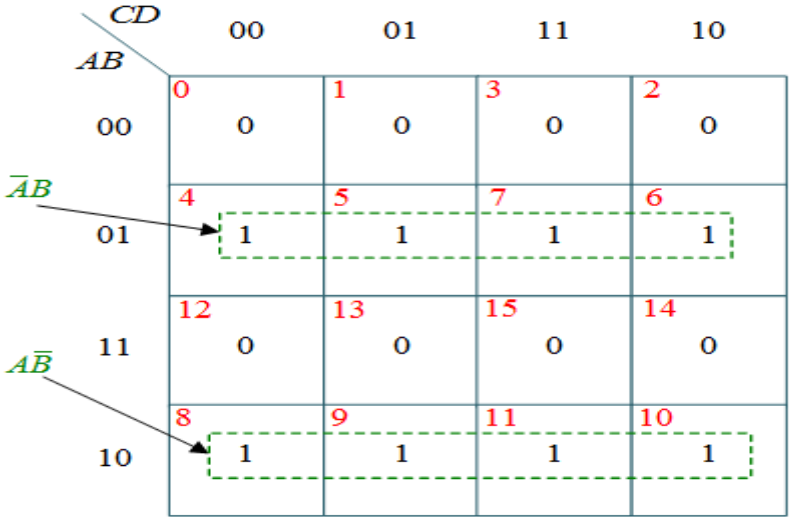


Converting Gray to Binary

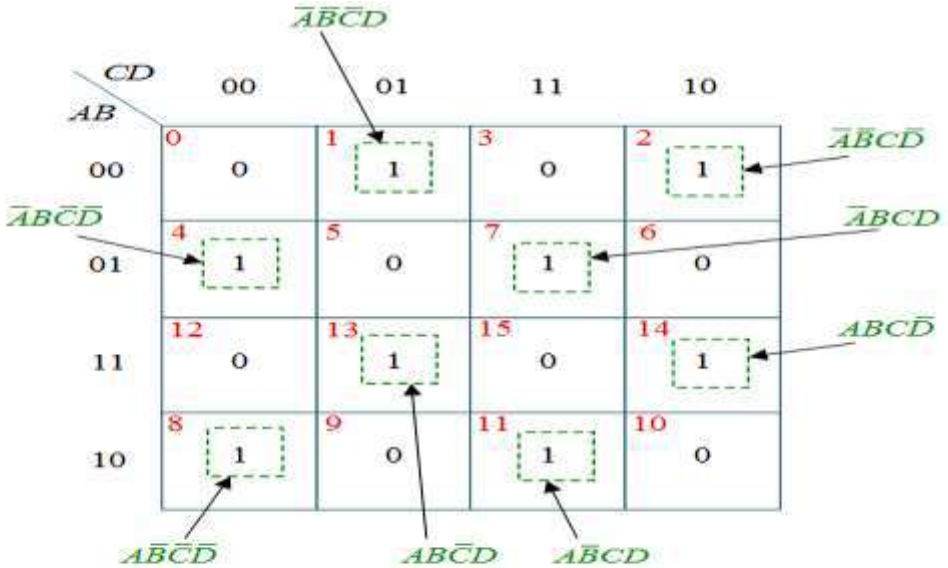
Gray Code (Input)				Binary Code (Output)			
A	*B*	*C*	*D*	*W*	*X*	*Y*	*Z*
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0



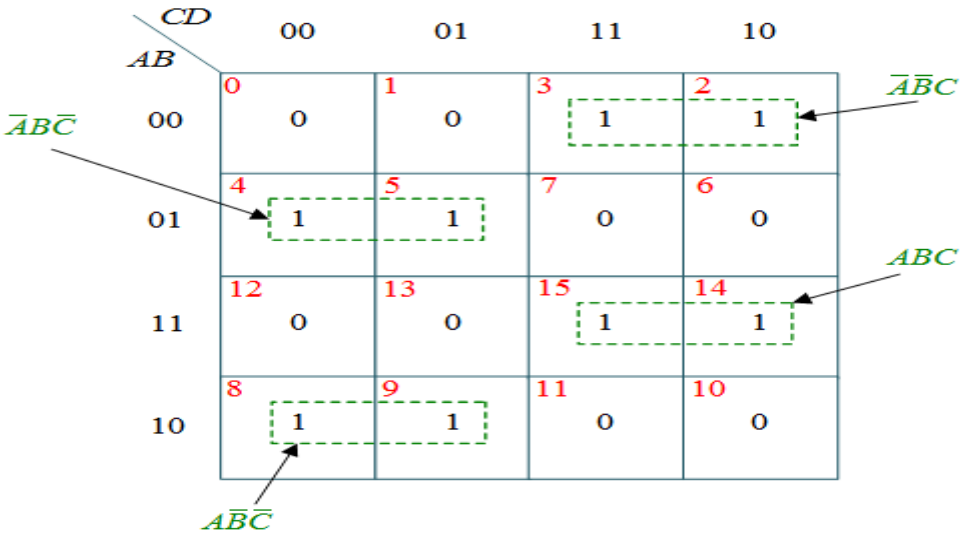
(a) *k-map for W*



(b) *k-map for X*



(c) *k-map for Y*



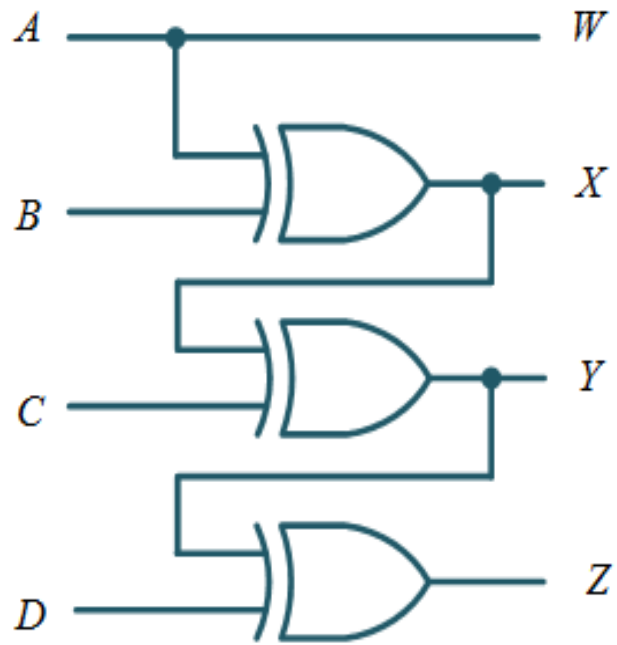
(d) *k-map for Z*

$$W = A$$

$$X = \bar{A}B + A\bar{B} = A \oplus B$$

$$Y = \bar{A}\bar{B}C + \bar{A}B\bar{C} + ABC + A\bar{B}\bar{C} = X \oplus C$$

$$Z = \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D}$$
$$+\bar{A}BCD + AB\bar{C}D + ABC\bar{D} + A\bar{B}\bar{C}\bar{D}$$
$$+A\bar{B}CD$$
$$= Y \oplus D$$



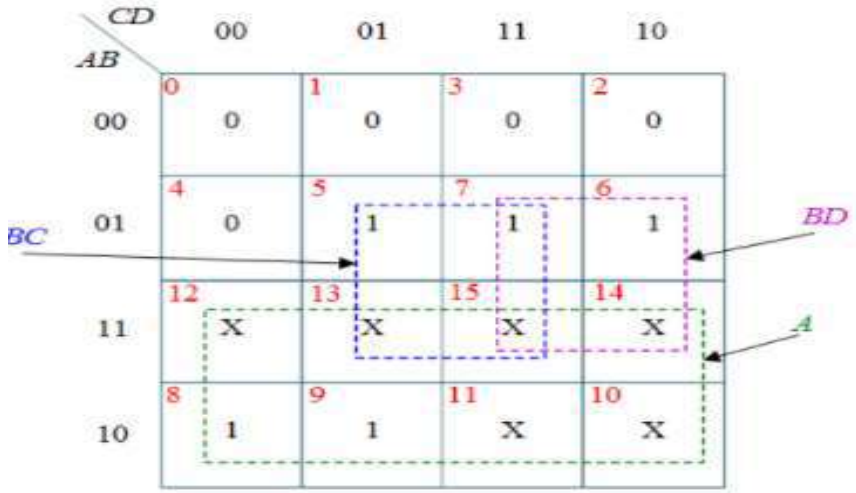
: *Gray to Binary Code Converter logic diagram.*

Excess 3 Code

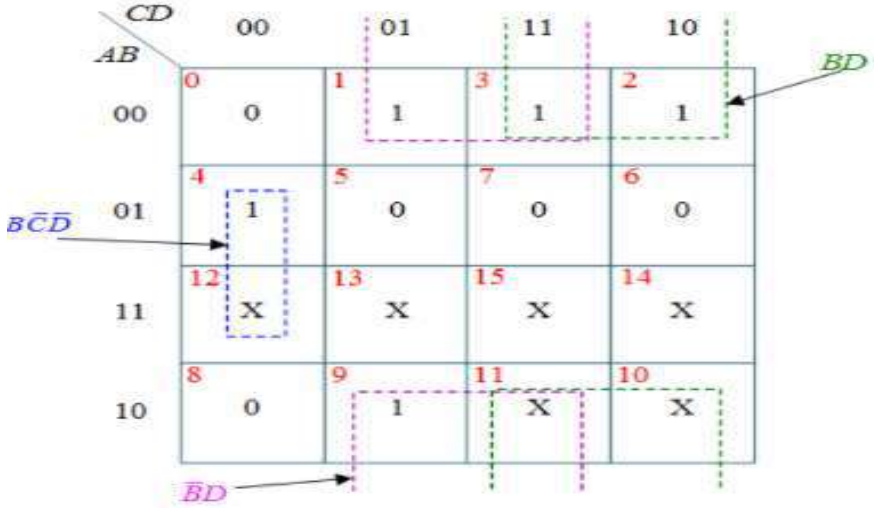
- Excess-3 codes are unweighted and can be obtained by adding 3 to each decimal digit then it can be represented by using 4 bit binary number for each digit. An Excess-3 equivalent of a given binary number is obtained using the following steps:
- Find the decimal equivalent of the given binary number.
- Add +3 to each digit of decimal number.
- Convert the newly obtained decimal number back to binary number to get required excess-3 equivalent.
- You can add 0011 to each four-bit group in binary coded decimal number (BCD) to get desired excess-3 equivalent.

BCD Code (Input)				Excess-3 Code (Output)			
A	*B*	*C*	*D*	*W*	*X*	*Y*	*Z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

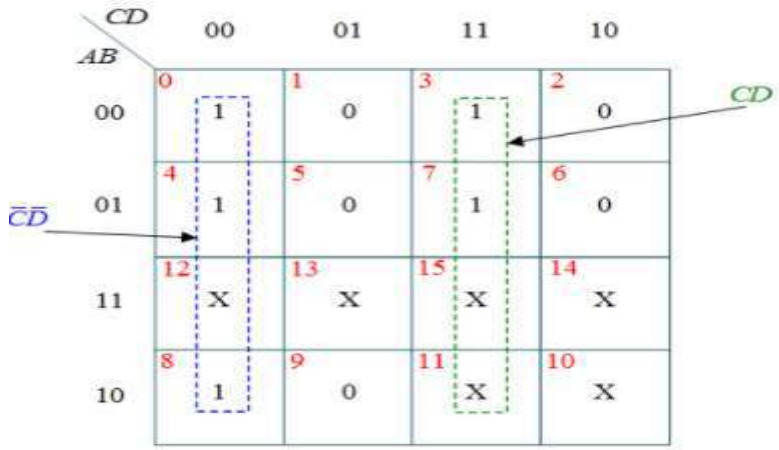
Excess-3 to BCD see in net



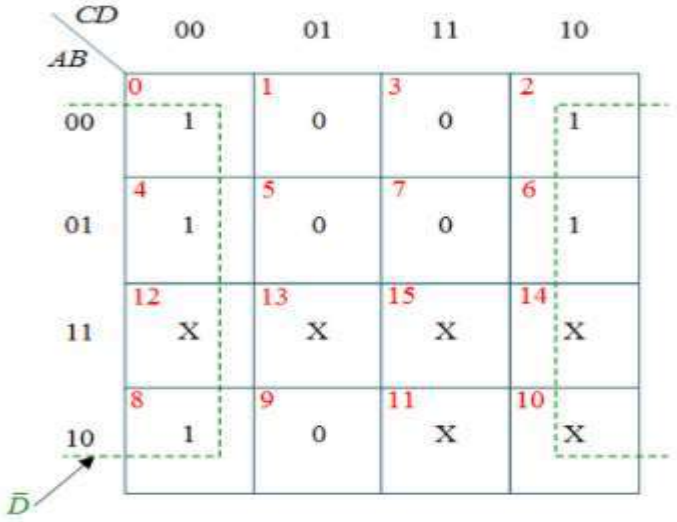
(a) *k-map for W*



(b) *k-map for X*



(c) *k-map for Y*



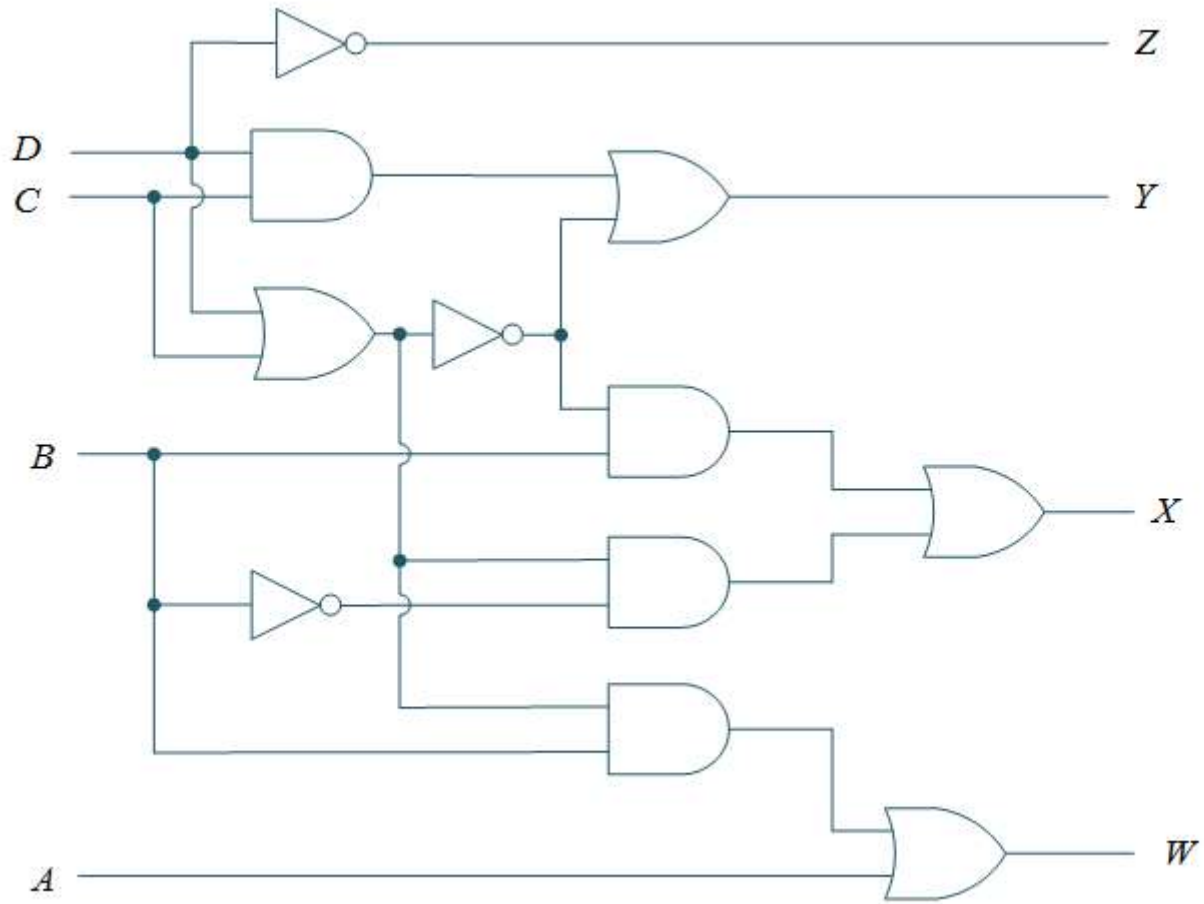
(d) *k-map for Z*

$$W = A + BC + BD$$

$$X = \bar{B}C + \bar{B}D + B\bar{C}\bar{D}$$

$$Y = CD + \bar{C}\bar{D}$$

$$Z = \bar{D}$$

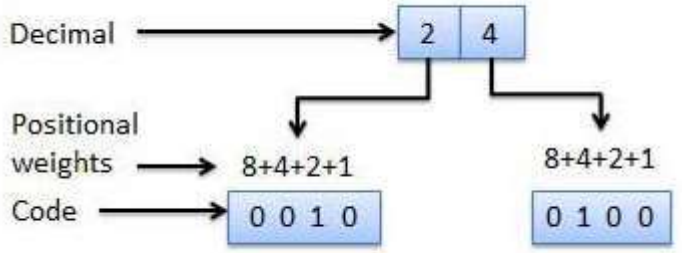


Binary Codes

- Binary codes are codes which are characterized in binary system with alteration from the original ones. Two types of it are the Weighted Binary Systems and Non Weighted Codes. Weighted binary codes are those which follow the positional weighting principles wherein each position of the number represents a specific weight. Like, 8421, 2421, and 5211 are weighted binary codes. Non weighted codes are codes that are not placed weighted. It means that each position within the binary number is not assigned a fixed value. Excess-3 and Gray codes are examples of non-weighted binary 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.



Decimal Digit	8421 Code	2421 Code
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	1011
6	0110	1100
7	0111	1101
8	1000	1110
9	1001	1111

8 4 2 1 code

•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 Binary Coded Decimal **code**.

2 4 2 1 code

•The weights of this code are 2, 4, 2 and 1.

•This code has all positive weights. So, it is a **positively weighted code**.

•It is an **unnatural BCD** code. Sum of weights of unnatural BCD codes is equal to 9.

•It is a **self-complementing** code. Self-complementing codes provide the 9's complement of a decimal number, just by interchanging 1's and 0's in its equivalent 2421 representation.

Example

Let us find the BCD equivalent of the decimal number 786. This number has 3 decimal digits 7, 8 and 6. From the table, we can write the BCD 8421 codes of 7, 8 and 6 are 0111, 1000 and 0110 respectively.

$$\therefore \quad 786_{10} = 011110000110_{BCD}$$

There are 12 bits in BCD representation, since each BCD code of decimal digit has 4 bits.

Example

Let us find the 2421 equivalent of the decimal number 786. This number has 3 decimal digits 7, 8 and 6. From the table, we can write the 2421 codes of 7, 8 and 6 are 1101, 1110 and 1100 respectively.

Therefore, the 2421 equivalent of the decimal number 786 is **110111101100**.

Self Complementing Codes:

- Self complementing binary codes are those who members complement on themselves. For a binary code to become a self- complementing code, the following two conditions must be satisfied:

1. The complement of a binary number should be obtained from that number by replacing 1’s with 0’s and 0’s with 1’s.
2. The sum of the binary number and its complement should be equal to decimal 9.

Example:

Excess-3 Code for 2 is 2+3 =5 = 0101

1’s complement of 0101 is 1010 - this is excess-3 code for decimal 7

9’s complement of 2 is : 9-2 =7

Which means 1’s complement of the coded number yields 9’s complement of number itself.

EBCDIC:

- EBCDIC is short for extended binary coded decimal interchange code is eight bits, or one byte, wide. This is a coding system used to represent characters-letters, numerals, punctuation marks, and other symbols in computerized text. A character is represented in EBCDIC by eight bit.
- EBCDIC is different from, and incompatible with, the ASCII character set used by all other computers. The EBCDIC code allows for 256 different characters. For personal computers, however, ASCII is the standard.

1.3: Error Detection Codes

- Binary information transmitted through some form of communication medium is subjected to external noise that could change bit from 1 to 0 and vice versa.
- An error detection code is binary code that detects error during transmission. The detected errors cannot be corrected but their presence is indicated.
- The usual procedure is to observe the frequency of errors. If errors occur too often then the system is checked for malfunctioning.
- Parity bit is used for detecting errors during transmission of binary information.
- A **parity bit** is an extra bit included with a binary message to make the total number of 1's either even or odd.
- **Even Parity:**

One bit is attached to the information so that the total number of **1** bits is an even numbers.

Example: 1011001 **0**

1010010 **1**

- **Odd Parity:**

One bit is attached to the information so that the total number of **1** bits is an odd numbers.

Example: 1011001 **1**

1010010 **0**

- The circuit that generates the parity bit in the transmitter is called the **parity generator** and the circuit that checks the parity bit in the receiver id called a **parity checker.**
- Parity generator and checker networks are logic circuits constructed with exclusive OR functions.
- Consider a 3-bit message to be transmitted with an odd parity. At the sending end, the odd parity is generated by a parity generator circuit. The output of the parity checker would be one when an error occurs i.e. number of 1's in the four inputs is even.

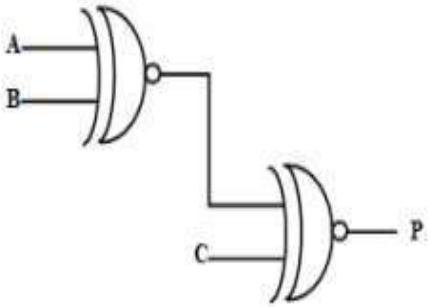
- **Parity generator**

3-bit message			Odd parity bit generator (P)
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

A \ BC	00	01	11	10
0	1	0	1	0
1	0	1	0	1

The output parity bit expression for this generator circuit is obtained as

$$P = (A \oplus B \oplus C)'$$

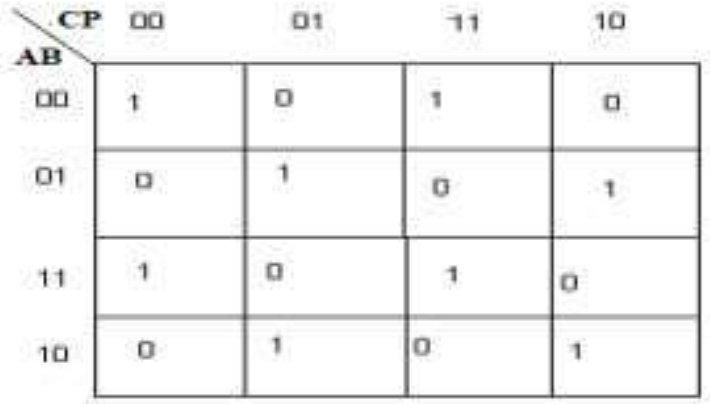


Odd Parity Generator for three bit Data Word

- **Parity Checker**

Consider an original message as well as parity bit.

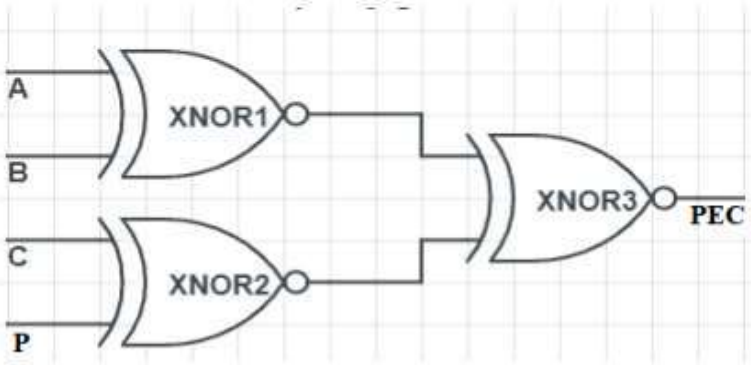
4-bit received message				Parity error check C_p
A	B	C	P	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1



After simplification, the final expression for the PEC is obtained as

PEC = (A Ex-NOR B) Ex-NOR (C Ex-NOR D)

The expression for the odd parity checker can be designed by using three Ex-NOR gates as shown below.



Circuit diagram for parity generator and parity checker

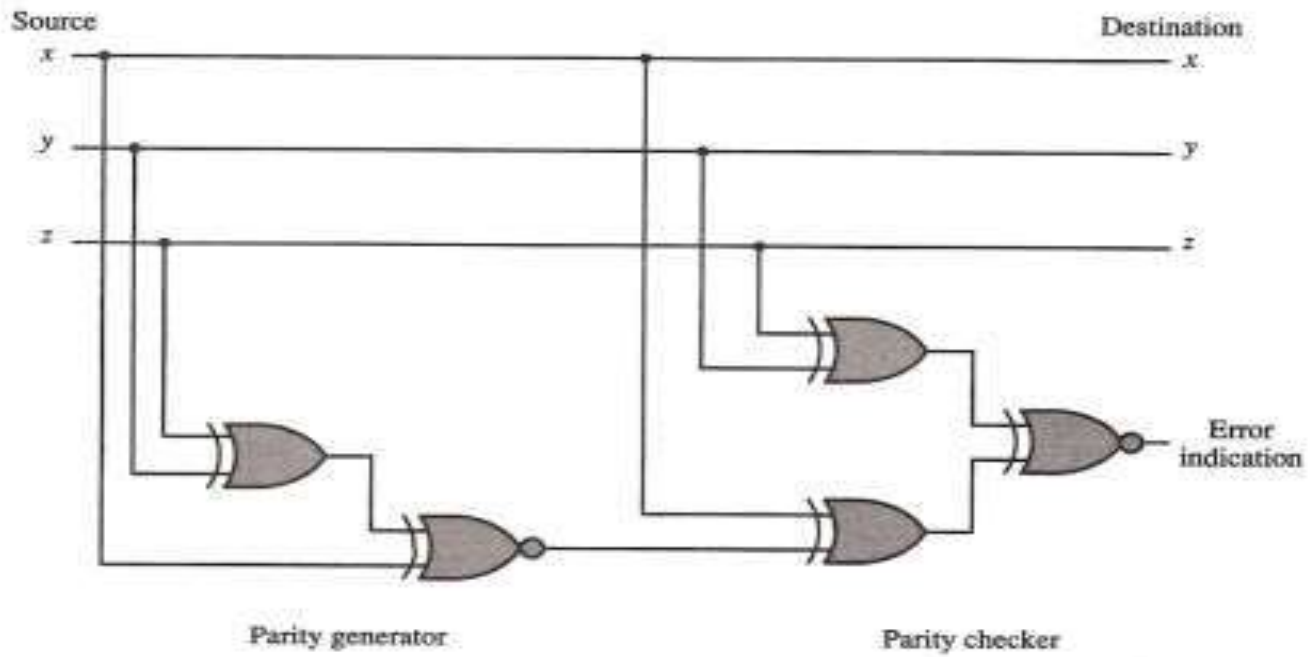


Fig: Error detection with odd parity bit.

For even parity: do it on your own