DIGITAL ELECTRONICS CIRCUIT(BCA 103)

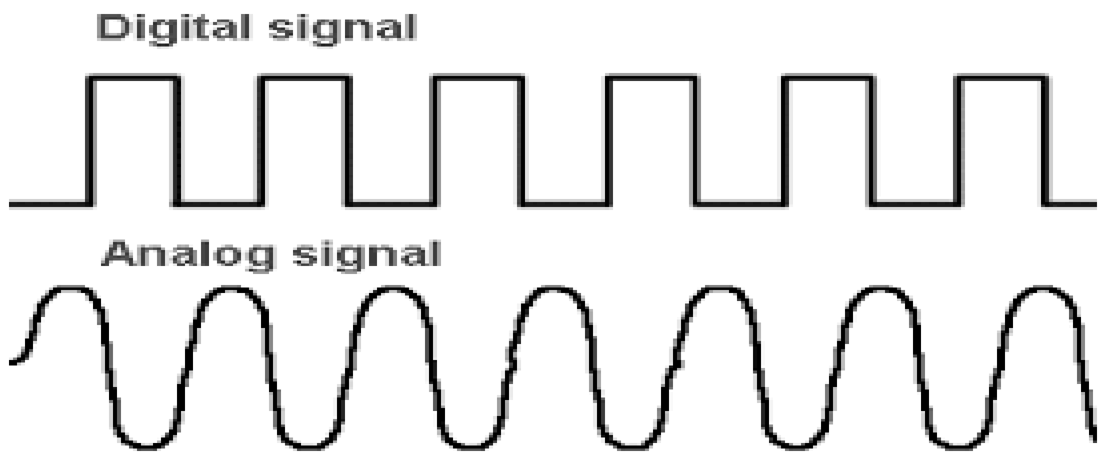
DEPARTMENT OF COMPUTER SCIENCE
PROGRAMME: BCA

CENTRAL UNIVERSITY OF ODISHA
KORAPUT

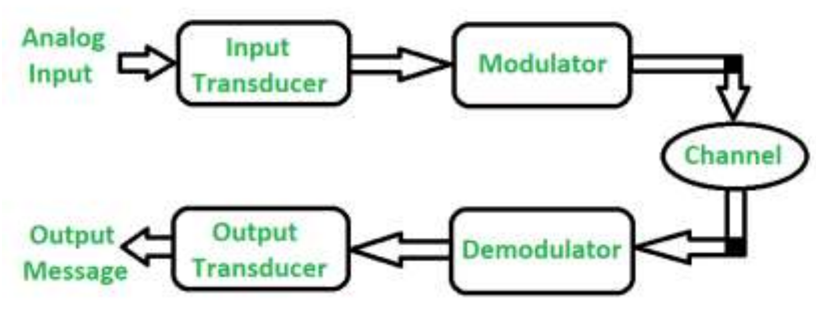
- “Digital Logic and Computer Design”, M Morris Mano.

- Signals carry information.
- The communication that occurs day to day life is in the form of signal.

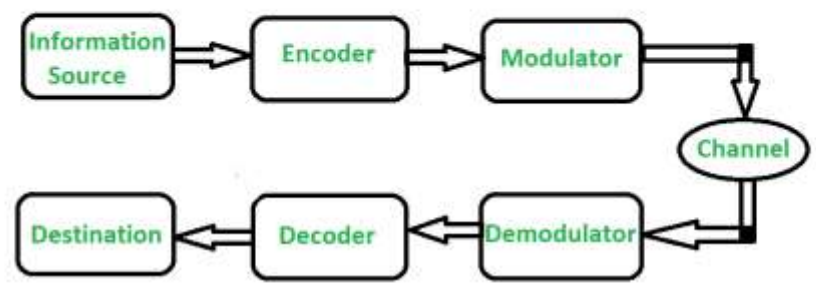
Analog Signal	Digital Signal
Ex: Temperature, FM radio signals, Human voice, natural sound	Ex: Computers, optical drives(CDs, DVDs)
It is a continuous signal that represents physical measurements.	These are discrete, time separated signals which are generated using digital modulation.
It is denoted by sine waves	It is denoted by square waves
It uses a continuous range of values that help you to represent information.	Digital signal uses discrete 0 and 1 to represent information.
The analog signal bandwidth is low	The digital signal bandwidth is high.
It is suited for audio and video transmission.	It is suited for Computing and digital electronics.
These signals are deteriorated by noise throughout transmission	Relatively a noise-immune system without deterioration during the transmission process


Digital signal
Analog signal

Analog Vs Digital



Analog Communication System



Digital Communication System

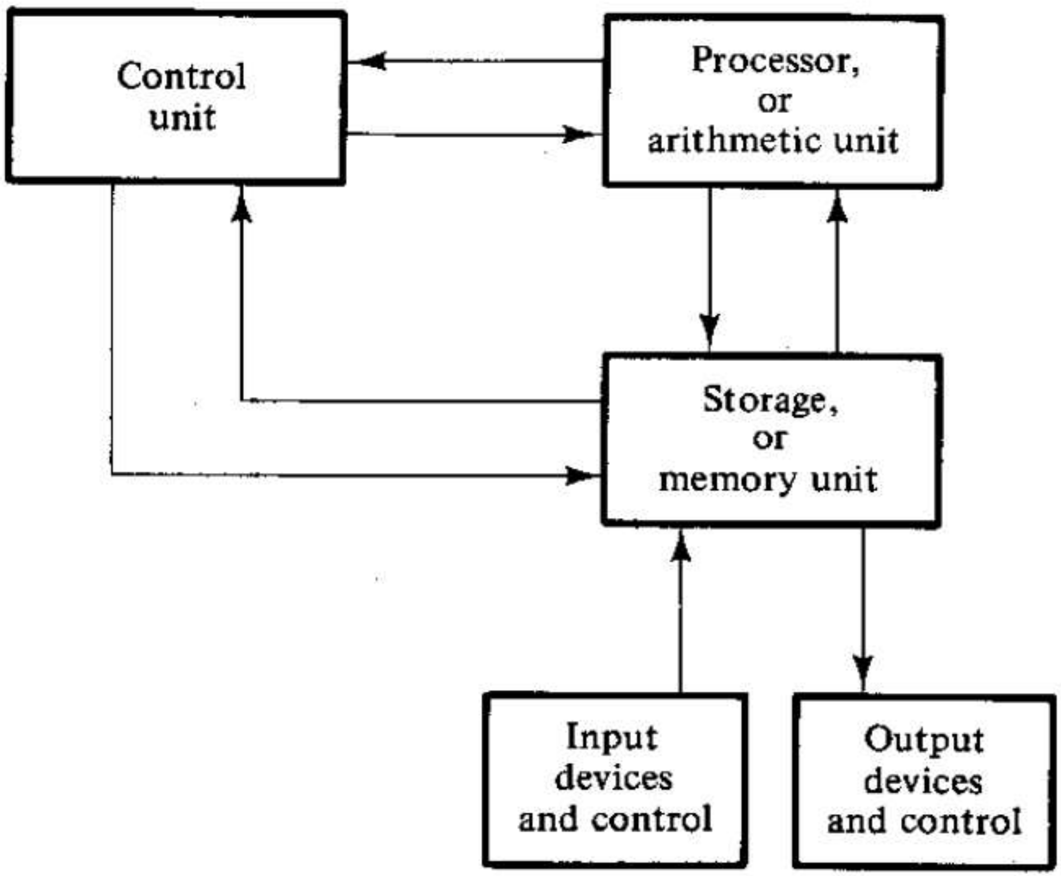
Analog Communication	Digital Communication
Analog signal is used for information transmission.	Digital signal is used for information transmission.
It uses analog signal whose amplitude varies continuously with time from 0 to 100.	It uses digital signal whose amplitude is of two levels either Low i.e., 0 or either High i.e., 1.
It gets affected by noise highly during transmission through communication channel.	It gets affected by noise less during transmission through communication channel.
Error Probability is high.	Error Probability is low.
Coding is not possible.	Different coding techniques can be used to detect and correct errors.
Separating out noise and signal in analog communication is not possible.	Separating out noise and signal in digital communication is possible.
This communication system is having complex hardware and less flexible.	This communication system is having less complex hardware and more flexible.
For multiplexing Frequency Division Multiplexing(FDM) is used.	For multiplexing Time Division Multiplexing(TDM) is used.

Power consumption is high.	Power consumption is low.
It is less portable.	Portability is high.
No privacy or privacy is less so not highly secured.	Privacy is high so it is highly secured.
Not assures an accurate data transmission.	It assures a more accurate data transmission.
Communication system is low cost.	Communication system is high cost.
It requires low bandwidth.	It requires high bandwidth.

Discrete Vs Continuous

- Discrete Data can only take certain values.
- Ex: Number of students in a class, the result of rolling a dice(i.e. 1, 2, 3, 4, 5, 6)
- **Discrete variables** are countable in a finite amount of time. The money in your bank account.
- Continuous Data can take any value (within a range)
- Ex: A person's height, Time in a race.
- **Continuous Variables** would (literally) take forever to count.

Block Diagram of Digital Computer



Positional Number Systems

- Digits represent different values depending on the position they occupy in the number.
- The value of each digit is determined by:
  - The digit itself
  - The position of the digit in the number
  - The base of the number system
- (**base/radix** = total number of digits in the number system)
- **Four Types:**
- **Decimal** *(Base(10), Digits-(0-9))*
- **Binary** *(Base(2), Digits-(0, 1))*
- **Octal** *(Base(8), Digits-(0-7))*
- **Hexadecimal** *(Base(16), Digits-(0-9, A, B, C, D, E, F))*

Decimal Number System

- It 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

- Ex: $(2586)_{10} = (2 \times 10^3) + (5 \times 10^2) + (8 \times 10^1) + (6 \times 10^0)$

$$= 2000 + 500 + 80 + 6$$

$$= 2586$$

Binary Number System

- It 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.

- Ex: $(10101)_2 = (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) \times (1 \times 2^0)$
  $= 16 + 0 + 4 + 0 + 1$
  $= (21)_{10}$
- In order to be specific about which number system we are referring to, it is a common practice to indicate the base as a subscript. Thus, we write:
- $(10101)_2 = (21)_{10}$

Bit

- Bit stands for **bi**nary digi**t**
- A bit in computer terminology means either a 0 or a 1
- A binary number consisting of *n* bits is called an n-bit number

Octal Number System

- It 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

- Ex: $(2057)_8 = (2 \times 8^3) + (0 \times 8^2) + (5 \times 8^1) + (7 \times 8^0)$

  $= 1024 + 0 + 40 + 7$

  $= (1071)_{10}$

Hexadecimal Number System

- It 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.

Ex: $(1AF)_{16} = (1 \times 16^2) + (A \times 16^1) + (F \times 16^0)$

$= 1 \times 256 + 10 \times 16 + 15 \times 1$

$= 256 + 160 + 15 = (431)_{10}$

Number System Conversion

- **Decimal to Non-Decimal**
  - Decimal to Binary
  - Decimal to Octal
  - Decimal to Hexadecimal
- **Non-Decimal to Decimal**
  - Binary to Decimal
  - Octal to Decimal
  - Hexadecimal to Decimal
- **Non-Decimal to Non-Decimal**

Decimal to Non-Decimal

- **Decimal to Binary**
- **Ex:** $(\ 28\ )_{10} = (\ ?\ )_2$ Ans. $(\ 28\ )_{10} = (\ 11100\ )_2$

	28	Remainder
2	14	0
2	7	0
2	3	1
2	1	1
2	0	1

Decimal to Non-Decimal

- **Decimal to Binary (Fraction)**
- **Ex:** $(.457)_{10} = (?)_2$ Ans. $(.457)_{10} = (.01110)_2$

0.457	X 2	
0.914	.914 X 2	0
1.828	.828 X 2	1
1.656	.656 X 2	1
1.312	.312 X 2	1
0.624		0

Decimal to Non-Decimal

- **Decimal to Octal**
- **Ex:** $(1628)_{10} = (?)_8$ Ans. $(1628)_{10} = (3134)_8$

8	1628	Remainder
8	203	4
8	25	3
8	3	1
	0	3

Decimal to Non-Decimal

- **Decimal to Octal (Fraction)**
- **Ex:** $(.257)_{10} = (?)_8$ Ans. $(.257)_{10} = (.20345)_8$

0.257	X 8
2.056	.056 X 8
0.448	.448 X 8
3.584	.584 X 8
4.672	.672 X 8
5.376	

2
0
3
4
5

Decimal to Non-Decimal

- **Decimal to Hexadecimal**
- **Ex:** $(1981)_{10} = (?)_{16}$ Ans. $(1981)_{10} = (7BD)_{16}$

	1981	Remainder
16	123	13 - D
16	7	11 - B
16	0	7

Decimal to Non-Decimal

- **Decimal to Hexadecimal (Fraction)**
- **Ex:** $(.974)_{10} = (?)_{16}$ Ans. $(.974)_{10} = (. F9581)_{16}$

0.974	X 16
15.584	.584 X 16
9.344	.344 X 16
5.504	.504 X 16
8.064	.064 X 16
1.024	

15 - F

9

5

8

1

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	16	8	4	2	1		1/2	1/4	1/8	1/16
							.5	.25	.125	.0625

Position	4	3	2	1	0	.	-1	-2	-3	-4
Position Value	8^4	8^3	8^2	8^1	8^0		8^{-1}	8^{-2}	8^{-3}	8^{-4}
Quantity	4096	512	64	8	1		1/8	1/64	1/512	1/4096
							.125	.015	.001	.0002

Position	4	3	2	1	0	.	-1	-2	-3	-4
Position Value	16^4	16^3	16^2	16^1	16^0		16^{-1}	16^{-2}	16^{-3}	16^{-4}
Quantity	65536	4096	256	16	1		1/16	1/256	1/4096	1/65536
							.0625	.0039	.0002	

Non-Decimal to Decimal

- **Binary to Decimal**
- **Ex:** $(11011)_2 = (\ ? \ )_{10}$ $\quad (11011)_2 = (\ 27 \ )_{10}$

11011 . 1010

$$(11011)_2 = 1 \text{ x } 2^4 + 1 \text{ x } 2^3 + 0 \text{ x } 2^2 + 1 \text{ x } 2^1 + 1 \text{ x } 2^0$$
$$= 16 + 8 + 0 + 2 + 1 = (27)_{10}$$

Ex: $(.1010)_2 = (\ ? \ )_{10}$ $\quad (1010)_2 = (\ 0.625 \ )_{10}$

$$(.1010)_2 = 1 \text{ x } 2^{-1} + 0 \text{ x } 2^{-2} + 1 \text{ x } 2^{-3} + 0 \text{ x } 2^{-4}$$
$$= 1 \text{ x } 0.5 + 0 + 1 \text{ x } 0.125 + 0 = 0.5 + 0.125 = (0.625)_{10}$$

- Ex: $(110.101)_2 = 1 \text{ x } 2^2 + 1 \text{ x } 2^1 + 0 \text{ x } 2^0 + 1 \text{ x } 2^{-1} + 0 \text{ x } 2^{-2} + 1 \text{ x } 2^{-3}$
  $= 4 + 2 + 0 + 0.5 + 0 + 0.125$
  $= (6.625)_{10}$

Non-Decimal to Decimal

- **Octal to Decimal**
- **Ex:** $(7034)_8 = (\ ?\ )_{10}$ $(7034)_8 = (\ 3609\ )_{10}$

7034 . 251

$(7034)_8 = 7 \times 8^3 + 0 \times 8^2 + 3 \times 8^1 + 4 \times 8^0$
$= 7 \times 512 + 0 + 24 + 1 = 3584 + 24 + 1 = (\ 3609\ )_{10}$

Ex: $(.251)_8 = (\ ?\ )_{10}$ $(.251)_8 = (\ 0.326\ )_{10}$

$(.251)_8 = 2 \times 8^{-1} + 5 \times 8^{-2} + 1 \times 8^{-3}$
$= 2 \times 0.125 + 5 \times 0.015 + 1 \times 0.001 = 0.250 + 0.075 + 0.001$
$= (0.326)_{10}$

- **Ex:** $(16.47)_8 = 1 \times 8^1 + 6 \times 8^0 + 4 \times 8^{-1} + 7 \times 8^{-2}$
$= 8 + 6 + 4 \times 0.125 + 7 \times 0.015 = 14 + 0.600 + 0.105$
$= (14.705)_{10}$

Non-Decimal to Decimal

- **Hexadecimal to Decimal**
- **Ex:** $(2BCA)_{16} = (\ ?\ )_{10}$ $(2BCA)_{16} = (\ 11210\ )_{10}$

AB . 4C

$(2BCA)_{16} = 2 \times 16^3 + B \times 16^2 + C \times 16^1 + A \times 16^0$
$= 2 \times 4096 + 11 \times 256 + 12 \times 16 + 10 \times 1$
$= 8192 + 2816 + 192 + 10 = (11210)_{10}$

Ex: $(.ED8)_{16} = (\ ?\ )_{10}$ $(ED8)_{16} = (\ 0.9273\ )_{10}$

$(.ED8)_{16} = E \times 16^{-1} + D \times 16^{-2} + 8 \times 16^{-3}$
$= 14 \times 0.0625 + 13 \times 0.0039 + 8 \times 0.0002$
$= 0.8750 + 0.0507 + 0.0016 = (0.9273)_{10}$

- Ex: $(AB.4C)_{16} = A \times 16^1 + B \times 16^0 + 4 \times 16^{-1} + C \times 16^{-2}$
$= 10 \times 16 + 11 \times 1 + 4 \times 0.0625 + 12 \times 0.0039$
$= 160 + 11 + 0.2500 + 0.0468$
$= 171 + 0.2968 = (171.2968)_{10}$

Non-Decimal to Non-Decimal

- **Octal to Binary**
- **Ex:** $(562)_8 = (?)_2$

$(5)_8 = (101)_2$ $(6)_8 = (110)_2$ $(2)_8 = (010)_2$

Hence $(562)_8 = (101110010)_2$

Ex : $(.174)_8 = (?)_2$

$(1)_8 = (001)_2$ $(7)_8 = (111)_2$ $(4)_8 = (100)_2$

Hence $(.174)_8 = (.001111100)_2$

Ex : $(143.62)_8 = (?)_2$

$(1)_8 = (001)_2$ $(4)_8 = (100)_2$ $(3)_8 = (011)_2$ $(6)_8 = (110)_2$ $(2)_8 = (010)_2$

Hence $(143.62)_8 = (001100011.110010)_2$

Non-Decimal to Non-Decimal

- **Hexadecimal to Binary**
- **Ex:** $(A9C)_{16} = (?)_2$

$(A)_{16} = (1010)_2$ $(9)_{16} = (1001)_2$ $(C)_{16} = (1100)_2$

Hence $(A9C)_{16} = (101010011100)_2$

Ex : $(.E57)_{16} = (?)_2$

$(E)_{16} = (1110)_2$ $(5)_{16} = (0101)_2$ $(7)_{16} = (0111)_2$

Hence $(.174)_{16} = (.111001010111)_2$

Ex : $(BD.F8)_{16} = (?)_2$

$(B)_{16} = (1011)_2$ $(D)_{16} = (1101)_2$ $(F)_{16} = (1111)_2$ $(8)_{16} = (0100)_2$

Hence $(BD.F8)_{16} = (10111101.11110100)_2$

Non-Decimal to Non-Decimal

- **Binary to Octal:**
- **Ex:** $(10110)_2 = (\ ?\ )_8$

$(10110)_2 = (\ 26\ )_8$

$(10\ 110)_2 = (\ 010\ 110)_2 = (26)_8$

Ex: $(\ .\ 1010010)_2 = (\ ?\ )_8$

$(.\ 1010010)_2 = (\ .122\ )_8$

$(1\ 010\ 010)_2 = (001\ 010\ 010)_2 = (.122)_8$

Ex: $(\ 1100.\ 1011011)_2 = (\ ?\ )_8$

$(1100.\ 1011011)_2 = (14.133\ )_8$

$(1100.\ 1011011)_2 = (001100.\ 001011011)_2 = (14.133)_8$

Non-Decimal to Non-Decimal

- **Binary to Hexadecimal:**
- **Ex:** $(10110)_2 = (?)_{16}$ $(10110)_2 = (16)_{16}$

$(10110)_2 = (0001\ 0110)_2 = (16)_{16}$

Ex: $(.\ 1010010)_2 = (?)_{16}$ $(.\ 1010010)_2 = (.52)_{16}$

$(1010010)_2 = (0101\ 0010)_2 = (.52)_{16}$

Ex: $(10100.\ 1011011)_2 = (?)_{16}$ $(10100.\ 1011011)_2 = (14.5B)_{16}$

$(10100.\ 1011011)_2 = (00010100.\ 01011011)_2 = (14.5B)_{16}$

Numbers with Different Bases

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Binary Arithmetic

- Binary Addition:

Rule for binary addition is as follows:

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 0 plus a carry of 1 to next higher column

Ex: Add binary numbers $(101101)_2$ and $(100111)_2$

Ans. $(1010100)_2$

Carry	1		1	1	1	1	
Augend		1	0	1	1	0	1
Addend		1	0	0	1	1	1
Sum	1	0	1	0	1	0	0

Carry		1	1	1	1	1	
Augend		1	0	0	1	1	1
Addend			1	1	0	1	1
Sum	1	0	0	0	0	1	0

Carry		1	1	1	1		
Augend			1	1	1	1	1
Addend				1	1	1	0
Sum		1	0	1	1	0	1

Carry		1	1	1	1		
Augend			1	1	0	1	1
Addend				1	1	1	0
Sum		1	0	1	0	0	1

- Binary Subtraction:
- **Rule for binary subtraction is as follows:**
- 0 - 0 = 0
- 0 - 1 = 1 with a borrow from the next column
- 1 - 0 = 1
- 1 - 1 = 0
- Ex: Subtract $(01110)_2$ from $(10101)_2$

Borrow			**0**	**1**	**0**	**1**	
Minuend			1	0	1	0	1
Subtrahend			0	1	1	1	0
Difference			0	0	1	1	1

Borrow				**0**	**0**	**1**	
Minuend		1	0	1	1	0	1
Subtrahend		1	0	0	1	1	1
Difference		0	0	0	1	1	0

Borrow				**0**	**0**	**1**	**1**
Minuend		1	1	1	1	0	0
Subtrahend		1	0	0	1	1	1
Difference		0	1	0	1	0	1

Multiplication

Multiplicand			1	0	1	1
Multiplier		X		1	0	1
			1	0	1	1
		0	0	0	0	
	1	0	1	1		
Product	1	1	0	1	1	1

Division

								Quotient
101		**1**	**1**	**0**	**1**	**0**		**101**
		1	**0**	**1**				
				1	1			
				0	0			
				1	**1**	**0**		
				1	0	1		
						1		**Remainder**

COMPLEMENTS

- Complements are used in digital computers for simplifying the subtraction operation and for logical manipulation.
- Simplifying operations leads to simpler, less expensive circuits to implement the operations.
- There are two types of complements for each base-r system:

Types	Also known as	Base (r)	2	10	8	16
Radix Complement	r's Complement		**2's**	**10's**	**8's**	**16's**
Diminished Radix Complement	(r-1)'s Complement		**1's**	**9's**	**7's**	**15's**

Diminished Radix Complement((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**
- For Binary numbers **r = 2** and **r – 1 = 1**

- **Ex:** The **9's** complement of 546700 is **------**?
- Ans: N = 946700 r = 10 n = 6

$$(10^6 - 1) – 546700 = (1000000 – 1) – 546700$$

$$= 999999 – 546700 = 453299$$

The 9's complement of a decimal number is obtained by subtracting each digit from 9.

Ex: The 9's complement of 012398 is : 999999 – 012398 = 987601

Diminished Radix Complement((r-1)’s Complement)

- Given a number **‘N’** in base **‘r’** having **‘n’** digits, the **(r-1)’s** complement of **‘N’** is defined as : $\mathbf{(r^n-1)-N}$.

- For Binary numbers **r = 2** and **r – 1 = 1**
- **Ex:** The **1’s** complement of 1011000 is ------?
- Ans: N = 1011000 r = 2 n = 7

$$(2^7 - 1) - 1011000 = (10000000 - 1) - 1011000$$
$$= 1111111 - 1011000 = 0100111$$

The 1’s complement of a binary number is obtained by changing 1’s to 0’s and 0’s to 1’s.

Ex: The 1’s complement of 0101101 is : 1010010

The (r – 1)’s complement of Octal and Hexadecimal numbers is obtained by subtracting each digit from 7 or F (or Decimal 15), respectively.

Radix Complement(r's Complement)

- The **r's** complement of an **'n'** digit number **'N'** in base **'r'** is defined as $\mathbf{r^n - N}$ for $\mathbf{N \neq 0}$ and **0** for $\mathbf{N = 0}$.
- Comparison with **(r-1)'s** complement: **r's** complement is obtained by adding **1** to the **(r – 1)'s** complement since

$$\mathbf{r^n - N = [\,(r^n - 1) - N] + 1}$$

Ex: 10's complement of 2389 is ------?

Ans: 10's complement of 2389 is = (9's complement of 2389) +1

= (9999 – 2389) + 1 = 7610 + 1 = 7611

Ex: 2's complement of 101100 is = (1's complement of 101100) +1

= 010011 + 1 = 010100

- **Second method:** Leaving all least significant 0’s unchanged, subtracting first non-zero least significant digit from 10, and subtracting all higher significant digits from 9.
- **Ex:** 10’s complement of 012398 -----?
- **Ans:** No least significant 0’s.

Subtracting first non-zero least significant digit (i.e. 8) from 10 = 10 – 8 =2

Subtracting all higher significant digits from 9 = 99999 – 01239 = 98760

Hence, 10’s complement of 012398 is 987602.

Ex: 10’s complement of 246700 -----?

Ans: Leaving **2** least significant 0’s.

Subtracting first non-zero least significant digit (i.e. 7) from 10 = 10 – 7 =3

Subtracting all higher significant digits from 9 = 999 – 246 = 753

Hence, 10’s complement of 246700 is 753300.

- **Second method:** Leaving all least significant 0’s and the first 1 unchanged, and replacing 1’s with 0’s and 0’s with 1’s in all other higher significant digits.

Ex: 2’s complement of 1101100 -----?

Ans: Leaving 2 least significant 0’s and the first 1 unchanged. (i.e. 1101100)

replacing 1’s with 0’s and 0’s with 1’s in all other higher significant digits . i.e. 1101 as 0010

Hence, 2’s complement of 1101100 is 0010100.

Ex: 2’s complement of 0110111 -----?

Ans: No least significant 0’s and leaving the first 1 unchanged. (i.e. 0110111)

replacing 1’s with 0’s and 0’s with 1’s in all other higher significant digits . i.e. 011011 as 100100

Hence, 2’s complement of 0110111 is 1001001.

- The complement of the complement restores the number to its original value.
- The **r's** complement of **N** is $\mathbf{r^n - N}$.
- The complement of the complement is $\mathbf{r^n - (r^n - N) = N}$, giving back the original number.

Subtraction With r's Complements

- When subtraction is implemented with digital hardware, borrow method using pen, pencil is found to be less efficient than the method that use complements.
- **Rules :** The subtraction of two n-digit unsigned numbers **M – N** in base **'r'**can be done as follows:
- 1. **Add** the minuend **M** to the **r's complement** of the subtrahend **N**. This performs $M + (r^n - N) = M - N + r^n$
- 2. **If M ≥ N**, the sum will produce **an end carry, r^n, which is discarded**; what is left is the result M – N.
- 3. **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, **take the r's complement of the sum and place a negative sign in front**.

- Ex: Using 10's complement subtract 72532 – 3250?
- Ans: M = 72532   N = 3250
- Step 1: Add the M(72532) to the 10's complement of the N(3250).
  - 9's complement of 03250 = 9999 – 03250 = 96749
  - 10's complement of 03250 = 9's complement of 03250 + 1

    = 96749 + 1 = 96750

- M + N = 72532 + 96750 =169282

Step 2: Since M > N and sum produces an end carry, so it should be discarded. i.e. answer will be 69282.

Note: M has 5 digits and N has only 4 digits. Both numbers must have the same number of digits. So we can write N = 3250 as N = 03250.

Note: The occurrence of the end carry signifies that $M \geq N$ and the result is positive.

- Ex: Using 10's complement subtract 3250 – 72532?
- Ans: M = 3250 N= 72532
- Step 1: Add the M(3250) to the 10's complement of the N(72532).
  - 9's complement of 72532 = 99999 – 72532 = 27467
  - 10's complement of 72532 = 9's complement of 72532 + 1

    = 27467 + 1 = 27468

- M + N = 03250 + 27468 = 30718

Step 2: Since M < N and sum produces no end carry, answer will be :

- (10's complement of 30718) .

- 9's complement of 30718 = 99999 – 30718 = 69281
- 10's complement of 30718 = 9's complement of 30718 + 1

  = 69281 + 1 = 69282

Hence, 10's complement subtract 3250 – 72532 = - (69282)

- Ex: Using 2’s complement subtract 1010100 – 1000011?
- Ans: M = 1010100 N = 1000011
- Step 1: Add the M(1010100) to the 2’s complement of the N(1000011).
  - 1’s complement of 1000011 = 0111100
  - 2’s complement of 1000011 = 1’s complement of 1000011 + 1
    = 0111100 + 1 = 0111101

- M + N = 1010100 + 0111101 =10010001

Step 2: Since M > N and sum produces an end carry, so it should be discarded. i.e. answer will be 0010001.

- Ex: Using 2’s complement subtract 1000011 – 1010100?
- Ans: M = 1000011 N= 1010100
- Step 1: Add the M(1000011) to the 2’s complement of the N(1010100).
  - 1’s complement of 1010100 = 0101011
  - 2’s complement of 1010100 = 1’s complement of 1010100 + 1

    = 0101011 + 1 = 0101100

- M + N = 1000011 + 0101100 = 1101111

Step 2: Since M < N and sum produces no end carry, answer will be :

- (2’s complement of 1101111) .

- 1’s complement of 1101111 = 0010000
- 2’s complement of 1101111 = 1’s complement of 1101111 + 1

  = 0010000 + 1 = 0010001

Hence, 2’s complement subtract 1000011 – 1010100 = - (0010001)

Subtraction With (r – 1)'s Complements

- The (r - 1)'s complement is one less than the r's complement.
- Because of this, the result of adding the minuend to the complement of the subtrahend produces a sum that is one less than the correct difference when an end carry occurs.
- Removing the end carry and adding 1 to the sum is referred to as an ***end-around carry***.
- **Rules :** The subtraction of two n-digit unsigned numbers **M – N** in base **'r'**can be done as follows:
- 1. **Add** the minuend **M** to the **(r – 1)'s complement** of the subtrahend **N**.
- 2. **If M ≥ N**, the sum will produce **an end carry, r^n, which is added to it**; what is the result of M – N.
- 3. **If M – N** , the sum does not produce an end carry and to obtain the answer, **take the (r – 1)'s complement of the sum and place a negative sign in front**.

Subtraction With (r – 1)'s Complements

- Ex: Using 9's complement subtract 72532 – 3250?
- Ans: M = 72532 N = 3250
- Step 1: Add the M(72532) to the 9's complement of the N(3250).
  - 9's complement of 03250 = 9999 – 03250 = 96749

  - M + N = 72532 + 96749 =169281

Step 2: Since M > N and sum produces an end carry, so it should be added. i.e. answer will be 69281 + 1 = 69282.

Note: M has 5 digits and N has only 4 digits. Both numbers must have the same number of digits. So we can write N = 3250 as N = 03250.

Note: The occurrence of the end carry signifies that M ≥ N and the result is positive.

Subtraction With (r – 1)'s Complements

- Ex: Using 9's complement subtract 3250 – 72532?
- Ans: M = 3250 N= 72532
- Step 1: Add the M(3250) to the 9's complement of the N(72532).
  - 9's complement of 72532 = 99999 – 72532 = 27467
  - M + N = 03250 + 27467 = 30717

Step 2: Since M < N and sum produces no end carry, answer will be :

- (9's complement of 30717) .

- 9's complement of 30717 = 99999 – 30717 = 69282

Hence, 9's complement subtract 3250 – 72532 = - (69282)

Subtraction With (r – 1)'s Complements

- Ex: Using 1's complement subtract 1010100 – 1000011?
- Ans: M = 1010100   N = 1000011
- Step 1: Add the M(1010100) to the 1's complement of the N(1000011).
  - 1's complement of 1000011 = 0111100
  - M + N = 1010100 + 0111100 =10010000

Step 2: Since M > N and sum produces an end carry, so it should be added. i.e. answer will be 0010000 + 1 = 0010001

Subtraction With (r – 1)'s Complements

- Ex: Using 1's complement subtract 1000011 – 1010100?
- Ans: M = 1000011 N= 1010100
- Step 1: Add the M(1000011) to the 1's complement of the N(1010100).
  - 1's complement of 1010100 = 0101011

  - M + N = 1000011 + 0101011 = 1101110

Step 2: Since M < N and sum produces no end carry, answer will be :

- (1's complement of 1101110) .

- 1's complement of 1101110 = 0010001

Hence, 1's complement subtract 1000011 – 1010100 = - (0010001)

Signed Binary Numbers

- Positive integers (including zero) can be represented as unsigned numbers.
- However, to represent negative integers, we need a notation for negative values.
- Because of hardware limitations, computers must represent everything with binary digits.
- It is customary to represent the sign with a bit placed in the leftmost position of the number.
- The convention is to make the sign bit 0 for positive and 1 for negative.
- If the binary number is signed, then the leftmost bit represents the sign and the rest of the bits represent the number.
- If the binary number is assumed to be unsigned, then the leftmost bit is the most significant bit of the number.

- Ex: **01001** can be considered as **9** (**unsigned binary**) or as **+9** (**signed binary**) because the leftmost bit is **0**.
- Ex: **11001** represents the binary equivalent of **25** when considered as an **unsigned** number and the binary equivalent of **- 9** when considered as a **signed** number.
- This is because the **1** that is in the leftmost position designates a negative and the other four bits represent binary 9.

0	**1**	**0**	**0**	**1**	**Unsigned**	**9**
0	**1**	**0**	**0**	**1**	**Signed**	**+9**
1	**1**	**0**	**0**	**1**	**Unsigned**	**25**
1	**1**	**0**	**0**	**1**	**Signed**	**-9**

Notation For Signed Number

- **signed-magnitude convention:** In this notation, the number consists of a magnitude and a symbol (+ or -) or a bit (0 or 1) indicating the sign.
- The signed-magnitude system negates a number by changing its sign.
- This is the representation of signed numbers used in ordinary arithmetic.
- **signed- complement system:** Arithmetic operations are implemented in a computer .
- For representing negative numbers.
- In this system, a negative number is indicated by its complement.
- Since positive numbers always start with 0 (plus) in the leftmost position, the complement will always start with a 1, indicating a negative number. The signed-complement system can use either the 1’s or the 2’s complement, but the 2’s complement is the most common.

- Ex: 9 represented in binary with eight bits.
- **+ 9**

0	0	0	0	1	0	0	1
Sign bit							

- Although there is only one way to represent +9, there are three different ways to represent -9 with eight bits:
-

Notation	Representation	?
signed-magnitude representation	10001001	Changing only sign bit in the left most position of +9
signed-1's-complement representation	11110110	1's complement of +9
signed-2's-complement representation	11110111	2's complement of +9

- The positive numbers in all three representations are identical and have 0 in the leftmost position.
- The signed-2's-complement system has only one representation for 0, which is always positive.
- The other two systems have either a positive 0 or a negative 0, something not encountered in ordinary arithmetic.
- All negative numbers have a 1 in the leftmost bit position; that is the way we distinguish them from the positive numbers.

- With four bits, we can represent 16 binary numbers.
- In the signed-magnitude and the 1's-complement representations, there are eight positive numbers and eight negative numbers, including two zeros.
- In the 2's-complement representation, there are eight positive numbers, including one zero, and eight negative numbers.

Signed Binary Numbers

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

Arithmetic Addition (Signed)

- The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmetic.
- If the **signs are the same**, we add the two magnitudes and give **the sum the common sign**.
- If the **signs are different**, we subtract the smaller magnitude from the larger and give **the difference the sign of the larger magnitude**.

- **Example:** (+25) + (-37) = -(37 - 25) = -12
- This is a process that requires a comparison of the signs and magnitudes and then performing either addition or subtraction.
- The same procedure applies to binary numbers in **signed-magnitude** representation.

- The addition of two signed binary numbers with negative numbers represented in signed-2's-complement form is obtained from the addition of the two numbers, including their sign bits. A carry out of the sign-bit position is discarded.
- **Note:** Negative numbers must be initially in 2's-complement form and that if the sum obtained after the addition is negative, it is in 2's-complement form.
- Any carry out of the sign-bit position is discarded, and negative results are automatically in 2's-complement form.

		Binary Representation
Augend	+6	00000110
Addend	+13	00001101
Result	+19	00010011

		Binary Representation (8-bits)	Signed 1's Complement	Signed 2's Complement	1's Complement	2's Complement (Normal Form)
Augend	+ 6	00000110		00000110		
Addend	- 13	00001101	11110010	11110011		
Result	- 7			11111001	00000110	00000111

		Binary Representation (8-bits)	Signed 1's Complement	Signed 2's Complement		
Augend	- 6	00000110	11111001	11111010		
Addend	+13	00001101		00001101		
Result	+ 7			**1**00000111		

		Binary Representation (8-bits)	Signed 1's Complement	Signed 2's Complement	1's Complement	2's Complement (Normal Form)
Augend	- 6	00000110	11111001	11111010		
Addend	- 13	00001101	11110010	11110011		
Result	- 19			**1**11101101	00010010	00010011

- In order to obtain a correct answer, we must ensure that the result has a sufficient number of bits to accommodate the sum. If we start with two n-bit numbers and the sum occupies $n + 1$ bits, we say that an overflow occurs.
- Overflow is a problem in computers because the number of bits that hold a number is finite, and a result that exceeds the finite value by 1 cannot be accommodated.
- The complement form of representing negative numbers is unfamiliar to those used to the signed-magnitude system.
- To determine the value of a negative number in signed-2's complement, it is necessary to convert the number to a positive number to place it in a more familiar form.

Arithmetic Subtraction (Signed)

- Subtraction of two signed binary numbers when negative numbers are in 2's-complement form is simple and can be stated as follows:
- - Take the 2's complement of the subtrahend (including the sign bit) and add it to the minuend (including the sign bit).
- - A carry out of the sign-bit position is discarded.
- This procedure is adopted because a subtraction operation can be changed to an addition operation if the sign of the subtrahend is changed, as is demonstrated by the following relationship:

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

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

- Changing a positive number to a negative number is easily done by taking the 2’s complement of the positive number.
- The reverse is also true, because the complement of a negative number in complement form produces the equivalent positive number.

		Binary Representation (8-bits)	Signed 1’s Complement	Signed 2’s Complement	1’s Complement	2’s Complement (Normal Form)
Minuend	- 6	00000110	11111001	11111010		11111010
Subtrahend	- 13	00001101	11110010	11110011		00001101
Difference	+7			00000111		
					Sum	100000111

- It is worth noting that binary numbers in the signed-complement system are added and subtracted by the same basic addition and subtraction rules as unsigned numbers.
- Therefore, **computers need only one common hardware circuit to handle both types of arithmetic .**
- This consideration has resulted in the signed-complement system being used in virtually all arithmetic units of computer systems.
- The user or programmer must interpret the results of such addition or subtraction differently, depending on whether it is assumed that the numbers are signed or unsigned.

Binary Code

- Digital systems represent and manipulate not only binary numbers, but also many other discrete elements of information.
- Any discrete element of information that is distinct among a group of quantities can be represented with a binary code (i.e., a pattern of 0's and 1's).
- An n-bit binary code is a group of n bits that assumes 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.
- The bit combination of an n-bit code is determined from the count in binary from 0 to $2^n - 1$.
- Each element must be assigned a unique binary bit combination, and no two elements can have the same value; otherwise, the code assignment will be ambiguous.
- Although the *minimum* number of bits required to code 2^n distinct quantities is n, there is no *maximum* number of bits that may be used for a binary code.

Binary Coded Decimal Code (BCD Code)

- Since the computer can accept only binary values, we must represent the decimal digits by means of a code that contains 1's and 0's.
- It is also possible to perform the arithmetic operations directly on decimal numbers when they are stored in the computer in coded form.
- A binary code that distinguishes among 10 elements must contain at least four bits, but 6 out of the 16 possible combinations remain unassigned.
- Different binary codes can be obtained by arranging four bits into 10 distinct combinations.
- This scheme is called *binary-coded decimal* and is commonly referred to as BCD.

Binary Coded Decimal Code (BCD Code)

- A number with ***k*** decimal digits will require **4*k*** bits in BCD.
- Ex: $(179)_{10} = (0001\ \ 0111\ \ 1001)_{BCD} = (10110011)_2$
- **Each group of 4 bits representing one decimal digit.**
- A decimal number in BCD is the same as its equivalent binary number only when the number is between 0 and 9.
- A 'BCD' number greater than 10 looks different from its equivalent binary number, even though both contain 1's and 0's.
- Moreover, **the binary combinations 1010 through 1111 are not used and have no meaning in BCD.**

Binary-Coded Decimal (BCD)

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

Binary Coded Decimal Code (BCD Code)

- Ex: $(179)_{10} = (0001\ \ 0111\ \ 1001)_{BCD} = (10110011)_2$
- The representation of a BCD number needs more bits than its equivalent binary value.
- However, there is an advantage in the use of decimal numbers, because computer input and output data are generated by people who use the decimal system.
- It is important to realize that BCD numbers are decimal numbers and not binary numbers, although they use bits in their representation.
- The only difference between a decimal number and BCD is that decimals are written with the symbols 0, 1, 2, … , 9 and BCD numbers use the binary code 0000, 0001, 0010, … , 1001.

BCD Addition

- When the binary sum is equal to or less than 1001 (without a carry), the corresponding BCD digit is correct.
- However, when the binary sum is greater than or equal to 1010, the result is an invalid BCD digit.
- The addition of 6 = $(0110)_2$ to the binary sum converts it to the correct digit and also produces a carry as required.
- This is because a carry in the most significant bit position of the binary sum and a decimal carry differ by 16 - 10 = 6.
-

4		0	1	0	0
5	+	0	1	0	1
9		1	0	0	1

4		0	1	0	0
8	+	1	0	0	0
12		1	1	0	0
	+	0	1	1	0
	1	0	0	1	0

BCD Addition

8			1	0	0	0
9	+		1	0	0	1
17	+	1	0	0	0	1
			0	1	1	0
		1	0	1	1	1

				1						1					
184	0	0	0	1			1	0	0	0		0	1	0	0
+ 576	0	1	0	1			0	1	1	1		0	1	1	0
= 760															
Binary Sum	0	1	1	1		1	0	0	0	0		1	0	1	0
Add 6							0	1	1	0		0	1	1	0
BCD Sum	0	1	1	1		1	0	1	1	0		0	0	0	0

Decimal Arithmetic

- The representation of signed decimal numbers in BCD is similar to the representation of signed numbers in binary.
- We can use either the familiar signed-magnitude system or the signed-complement system.
- The sign of a decimal number is usually represented with four bits to conform to the four-bit code of the decimal digits.
- A plus with four 0's (0000) and a minus with the BCD equivalent of 9, which is 1001 .
- The signed-magnitude system is seldom used in computers.
- The signed-complement system can be either the 9's or the 10's complement, but the 10's complement is the one most often used.
- Addition is done by summing all digits, including the sign digit, and discarding the end carry.
- This operation assumes that all negative numbers are in 10's complement form.

- Ex: addition (+375) + (-240) = +135, done in the signed-complement system :

Ans:

Decimal	+	375
	-	240
		135

BCD	0	375
	9	760
	0	1135

10’s complement of (240) is = 9’s complement of (240) + 1 = (999 – 240) + 1 = 759 + 1 = 760

Hence, (+375) + (-240) = +135, discarding end carry.

The decimal numbers inside the computer, including the sign digits, must be in BCD.

The subtraction of decimal numbers, either unsigned or in the signed-10’s complement system, is the same as in the binary case:

Take the 10’s complement of the subtrahend and add it to the minuend.

Other Decimal Codes

- In a weighted code, each bit position is assigned a weighting factor in such a way that each digit can be evaluated by adding the weights of all the 1's in the coded combination.
- Ex: BCD and the 2421 code.
- The **BCD code** has weights of 8, 4, 2, and 1, which correspond to the power-of-two values of each bit.

  Ex: 0110 = **8** x 0 + **4** x 1 + **2** x 1 + **1** x 0 = $(6)_{10}$
- **2421 code :**

  Ex: 1101 = **2** x 1 + **4** x 1 + **2** x 0 + **1** x 1 =$(7)_{10}$
- Some digits can be coded in two possible ways in the 2421 code. For instance, decimal 4 can be assigned to bit combination 0100 or 1010, since both combinations add up to a total weight of 4.

Four Different Binary Codes for the Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
Unused bit combi- nations	1010	0101	0000	0001
	1011	0110	0001	0010
	1100	0111	0010	0011
	1101	1000	1101	1100
	1110	1001	1110	1101
	1111	1010	1111	1110

- **Excess-3** is an unweighted code in which each coded combination is obtained from the corresponding binary value plus 3.
- This code has been used in some older computers because of its self-complementing property.
- But, the BCD code is not self-complementing.
- The **2421 and the excess-3 codes** are examples of self-complementing codes.
- Such codes have the property that the 9's complement of a decimal number is obtained directly by changing 1's to 0's and 0's to 1's.

	Binary	Excess-3	9's Complement	Binary	Excess-3
395	0011 1001 0101	0110 1100 1000	999 – 395 = 604	0110 0000 0100	1001 0011 0111

- The **8, 4, -2, -1 code** is an example of assigning both positive and negative weights to a decimal code.
- Ex: 0110 = **8** x 0 + **4** x 1 + **(-2)** x 1 + **(-1)** x 0 = 2

Gray Code

- Gray Code : Sometimes this code is used to represent digital data that have been converted from analog data.
- Advantage of Gray code over binary number : only one bit in the code group changes in going from one number to the next.
-

Gray Code

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

ASCII

- 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 a set contains between 36 and 64 elements if only capital letters are included. It requires binary code of 6 bits.
- Such a set contains between 64 and 128 elements if both uppercase and lowercase letters are included. It requires binary code of 7 bits.
- The standard binary code for the alphanumeric characters is the American Standard Code for Information Interchange (ASCII), which uses seven bits to code 128 characters.
- ASCII is a seven-bit code, but most computers manipulate an eight-bit quantity as a single unit called a *byte*.
- Therefore, ASCII characters most often are stored one per byte. The extra bit is sometimes used for other purposes, depending on the application.

American Standard Code for Information Interchange (ASCII)

	$b_7b_6b_5$							
$b_4b_3b_2b_1$	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	−	=	M	]	m	}
1110	SO	RS	.	>	N	∧	n	~
1111	SI	US	/	?	O	−	o	DEL

Control Characters

NUL	Null	DLE	Data-link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End-of-transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

Error-Detecting Code

- To detect errors in data communication and processing, an eighth bit is sometimes added to the ASCII character to indicate its parity.
- A *parity bit* is an extra bit included with a message to make the total number of 1's either even or odd.
-

	ASCII	Even Parity	Odd Parity
A	1000001	01000001	1100000
T	1010100	11010100	01010100

- The parity bit is helpful in detecting errors during the transmission of information from one location to another.
- This function is handled by generating an even parity bit at the sending end for each character.
- The eight-bit characters that include parity bits are transmitted to their destination.
- The parity of each character is then checked at the receiving end.
- If the parity of the received character is not even, then at least one bit has changed value during the transmission.
- This method detects one, three, or any odd combination of errors in each character that is transmitted.
- What is done after an error is detected depends on the particular application.