

# **Data Representation**

# Contents

- Data Representation: Binary Representation, BCD, Alphanumeric Representation, Complements, Fixed Point representation, Representing Negative Numbers, Floating Point Representation, Arithmetic with Complements, Overflow, Detecting Overflow
- Other Binary Codes: Gray Code, self Complementing Code, Weighted Code, Excess-3 Code, EBCDIC
- Error Detection Codes: Parity Bit, Odd Parity, Even parity, Parity Generator & Checker

## 1.1 Data Types

- Binary information in digital computer is stored in memory or processor registers. Registers contain either data or control information .
- Control information is a bit or group of bits used to specify the sequence of command signals needed for data manipulation .
- Data are numbers and other binary-coded information that are operated on
- Possible data types in registers:
  - Numbers used in arithmetic computations
  - Letters of the alphabet used in data processing
  - Other discrete symbols used for specific purposes
- All types of data, except binary numbers, are represented in binary-coded form.

## 1.2 Number Systems

- A number system of base, or radix,  $r$  is a system that uses distinct symbols for  $r$  digits.
- Numbers are represented by a string of digit symbols. To determine the quantity that the number represents, it is necessary to multiply each digit by an integer power of  $r$  and then form the sum of all weighted digits.
- For example, the **decimal number system** in everyday use employs the radix 10 system.
- The 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.
- The string of digits 724.5 is interpreted to represent the quantity
$$(724.5)_{10} = 7 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 + 5 \times 10^{-1}$$

- The **binary number system** uses the radix 2. The two digit symbols used are 0 and 1.
- The string of digits 101101 is interpreted to represent the quantity
$$101101 \Rightarrow 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 45$$
- To show the equality between decimal and binary forty-five we will write  $(101101)_2 = (45)_{10}$

## **Octal and Hexadecimal Number system**

- **Octal (radix 8)** and **hexadecimal (radix 16)** number systems
- The eight symbols of the octal system are 0, 1, 2, 3, 4, 5, 6, and 7. The 16 symbols of the hexadecimal system are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

- The symbols A, B, C, D, E, F correspond to the decimal numbers 10, 11, 12, 13, 14, 15, respectively.
- For example, octal 736.4 is converted to decimal as follows:

$$\begin{aligned}
 (736.4)_8 &= 7 \times 8^2 + 3 \times 8^1 + 6 \times 8^0 + 4 \times 8^{-1} \\
 &= 7 \times 64 + 3 \times 8 + 6 \times 1 + 4/8 \\
 &= (478.5)_{10}
 \end{aligned}$$

- The equivalent decimal number of hexadecimal F3 as:

$$(F3)_{16} = F \times 16 + 3 = 15 \times 16 + 3 = (243)_{10}$$

- Decimal to binary conversion(float numbers)

Example: 28.125 and 28 and 0.125

$$\begin{array}{ll}
 0.125 \times 2 = .250 & 0 \\
 .250 \times 2 = .500 & 0 \\
 .500 \times 2 = 1.00 & 1
 \end{array}
 \begin{array}{c}
 | \\
 \downarrow
 \end{array}$$

0.125=001 so, 28.125=11100.001

division = quotient + remainder ;

$$28 \div 2 = 14 + 0;$$

$$14 \div 2 = 7 + 0;$$

$$7 \div 2 = 3 + 1;$$

$$3 \div 2 = 1 + 1;$$

$$1 \div 2 = 0 + 1;$$

$$\text{So, } 28 = 11100$$

- Conversion from decimal to radix  $r$  system is carried out by separating the number into its integer and fraction parts and converting each part separately
- Divide the integer successively by  $r$  and accumulate the remainders
- Multiply the fraction successively by  $r$  until the fraction becomes zero

<p><b>Integer = 41</b></p> <div style="display: flex; align-items: center;"> <div style="text-align: right; padding-right: 10px;"> 41 20 10 5 2 1 0 </div> <div style="border-left: 1px solid black; padding-left: 10px; text-align: center;">  1 0 0 1 0 1 </div> </div> <p><math>(41)_{10} = (101001)_2</math></p>	<p><b>Fraction = 0.6875</b></p> <div style="display: flex; align-items: center;"> <div style="text-align: right; padding-right: 10px;"> 0.6875 <u>2</u> 1.3750 x 2 0.7500 x 2 1.5000 x 2 1.0000 </div> <div style="text-align: center;">  2 1 1 1 </div> </div> <p><math>(0.6875)_{10} = (0.1011)_2</math></p>
<p><math>(41.6875)_{10} = (101001.1011)_2</math></p>	

Figure 1. Conversion of decimal 41.6875 into binary.

- Each octal digit corresponds to three binary digits
- Each hexadecimal digit corresponds to four binary digits
- Rather than specifying numbers in binary form, refer to them in octal or hexadecimal and reduce the number of digits by 1/3 or 1/4, respectively

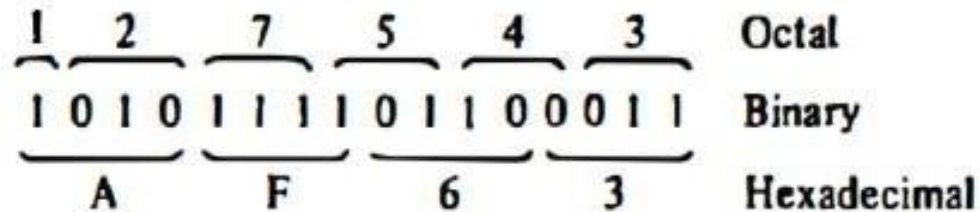


Figure 2. Binary ,octal and hexadecimal conversion.



Q. Convert  $(761)_8 = ( )_2$  and  $(71.3)_8 = ( )_2$  ?

Solution:  $7=111, 6=110, 1=001, 3=011$

$(761)_8 = (111110001)_2$  and  $(71.3)_8 = (111001.011)_2$

Q. Convert  $(10011)_2 = ( )_8$  and  $(10.11)_2 = ( )_8$

Solution: Group 3 bits,

$(10011)_2 \Rightarrow 010=2$  and  $011=3$  so,  $(23)_8$

$(10.11)_2 \Rightarrow 010=2$  and  $110=6$  so,  $(2.6)_8$

Q. Convert  $(61)_{16} = ( )_2$  and  $(8A.D)_{16} = ( )_2$  ?

Solution: Group 4 bits,  $6=0110$  and  $1=0001$  So,  $(01100001)_2$

$8=1000$   $A=1010$  and  $D=1101$  So,  $(100010101.1101)_2$



Q. Convert  $(7A.c)_{16} = ( )_8$  ?

Solution:  $7=0111$   $A=1010$  and  $C=1100$  so,  $(01111010.1100)_2$

Group 3 bits using table now,  $(172.60)_8$

Octal number	Binary-coded octal	Decimal equivalent	
0	000	0	↑ Code for one octal digit ↓
1	001	1	
2	010	2	
3	011	3	
4	100	4	
5	101	5	
6	110	6	
7	111	7	
10	001 000	8	
11	001 001	9	
12	001 010	10	
24	010 100	20	
62	110 010	50	
143	001 100 011	99	
370	011 111 000	248	

**Table 1.1 Binary-Coded Octal Numbers**

Hexadecimal number	Binary-coded hexadecimal	Decimal equivalent	
0	0000	0	 Code for one hexadecimal digit 
1	0001	1	
2	0010	2	
3	0011	3	
4	0100	4	
5	0101	5	
6	0110	6	
7	0111	7	
8	1000	8	
9	1001	9	
A	1010	10	
B	1011	11	
C	1100	12	
D	1101	13	
E	1110	14	
F	1111	15	
14	0001 0100	20	
32	0011 0010	50	
63	0110 0011	99	
F8	1111 1000	248	

**Table 1.2 Binary-Coded hexadecimal Numbers**

- A binary code is a group of  $n$  bits that assume up to  $2^n$  distinct combinations of 1's and 0's with each combination representing one element of the set that is being coded.
- For example, a set of four elements can be coded by a 2-bit code with each element assigned one of the following bit combinations; 00, 01, 10, or 11. A set of eight elements requires a 3-bit code, a set of 16 elements requires a 4-bit code, and so on.
- A binary code that distinguishes among 10 elements must contain at least four bits, but six combinations will remain unassigned.

## Binary-coded Decimal (BCD)

- Also **8421 Code** .
- The most popular decimal code is called **binary-coded decimal (BCD)** .
- For example, when converted to a binary number, the decimal number 99 is represented by the string of bits 1100011, but when represented in BCD, it becomes 1001 1001.
- The only difference between a decimal number represented by the familiar digit symbols 0, 1, 2, ... 9 and the BCD symbols 0001,0010, ....., 1001 is in the symbols used to represent the digits-the number itself is exactly the


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	

Table 1.3 Binary-Coded decimal (BCD) Numbers

## **Alphanumeric Representation**

- An alphanumeric character set is a set of elements that includes the 10 decimal digits, the 26 letters of the alphabet and a number of special characters, such as \$, +, and = .
- Such a set contains between 32 and 64 elements (if only uppercase letters are included) or between 64 and 128 (if both uppercase and lowercase letters are included). In the first case, the binary code will require six bits and in the second case, seven bits.
- The standard alphanumeric binary code is ASCII (American Standard Code for Information Interchange), which uses seven bits to code 128 characters.
- Binary codes are required since registers can hold binary information only.

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		

Table 1.4 American Standard Code for Information Interchange (ASCII)



## 1.3 Complements

- Complements are used in digital computers for simplifying subtraction and logical manipulation
- Two types of complements for each base  $r$  system:
  - $r$ 's complement and
  - $(r - 1)$ 's complement
- When the value of the base  $r$  is substituted in the name, the two types are referred to as the 2's and 1's complement for binary numbers and the 10's and 9's complement for decimal numbers.

## **$(r - 1)$ 's Complement**

- Given a number  $N$  in base  $r$  having  $n$  digits, the  $(r - 1)$ 's complement of  $N$  is defined as  $(r^n - 1) - N$ .
- For decimal numbers  $r = 10$  and  $r - 1 = 9$ , so the **9's complement** of  $N$  is  $(10^n - 1) - N$ . Now,  $10^n$  represents a number that consists of a single 1 followed by  $n$  0's.
- $10^n - 1$  is a number represented by  $n$  9's. For example, with  $n = 4$  we have  $10^4 = 10000$  and  $10^4 - 1 = 9999$ .
- It shows that the 9's complement of a decimal number is obtained by subtracting each digit from 9.
- For example, the 9's complement of 546700 is  $999999 - 546700 = 453299$  and the 9's complement of 12389 is  $99999 - 12389 = 87610$ .

- For binary numbers,  $r = 2$  and  $r - 1 = 1$ , so the 1's complement of  $N$  is  $(2^n - 1) - N$ . Again,  $2^n$  is represented by a binary number that consists of a 1 followed by  $n$  0's.  $2^n - 1$  is a binary number represented by  $n$  1's.
- For example, with  $n = 4$ , we have  $2^n = (10000)_2$ , and  $2^4 - 1 = (1111)_2$ .
- Thus, the **1's complement** of a binary number is obtained by subtracting each digit from 1.
- However, the subtraction of a binary digit from 1 causes the bit to change from 0 to 1 or from 1 to 0.
- For example, the 1's complement of 1011001 is 0100110 and the 1's complement of 0001111 is 1110000.

## (r's) Complement

- The  $r$ 's complement of an  $n$ -digit number  $N$  in base  $r$  is defined as  $r^n - N$  for  $N \neq 0$  and  $0$  for  $N = 0$ .
- Comparing with the  $(r - 1)$ 's complement, we note that the  $r$ 's complement is obtained by adding 1 to the  $(r - 1)$ 's complement since  $r^n - N = [(r^n - 1) - N] + 1$ .
- Thus the 10's complement of the decimal 2389 is  $7610 + 1 = 7611$  and is obtained by adding 1 to the 9's complement value.
- The **2's complement** of binary 101100 is  $010011 + 1 = 010100$  and is obtained by adding 1 to the 1's complement value.

## Subtraction of Unsigned Numbers

Subtraction of unsigned  $n$ -digit numbers:  $M - N$

- Add  $M$  to the  $r$ 's complement of  $N$  – this results in  $M + (r^n - N) = M - N + r^n$
- If  $M \geq N$ , the sum will produce an end carry  $r^n$  which is discarded
- If  $M < N$ , the sum does not produce an end carry and is equal to  $r^n - (N - M)$ , which is the  $r$ 's complement of  $(N - M)$ . To obtain the answer in a familiar form, take the  $r$ 's complement of the sum and place a negative sign in front.

- Example 1: the subtraction  $72532 - 13250 = 59282$ . The 10's complement of 13250 is 86750. Therefore:

$$M = 72532$$

$$\text{10's complement of } N = +86750$$

$$\text{Sum} = 159282$$

$$\text{Discard end carry} = -100000$$

$$\text{Answer} = 59282$$

Note : Both numbers must have the same number of digits; if not add 0 .eg.  
 $72532 - 3250$  then  $72532 - 03250$ .

- Example for  $M < N$ :  $13250 - 72532 = -59282$

$$M = 13250$$

$$\text{10's comp. of } N = +\underline{27468}$$

$$\text{Sum} = 40718$$

(There is no end carry)

$$\text{Answer} = -59282 \text{ (10's comp. of 40718)}$$

- Example for  $X = 1010100$  and  $Y = 1000011$  we perform the subtraction  $X - Y$  and  $Y - X$  using 2's complements:

$$X = 1010100$$

$$\text{2's comp. of } Y = +\underline{0111101}$$

$$\text{Sum} = 10010001$$

$$\text{Discard end carry} = -\underline{10000000}$$

$$\text{Answer } X - Y = 0010001$$

$$Y = 1000011$$

$$\text{2's comp. of } X = +\underline{0101100}$$

$$\text{Sum} = 1101111$$

No end carry

$$\text{Answer} = -0010001 \text{ (2's comp. of } 1101111\text{)}$$

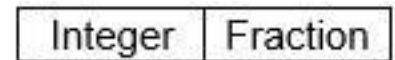
## Fixed-point representation

- Fixed number of bits for integer part and for fractional part.
- The fixed-point numbers in binary uses a sign bit.
- Unsigned Notation : A positive number has a sign bit 0, while Signed Notation: the negative number has a sign bit 1.

- Two ways to designate binary point position in a register

- Fixed point position

Unsigned fixed point



- Floating-point representation

Signed fixed point



- Less hardware required for processing.
- Fixed point position usually uses one of the two following positions
  - A binary point in the extreme left of the register to make it a fraction
  - A binary point in the extreme right of the register to make it an integer
  - In both cases, a binary point is not actually present



## Representing Negative Numbers

- A negative number can be represented in one of the following ways:
  - Signed magnitude representation
  - Signed 1 's complement representation, or
  - Signed 2's complement representation.

Examples 1:

Assumption: size of register=8bits including the sign bit.

- Signed magnitude representation

+6 = 0 0000110      =>( sign bit(1 bit) and (7 bits) magnitude)

-6= 1 0000110

No change in the magnitude, only sign bit changes

- Signed 1 's complement representation

+6= 0 0000110

-6= 1 1111001

- For negative number take 1's complement of all the bits( including sign bit) of the positive number then add 1.

- Signed 2's complement representation.

$$+6 = 0\ 0000110$$

$$-6 = 11111010\ (1's\ complement + 1)$$

For negative number take 2's complement of all the bits (including sign bit) the positive number.

Example 2: Consider an 8-bit register and the number +14

- The only way to represent it is 00001110
- Consider an 8-bit register and the number -14
  - Signed magnitude:  $1\ 0001110$
  - Signed 1's complement:  $1\ 1110001$
  - Signed 2's complement:  $1\ 1110010$

## Arithmetic Addition

- Addition of two signed-magnitude numbers follow the normal rules
  - If same signs, add the two magnitudes and use the common sign
  - Different signs, subtract the smaller from the larger and use the sign of the larger magnitude
  - Must compare the signs and magnitudes and then either add or subtract

For example,

$(+25) + (-37) = -(37 - 25) = -12$  and is done by subtracting the smaller magnitude 25 from the larger magnitude 37 and using the sign of 37 for the sign of the result

- Addition of two signed 2's complement numbers does not require a comparison or subtraction – only addition and complementation
  - Add the two numbers, including their sign bits
  - Discard any carry out of the sign bit(leftmost) position
  - All negative numbers must be in the 2's complement form
  - If the sum obtained is negative, then it is in 2's complement form

Examples:

+6	00000110	-6	11111010
+13	<u>00001101</u>	+13	<u>00001101</u>
+19	00010011	+7	00000111

+6	00000110	-6	11111010
-13	<u>11110011</u>	-13	<u>11110011</u>
-7	11111001	-19	11101101

## Arithmetic Subtraction

- Subtraction of two signed 2's complement numbers is as follows
  - Take the 2's complement form of the subtrahend (including sign bit)
  - Add it to the minuend (including the sign bit)
  - A carry out of the sign bit position is discarded
- This procedure stems from the fact that a subtraction operation can be changed to an 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 eight bits this is written as  $11111010 - 11110011$ . The subtraction is changed to addition by taking the 2's complement of the subtrahend  $(-13)$  to give  $(+13)$ . In binary this is  $11111010 + 00001101 = 100000111$ . Removing the end carry, we obtain the correct answer  $00000111 (+7)$ .

# Overflow

- An overflow occurs when two numbers of  $n$  digits each are added and the sum occupies  $n + 1$  digits
- An overflow is a problem in digital computers because the width of registers is finite.
- Therefore, when it occurs, a corresponding flip-flop is set which can then be checked by the user
- Detection of an overflow depends on if the numbers are signed or unsigned
- For unsigned numbers addition, an overflow is detected from the end carry out of the MSB
- For addition of signed numbers, an overflow cannot occur if one is positive and one is negative – both have to have the same sign
- An overflow can be detected if the carry into the sign bit position and the carry out of the sign bit position are not equal

Example:

- Two signed binary numbers, +70 and +80, are stored in two 8 bit registers. Range from binary 127 to binary -128. since sum is +150
- Overflow occurred if the numbers are both +ve or both -ve.

Carries: 0 1

+70      0 1000110

+80      0 1010000

+150     1 0010110

Carries: 1 0

-70    1 0111010

-80    1 0110000

-150   0 1101010

## Decimal Fixed-Point Representation

- The representation of decimal numbers in registers is a function of the binary code used to represent a decimal digit .
- A 4-bit decimal code requires four flip-flops for each decimal digit.
- The representation of 4385 in BCD requires 16 flip-flops, four flip-flops for each digit. The number will be represented in a register with 16 flip-flops as follows: 0100 0011 1000 0101
- This takes much more space than the equivalent binary representation and the circuits required to perform decimal arithmetic are more complex.
- This eliminates the need for conversion to binary and back to decimal .
- Representation of signed decimal numbers in BCD is similar to the representation of signed numbers in binary.
- Either signed magnitude or signed complement systems



- **It is customary** to designate a plus with four 0' s and a minus with the BCD equivalent of 9, which is 1001 .
- The sign of a number is represented with four bits  
0000 for +  
1001 for –
- To obtain the 10's complement of a BCD number, first take the 9's complement and then add one to the least significant digit

Example:  $(+375) + (-240) = +135$

$$\begin{array}{rcl}
 0 \ 375 & (0000 \ 0011 \ 0111 \ 0101)_{\text{BCD}} & \\
 +9 \ 760 & \underline{(1001 \ 0111 \ 0110 \ 0000)_{\text{BCD}}} & \\
 0 \ 135 & (0000 \ 0001 \ 0011 \ 0101)_{\text{BCD}} & \text{(End carry discarded)}
 \end{array}$$

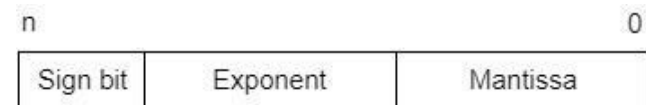
The 9 in the leftmost position of the second number indicates that the number is negative. 9 760 is the 10's complement of 0240. The two numbers are added and the end carry is discarded to obtain + 135.

## Floating-Point Representation

- The floating-point representation of a number has two parts
- The first part represents a signed, fixed-point number – the *mantissa*
- The second part designates the position of the binary point – the *exponent*
- The mantissa may be a fraction or an integer

### Decimal Value

$$N = m * r^e$$



Where , m is mantissa, r is base ,e is exponent

$$11 * 10^8 \Rightarrow m=11, r=10, e=8$$

Example 1: the decimal number  $N = +6132.789$  is

Fraction (m): +6132789

Exponent (e): +04 , r=10

Equivalent to the scientific notation  $N = m * r^e = +0.6132789 \times 10^{+4}$

- A floating-point number is always interpreted to represent  $m \times r^e$

Example 2: the binary number +1001.11 (with 8-bit fraction and 6-bit exponent)

Fraction (m): 01001110

Exponent (e): +4 = 000100

$r=2$

Sign bit=0

Equivalent to  $+(.1001110)_2 \times 2^{+4}$

- A floating-point number is said to be **normalized** if the most significant digit of the mantissa is nonzero digit.

- The decimal number 350 is normalized, 00350 is not.
- The 8-bit number 00011010 is not normalized because the three leading 0's.
- Normalize it by fraction = 11010000 and exponent = -3
- Normalized numbers provide the maximum possible precision for the floating-point number.

## 1.4 Other Binary Codes

- Digital systems can process data in discrete form only.
- Continuous, or analog, information is converted into digital form by means of an analog-to-digital converter
- The reflected binary or **Gray code**, is sometimes used for the converted digital data .
- The advantage of the Gray code over straight binary numbers is that the Gray code changes by only one bit as it sequences from one number to the next.
- In other words, the change from any number to the next in sequence is recognized by a change of only one bit from 0 to 1 or from 1 to 0.
- Gray code counters are sometimes used to provide the timing sequences that control the operations in a digital system
- Binary codes for decimal digits require a minimum of four bits.
- Other codes besides BCD exist to represent decimal digits.

- Gray codes are used in the general sequence of hardware-generated binary numbers.
- Cause ambiguities or errors when the transition from one number to its successive is done.
- This code simply solves this problem by changing only one bit when the transition is between numbers is done.
- Exclusive –OR operation used during conversion.

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

**Table 1.5 : 4-bit Gray Code**

Decimal	Binary Code (input)	Gray Code (output)
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

## Weighted code (2421)

- 2421 code is another BCD code.
- It is weighed code.
- 4 bit application code where the binary weights carry 2, 4, 2, 1 from left to right.
- For example, 6 is 1100 and 3 is 0011.

Decimal Number	Binary Number	2421 Code
0	0	0000
1	1	0001
2	10	0010
3	11	0011
4	100	0100
5	101	1011
6	110	1100
7	111	1101
8	1000	1110
9	1001	1111

**Table 1.6 : 2421 code**



## **Self Complementing Codes**

- Self-complementing binary codes are those whose members complement on themselves. For a binary code to become a self-complementing code, the following two conditions must be satisfied:
  - 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 (already stated procedure).
  - The sum of the binary number and its complement should be equal to decimal 9.

### **The Excess-3 (Xs-3) Code**

- It is a non-weighted code used to express code used to express decimal numbers.
- It is a self-complementary binary coded decimal (BCD) code and numerical system which has biased representation.

- An Xs-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 the decimal equivalent obtained in 1.
  - Convert the newly obtained decimal number back to binary number to get the desired Xs-3 equivalent.
- The codes 0000 and 1111 are not used for any digit.

<b>Binary numbers</b>	<b>Decimal equivalent</b>	<b>Decimal +3</b>	<b>Xs-3 equivalent</b>
0000	0	3	0011
0001	1	4	0100
0010	2	5	0101
0011	3	6	0110
0100	4	7	0111
0101	5	8	1000
0110	6	9	1001
0111	7	10	1010
1000	8	11	1011
1001	9	12	1100

**Table 1.7 : Excess-3 Code**

Example 1: Convert decimal number 23 to Excess-3 code.

Solution:

add 3 to both digit in the decimal number then convert into 4-bit binary number for result of each digit.

So,  $23+33=56=0101\ 0110$

Example 2: Convert decimal number 15.46 into Excess-3 code.

Solution:

$= 15.46+33.33=48.79=0100\ 1000.0111\ 1001$

Example 3: Convert  $(11110)_2$  to Excess-3

Solution:

$(11110)_2 = ((1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0))_{10} = (16 + 8 + 4 + 2 + 0)_{10} = (30)_{10}$

So,  $=30+33$

$= 63$

## Binary Coding Schemes

- Binary Coding schemes represent the data such as alphabets, digits 0–9, and symbols in a standard code.
- The binary coding schemes that are most commonly used are:
  - Extended Binary Coded Decimal Interchange Code (EBCDIC),
  - American Standard Code for Information Interchange (ASCII), and
  - Unicode

### EBCDIC

- The Extended Binary Coded Decimal Interchange Code (EBCDIC) uses 8 bits (4 bits for zone, 4 bits for digit) to represent a symbol in the data.
- EBCDIC allows  $2^8 = 256$  combinations of bits.
- 256 unique symbols are represented using EBCDIC code. It represents decimal numbers (0–9), lower case letters (a–z), uppercase letters (A–Z), Special characters, and Control characters (printable and non–printable, e.g., for cursor movement, printer vertical spacing, etc.)

- EBCDIC codes are mainly used in the mainframe computers.

## ASCII

- The American Standard Code for Information Interchange (ASCII) is widely used in computers of all types. ASCII codes are of two types—ASCII-7 and ASCII-8.
  - **ASCII-7** is a 7-bit standard ASCII code. In ASCII-7, the first 3 bits are the zone bits and the next 4 bits are for the digits. ASCII-7 allows  $2^7 = 128$  combinations. 128 unique symbols are represented using ASCII-7. ASCII-7 has been modified by IBM to ASCII-8.
  - **ASCII-8** is an extended version of ASCII-7. ASCII-8 is an 8-bit code having 4 bits for zone and 4 bits for the digit. ASCII-8 allows  $2^8 = 256$  combinations. ASCII-8 represents 256 unique symbols. ASCII is used widely to represent data in computers.

- The ASCII-8 code represents 256 symbols.
  - Codes 0 to 31 represent control characters (non-printable), because they are used for actions like, Carriage return (CR), Bell (BEL), etc.
  - Codes 48 to 57 stand for numeric 0–9.
  - Codes 65 to 90 stand for uppercase letters A–Z.
  - Codes 97 to 122 stand for lowercase letters a–z.
  - Codes 128 to 255 are the extended ASCII codes.

## Unicode

- Unicode is a universal character encoding standard for the representation of text which includes letters, numbers and symbols in multi-lingual environments. Unicode uses 32 bits to represent a symbol in the data.
- Unicode allows  $2^{32} = 4164895296$  (~ 4 billion) combinations.
- Unicode can uniquely represent any character or symbol present in any language like Chinese, Japanese, etc. In addition to the letters; mathematical and scientific symbols are also represented in Unicode codes.
- An advantage of Unicode is that it is compatible with the ASCII-8 codes. The first 256 codes in Unicode are identical to the ASCII-8 codes.



Symbol	ASCII	EBCDIC	Symbol	ASCII	EBCDIC
A	0100 0001	1100 0001	X	0101 1000	1110 0111
B	0100 0010	1100 0010	Y	0101 1001	1110 1000
C	0100 0011	1100 0011	Z	0101 1010	1110 1001
D	0100 0100	1100 0100	!	0010 0001	0101 1010
E	0100 0101	1100 0101	#	0010 0011	0111 1011
F	0100 0110	1100 0110	\$	0010 0100	0101 1011
G	0100 0111	1100 0111	%	0010 0101	0110 1100
H	0100 1000	1100 1000	&	0010 0110	0101 0000
I	0100 1001	1100 1001	(	0010 1000	0100 1101
J	0100 1010	1101 0001	)	0010 1001	0101 1101
K	0100 1011	1101 0010	*	0010 1010	0101 1100
L	0100 1100	1101 0011	+	0010 1011	0100 1110
M	0100 1101	1101 0100	0	0011 0000	1111 0000
N	0100 1110	1101 0101	1	0011 0001	1111 0001
O	0100 1111	1101 0110	2	0011 0010	1111 0010
P	0101 0000	1101 0111	3	0011 0011	1111 0011
Q	0101 0001	1101 1000	4	0011 0100	1111 0100
R	0101 0010	1101 1001	5	0011 0101	1111 0101
S	0101 0011	1110 0010	6	0011 0110	1111 0110
T	0101 0100	1110 0011	7	0011 0111	1111 0111
U	0101 0101	1110 0100	8	0011 1000	1111 1000
V	0101 0110	1110 0101	9	0011 1001	1111 1001
W	0101 0111	1110 0110			

Table 1.8 : ASCII and EBCDIC binary coding schemes

## 1.5 Error detection codes

- Binary information transmitted through some form of communication medium is subject to external noise that could change bits from 1 to 0, and vice versa.
- An error detection code is a binary code that detects digital errors during transmission.
- The most common error detection code used is the parity bit.
- A parity bit is an extra bit included with a binary message to make the total number of 1's either odd or even. Or A parity bit(s) is an extra bit that is added with original message to detect error in the message during data transmission.
- Two possible parity bits: **Even Parity** and **Odd Parity**

## Even Parity:

- One bit is attached to the information so that the total number of 1's is an even number .

Message	Parity
10110100	0
10010010	1

## Odd Parity:

- One bit is attached to the information so that the total number of 1's is an odd number .

Message	Parity
10110100	1
10010010	0

## **Parity Generator and Parity Checker**

- A Parity Generator is a combinational logic circuit that generates the parity bit in the transmitter (sender end).
- A circuit that checks the parity in the receiver end 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.

<b>Message</b>	<b>p(odd)</b>	<b>P(even)</b>
<b>x y z</b>		
0 0 0	1	0
0 0 1	0	1
0 1 0	0	1
0 1 1	1	0
1 0 0	0	1
1 0 1	1	0
1 1 0	1	0
1 1 1	0	1

- Parity generator and checker networks are logic circuits constructed with exclusive-OR functions.

$$P = \overline{x \oplus y \oplus z}$$

### **Parity Checker:**

Considers original message as well as parity bit

$$e = \overline{p \oplus x \oplus y \oplus z}$$

$e = 1 \Rightarrow$  No. of 1's in pxyz is even  $\Rightarrow$  Error in data

$e = 0 \Rightarrow$  No. of 1's in pxyz is odd  $\Rightarrow$  Data is error free

- This circuit consists of one exclusive-OR and one exclusive-NOR gate
- even-parity generators and checkers can be implemented with exclusive-OR functions. Odd-parity networks need an exclusive-NOR at the output to complement the function.

## Circuit diagram for parity generator and parity checker

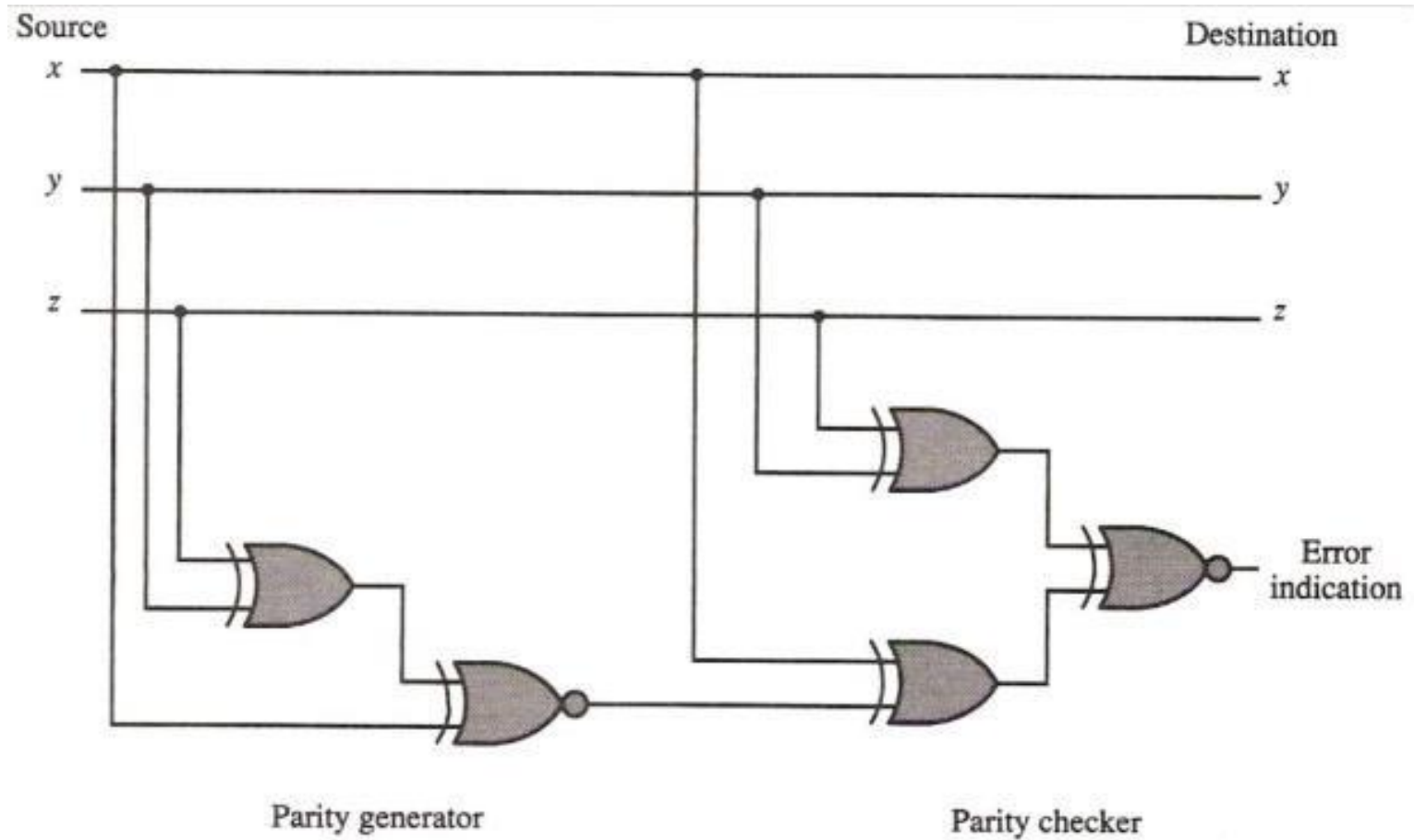


Figure: Error detection with odd parity bit

## Questions

- Find the signed magnitude of  $-(63)_{10}$  using 8-bit binary sequence?
- Derive the circuit for a 3-bit parity generator and a 4-bit parity checker using even parity bit.
- Differentiate between parity checker and parity generator. (T.U. 2066)
- Explain the error detection code with example. (T.U. 2068)
- Explain the subtraction algorithm with signed 2's compliment. (T.U. 2067)
- What is an error detection code? Explain with example. (T.U. 2070)
- Differentiate between fixed point representation and floating point representation. (T.U.2069)
- Write short notes on: (T.U. 2069)
  - Alphanumeric Representation
  - Parity Generator