Digital Systems and Number Systems

We are in "Information age" since digital systems have such a prominent and growing role in modern society. They are involved in our business transactions, communications, transportation, medical treatment and entertainment. In industrial world they are heavily employed in design, manufacturing, distribution and sales.

Analog System
Analog systems process analog signals (continuous time signals) which can take any value within a range, for example the output from a speaker or a microphone.

BASIS FOR COMPARISON	ANALOG SIGNAL	DIGITAL SIGNAL
Basic	An analog signal is a continuous wave that changes over a time period.	A digital signal is a discrete wave that carries information in binary form.
Representation	An analog signal is represented by a sine wave.	A digital signal is represented by square waves.
Description	An analog signal is described by the amplitude, period or frequency, and phase.	A digital signal is described by bit rate and bit intervals.
Range	Analog signal has no fixed range.	Digital signal has a finite numbers i.e. 0 and 1.
Distortion	An analog signal is more prone to distortion.	A digital signal is less prone to distortion.
Transmit	An analog signal transmit data in the form of a wave.	A digital signal carries data in the binary form i.e. 0 nad 1.
Example	The human voice is the best example of an analog signal.	Signals used for transmission in a computer are the digital signal.

Digital System

- Digital systems process digital signals, which can take only a limited number of values (discrete steps), usually just two values are used: the positive supply voltage (+Vs) and zero volts (0V).
- Digital systems contain devices such as logic gates, flip-flops, shift registers and counters.

The general purpose digital computer is a best known example of **digital system**.

Generic Digital computer structure

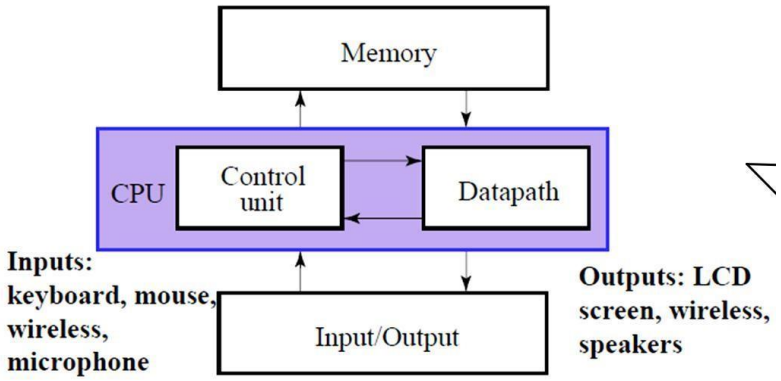


Fig: Block diagram of digital computer

- Actually processor contains 4 functional modules: CPU, FPU, MMU and internal cache.
- Here only CPU is specified.

Working principle of generic digital computer: Memory stores programs as well as input, output and intermediate data. The datapath performs arithmetic and other data-processing operations as specified by the program. The control unit supervises the flow of information between the various units. A datapath, when combined with the control unit, forms a component referred to as a *central processing unit*, or CPU. The program and data prepared by the user are transferred into memory by means of an input device such as a keyboard. An output device, such as a CRT (cathode-ray tube) monitor, displays the results of the computations and presents them to the user.

Advantages of digital system:

- Have made possible many scientific, industrial, and commercial advances that would have been unattainable otherwise.
- Less expensive
- More reliable
- Easy to manipulate
- Flexibility and Compatibility
- Information storage can be easier in digital computer systems than in analog ones. New features can often be added to a digital system more easily too

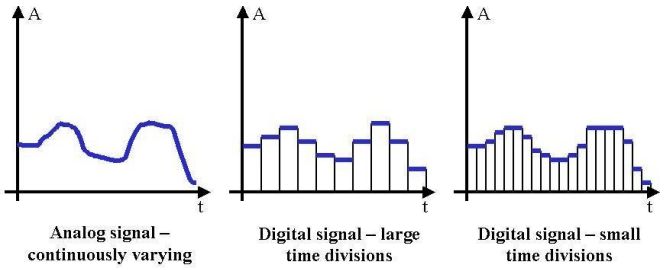
Disadvantages of digital system:

- Use more energy than analog circuits to accomplish the same tasks, thus producing more heat as well.
- Digital circuits are often fragile, in that if a single piece of digital data is lost or misinterpreted, the meaning of large blocks of related data can completely change.
- Digital computer manipulates discrete elements of information by means of a binary code.
- Quantization error during analog signal sampling.

Information Representation

Signals

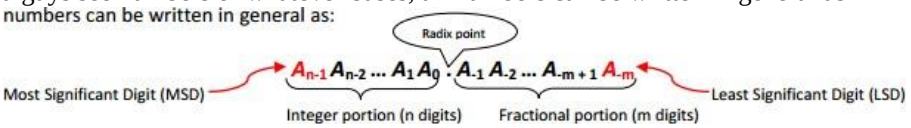
- Information variables represented by physical quantities.
- For digital systems, the variables take on discrete values.
- Two level or binary values are the most prevalent values in digital systems.
- Binary values are represented abstractly by:
  - digits 0 and 1
  - words (symbols) False (F) and True (T)
  - words (symbols) Low (L) and High (H)
  - and words On and Off.



Analog signal – continuously varying | Digital signal – large time divisions | Digital signal – small time divisions

Number Systems

Here we discuss positional number systems with Positive radix (or base) ***r***. A number with radix ***r*** is represented by a string of digits as below i.e. wherever you guys see numbers of whatever bases, all numbers can be written in general as



in which $\mathbf{0} \le A_i < r$ (since each being a symbol for particular base system viz. for r = 10 (decimal number system) A_i will be one of 0,1,2,…,8,9). Subscript i gives the position of the coefficient and, hence, the weight r^i by which the coefficient must be multiplied. In general, a number in base *r* contains *r* digits, 0, l, 2... *r*- 1, and is expressed as a power series in *r* with the general form:

$$(\text{Number})_r = A_{n-1}\,r^{n-1} + A_{n-2}\,r^{n-2} + \ldots + A_1\,r^1 + A_0\,r^0 + A_{-1}\,r^{-1} + A_{-2}\,r^{-2} + \ldots + A_{-m+1}\,r^{-m+1} + A_{-m}\,r^{-m}$$

$$(\text{Number})_r = \left(\sum_{i=0}^{i=n-1} A_i \cdot r^i\right) + \left(\sum_{j=-m}^{j=-1} A_j \cdot r^j\right)$$

$$\text{(Integer Portion)} + \text{(Fraction Portion)}$$

Decimal Number System (Base-10 system)
Radix (r) = 10
Symbols = 0 through r-1 = 0 through 10-1 = {0, 1, 2... 8, 9}
I am starting from base-10 system since it is used vastly in everyday arithmetic besides computers to represent numbers by strings of digits or symbols defined above, possibly with a *decimal point*. Depending on its position in the string, each digit has an associated value of an integer raised to the power of 10.
Example: decimal number 724.5 is interpreted to represent 7 hundreds plus 2 tens plus 4 units plus *5* tenths.

$$724.5 = 7 \text{ X } 10^2 + 2 \text{ X } 10^1 + 4 \text{ X } 10^0 + 5 \text{ X } 10^{-1}$$

Binary Number System (Base-2 system)
Radix (r) = 2
Symbols = 0 through r-1 = 0 through 2-1 = {0, 1}
A binary numbers are expressed with a string of 1'sand 0's and, possibly, a *binary point* within it. The decimal equivalent of a binary number can be found by expanding the number into a power series with a base of 2.
Example: $(11010.01)_2$ can be interpreted using power series as:
$(11010.01)_2 = 1 \text{ X } 2^4 + 1 \text{ X } 2^3 + 0 \text{ X } 2^2 + 1 \text{ X } 2^1 + 0 \text{ X } 2^0 + 0 \text{ X } 2^{-1} + 1 \text{ X } 2^{-2} = (26.25)_{10}$
Digits in a binary number are called bits (**B**inary dig**IT**s). When a bit is equal to 0, it does not contribute to the sum during the conversion. Therefore, the conversion to decimal can be obtained by adding the numbers with powers of 2 corresponding to the bits that are equal to 1. Looking at above example, $(11010.01)_2 = 16 + 8 + 2 + 0.25 = (26.25)_{10}$.

n	2^n	n	2^n	n	2^n
0	1	8	256	16	65,536
1	2	9	512	17	131,072
2	4	10	1,024	18	262,144
3	8	11	2,048	19	524,288
4	16	12	4,096	20	1,048,576
5	32	13	8,192	21	2,097,152
6	64	14	16,384	22	4,194,304
7	128	15	32,768	23	8,388,608

Table: Numbers obtained from 2 to the power of n

In computer work,
- 2^{10} is referred to as K (kilo),
- 2^{20} as M (mega),
- 2^{30} as G (giga),
- 2^{40} as T (tera) and so on.

Octal Number System (Base-8 system)
Radix (r) = 8
Symbols = 0 through r-1 = 0 through 8-1 = {0, 1, 2…6, 7}
An octal numbers are expressed with a strings of symbols defined above, possibly, an *octal point* within it. The decimal equivalent of a octal number can be found by expanding the number into a power series with a base of 8.
Example: $(40712.56)_8$ can be interpreted using power series as:
$(40712.56)_8 = 4 \text{ X } 8^4 + 0 \text{ X } 8^3 + 7 \text{ X } 8^2 + 1 \text{ X } 8^1 + 2 \text{ X } 8^0 + 5 \text{ X } 8^{-1} + 6 \text{ X } 8^{-2} = (16842.1)_{10}$

Hexadecimal Number System (Base-16 system)
Radix (r) = 16
Symbols = 0 through r-1 = 0 through 16-1 = {0, 1, 2…9, A, B, C, D, E, F}
A hexadecimal numbers are expressed with a strings of symbols defined above, possibly, a *hexadecimal point* with in it. The decimal equivalent of a hexadecimal number can be found by expanding the number into a power series with a base of 16.
Example: $(4D71B.C6)_{16}$ can be interpreted using power series as:

$$(4D71B.C6)_{16} = 4 \text{ X } 16^4 + D \text{ X } 16^3 + 7 \text{ X } 16^2 + 1 \text{ X } 16^1 + B \text{ X } 16^0 + C \text{ X } 16^{-1} + 6 \text{ X } 16^{-2}$$
$$= 4 \text{ X } 16^4 + 13 \text{ X } 16^3 + 7 \text{ X } 16^2 + 1 \text{ X } 16^1 + 11 \text{ X } 16^0 + 12 \text{ X } 16^{-1} + 6 \text{ X } 16^{-2}$$
$$= (317211.7734375)_{10}$$

Number Base Conversions
Case I: *Base-r system to Decimal*: Base-r system can be binary (r=2), octal (r=8), hexadecimal (r=16), base-60 system or any other. For decimal system as destination of conversion, we just use power series explained above with varying *r* and sum the result according to the arithmetic rules of base-10 system. I have already done examples for binary to decimal, octal to decimal and hexadecimal to decimal.
For refreshment lets assume base-1000 number $(458HQY)_{1000}$. Where n = 6 and m = 0.

$(458HQY)_{1000} = A_{n-1}\,r^{n-1} + A_{n-2}\,r^{n-2} + \ldots + A_1\,r^1 + A_0\,r^0 + A_{-1}\,r^{-1} + A_{-2}\,r^{-2} + \ldots + A_{-m+1}\,r^{-m+1} + A_{-m}\,r^{-m}$
$= 4 \text{ X } 1000^5 + 5 \text{ X } 1000^4 + 8 \text{ X } 1000^3 + \text{H X } 1000^2 + \text{Q X } 1000^1 + \text{Y X } 1000^0$
=Resulting number will be in decimal. Here I have supposed various symbols for base-1000 system. Don't worry, if someone gives you base-1000 number for conversion, he should also define all 1000 symbols (0-999).

Case II: *Decimal to Base-r system*: Conversion follows following algorithm.

1. Separate the number into **integer** and **fraction** parts if radix point is given.
2. Divide "*Decimal Integer part*" by base *r* repeatedly until quotient becomes zero and storing remainders at each step.
3. Multiply "*Decimal Fraction part*" successively by r and accumulate the integer digits so obtained.
4. Combine both accumulated results and parenthesize the whole result with subscript *r*.

Example I: Decimal to binary

- $(41.6875)_{10} = (?)_2$

Here Integer part = 41 and fractional part = 0.6875

Integer = 41

41	
20	1
10	0
5	0
2	1
1	0
0	1

$(41)_{10} = (101001)_2$

Fraction = 0.6875

0.6875
X 2
1.3750
X 2
0.7500
X 2
1.5000
X 2
1.0000

$(0.6875)_{10} = (0.1011)_2$

$(41.6875)_{10} = (101001.1011)_2$

Example II: Decimal to octal

- $(153.45)_{10} = (?)_8$

Here integer part = 153 and fractional part = 0.45

153	
19	1
2	3
0	2

This is simply division by 8, I am writing Quotients and remainders only.

$(153)_{10} = (231)_8$

0.45
X 8
3.60
X 8
4.80
X 8
6.40 (may not end, choice is upon you to end up)

Multiply always the portion after radix point.

$(0.45)_{10} = (346)_8$

$(153.45)_{10} = (231.346)_8$

Example III: Decimal to Hexadecimal

- $(1459.43)_{10} = (?)_{16}$

Here integer part = 1459 and fractional part = 0.43

1459	
91	4
5	11 (=B)
0	5

$(1459)_{10} = (5B4)_{16}$

0.43
X16
6.80
X16
12.80
X16
12.80 (Never ending...)

$(0.43)_{10} = (6CC)_8$

$(1459.43)_{10} = (5B4.6CC)_{16}$

Case III: *Binary to octal & hexadecimal and vice-versa*: Conversion from and to binary, octal and hexadecimal representation plays an important part in digital computers. Since,

- $2^3 = 8$, octal digit can be represented by at least 3 binary digits. (We have discussed this much better in class). So to convert given binary number into its equivalent octal, we divide it into groups of 3 bits, give each group an octal symbol and combine the result.
  - Integer part: Group bits from right to left of an octal point. 0's can be added to makeit multiple of 3 (**not compulsory**).
  - Fractional part: Group bits from left to right of an octal point. 0's must be added to if bits are not multiple of 3 (**Note it**).
- 24 = 16, each hex digit corresponds to 4 bits. So to convert given binary number into its equivalent hex, we divide it into groups of 4 bits, give each group a hex digit and combine the result. If hex point is given, then process is similar as of octal.
- 15 numbers in 4 systems summarized below for easy reference.

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

Example:

1. Binary to octal:
   $(10110001101011.11110000011)_2 = (\underbrace{010}_{2}\ \underbrace{110}_{6}\ \underbrace{001}_{1}\ \underbrace{101}_{5}\ \underbrace{011}_{3}.\ \underbrace{111}_{7}\ \underbrace{100}_{4}\ \underbrace{000}_{0}\ \underbrace{110}_{6})_2 = (26153.7406)_8$
2. Binary to hexadecimal:
   $(10110001101011.11110000011)_2 = (\underbrace{0010}_{2}\ \underbrace{1100}_{C}\ \underbrace{0110}_{6}\ \underbrace{1011}_{B}.\ \underbrace{1111}_{F}\ \underbrace{0000}_{0}\ \underbrace{0110}_{6})_2 = (2C6B.F06)_{16}$
3. From hex & octal to binary is quite easy, we just need to remember the binary of particular hex or octal digit.
   $(673.12)_8 = 110\ \ 111\ \ 011.\ 001\ \ 010 = (110111011.00101)_2$
   $(3A6.C)_{16} = 0011\ 1010\ 0110.\ 1100 = (1110100110.11)_2$

Complements

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

(r-1)'s Complement (diminished radix compl.)
(r-1)'s complement of a number N is defined as $(r^n - 1) - N$
Where **N** is the given number
r is the base of number system
n is the number of digits in the given number
To get the (r-1)'s complement fast, subtract each digit of a number from (r-1).

Example:
- 9's complement of 835_{10} is 164_{10} (Rule: $(10^n - 1) - N$)
- 1's complement of 1010_2 is 0101_2 (bit by bit complement operation)

r's Complement (radix complement)
r's complement of a number N is defined as $r^n - N$
Where **N** is the given number
r is the base of number system
n is the number of digits in the given number
To get the r's complement fast, add 1 to the low-order digit of its (r-1)'s complement.

Example:
- 10's complement of 835_{10} is $164_{10} + 1 = 165_{10}$
- 2's complement of 10102 is $0101_2 + 1 = 0110_2$

Subtraction with complements

The direct method of subtraction taught in elementary schools uses the borrow concept. When subtraction is implemented with digital hardware, this method is found to be less efficient than the method that uses complements.
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 $M + (r^n - N) = M - N + r''$.
2. If $M >= N$, the sum will produce an end carry, *r''*, 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 in a familiar form, take the r's complement of the sum and place a negative sign in front.

Example I:

Using 10's complement, subtract 72532 − 3250.

M =	72532
10's complement of N =	+ 96750
Sum =	169282
Discard end carry 10^5 =	−100000
Answer =	69282

M has 5 digits and *N* has only 4 digits. Both numbers must have the same number of digits; so we can write *N* as 03250. Taking the 10's complement of *N* produces a 9 in the most significant position. The occurrence of the end carry signifies that *M* >= *N* and the result is positive.

Example II:

Using 10's complement, subtract 3250 − 72532.

M =	03250
10's complement of N =	+ 27468
Sum =	30718

There is no end carry.

Answer: −(10's complement of 30718) = −69282

Example III:

Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction (a) $X - Y$ and (b) $Y - X$ using 2's complements.

(a)

X =	1010100
2's complement of Y =	+ 0111101
Sum =	10010001
Discard end carry 2^7 =	−10000000
Answer: $X - Y$ =	0010001

(b)

Y =	1000011
2's complement of X =	+ 0101100
Sum =	1101111

There is no end carry.

Answer: $Y - X$ = −(2's complement of 1101111) = −0010001 ■

Example IV: Repeating Example III using 1's complement

(a) $X - Y = 1010100 - 1000011$

X =	1010100
1's complement of Y =	+ 0111100
Sum =	10010000
End-around carry	+ 1
Answer: $X - Y$ =	0010001

(b) $Y - X = 1000011 - 1010100$

Y =	1000011
1's complement of X =	+ 0101011
Sum =	1101110

There is no end carry.

Answer: $Y - X$ = −(1's complement of 1101110) = −0010001

Binary Codes

Electronic digital systems use signals that have two distinct values and circuit elements that have two stable states. There is a direct analogy among binary signals, binary circuit elements, and binary digits. A binary number of *n* digits, for example, may be represented by *n* binary circuit elements, each having an output signal equivalent to a 0 or a 1. Digital systems represent and manipulate not only binary numbers, but also many other discrete elements of information. Any discrete element of information distinct among a group of quantities can be represented by a binary code. Binary codes play an important role in digital computers. The codes must be in binary because computers can only hold 1's and 0's.

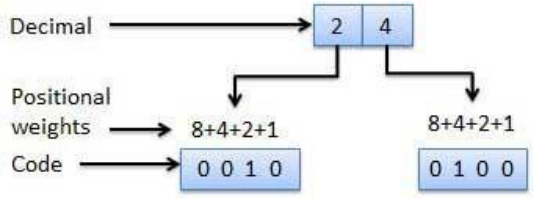
Advantages of Binary Code
Following is the list of advantages that binary code offers.

- Binary codes are suitable for the computer applications.
- Binary codes are suitable for the digital communications.
- Binary codes make the analysis and designing of digital circuits if we use the binary codes.
- Since only 0 & 1 are being used, implementation becomes easy.

Classification of binary codes
The codes are broadly categorized into following four categories.

- Weighted Codes
- Non-Weighted Codes
- Binary Coded Decimal Code
- Alphanumeric Codes
- Error Detecting Codes
- Error Correcting 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.



Non-Weighted Codes
In this type of binary codes, the positional weights are not assigned. The examples of non-weighted codes are Excess-3 code and Gray code.

1. Binary Coded Decimal (BCD)
The binary number system is the most natural system for a computer, but people are accustomed to the decimal system. So, to resolve this difference, computer uses decimals in coded form which the hardware understands. A binary code that distinguishes among 10 elements of decimal digits must contain at least four bits. Numerous different binary codes can be obtained by arranging four bits into 10 distinct combinations. The code most commonly used for the decimal digits is the straightforward binary assignment listed in the table below. This is called ***binary-coded decimal*** and is commonly referred to as **BCD**.

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

Table: 4-bit BCD code for decimal digits

- A number with *n* decimal digits will require 4n bits in BCD. E.g. decimal 396 is represented in BCD with 12 bits as 0011 1001 0110.
- Numbers greater than 9 has a representation different from its equivalent binary number, even though both contain 1's and 0's.
- Binary combinations 1010 through 1111 are not used and have no meaning in the BCD code.
- Example :
  $(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (10111001)_2$

Advantages of BCD Codes

- It is very similar to decimal system.
- We need to remember binary equivalent of decimal numbers 0 to 9 only.

Disadvantages of BCD Codes

- The addition and subtraction of BCD have different rules.
- The BCD arithmetic is little more complicated.
- BCD needs more number of bits than binary to represent the decimal number. So BCD is less efficient than binary.

2. Error-Detection codes

Electric wires or other communication medium can transmit binary information from one location to another. Any external noise introduced into the physical communication medium may change some of the bits from 0 to 1 or vice versa.

The purpose of an error-detection code is to detect such bit-reversal errors. One of the most common ways to achieve error detection is by means of a **parity bit**. A *parity bit* is the extra bit included to make the total number of 1's in the resulting code word either even or odd. A message of 4-bits and a parity bit P are shown in the table below:

Odd parity		Even parity	
Message	P	Message	P
0000	1	0000	0
0001	0	0001	1
0010	0	0010	1
0011	1	0011	0
0100	0	0100	1
0101	1	0101	0
0110	1	0110	0
0111	0	0111	1
1000	0	1000	1
1001	1	1001	0
1010	1	1010	0
1011	0	1011	1
1100	1	1100	0
1101	0	1101	1
1110	0	1110	1
1111	1	1111	0

Error Checking Mechanism:

- During the transmission of information from one location to another, an even parity bit is generated in the sending end for each message transmission. The message, together with the parity bit, is transmitted to its destination. The parity of the received data is checked in the receiving end. If the parity of the received information is not even, it means that at least one bit has changed value during the transmission.
- This method detects one, three, or any odd combination of errors in each message that is transmitted. An even combination of errors is undetected. Additional error-detection schemes may be needed to take care of an even combination of errors.

3. Gray code (Reflected code)

It is a binary coding scheme used to represent digits generated from a mechanical sensor that may be prone to error. Used in telegraphy in the late 1800s, and also known as "reflected binary code".

Bell Labs researcher Frank Gray patented Gray code in 1947. In Gray code, there is **only one bit location different between two successive values**, which makes mechanical transitions from one digit to the next less error prone. The following chart shows normal binary representations from 0 to 15 and the corresponding Gray code.

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

Application of Gray code

- Gray code is popularly used in the shaft position encoders.
- A shaft position encoder produces a code word which represents the angular position of the shaft.

The Gray code is used in applications where the normal sequence of binary numbers may produce an error or ambiguity during the transition from one number to the next. If binary numbers are used, a change from 0111 to 1000 may produce an intermediate

erroneous number 1001 if the rightmost bit takes more time to change than the other three bits. The Gray code eliminates this problem since only one bit changes in value during any transition between two numbers.

4. Alphanumeric codes

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. It is necessary to formulate a binary code for this set to handle different data types. If only capital letters are included, we need a binary code of at least six bits, and if both uppercase letters and lowercase letters are included, we need a binary code of at least seven bits.

The following three alphanumeric codes are very commonly used for the data representation.

- American Standard Code for Information Interchange (ASCII).
- Extended Binary Coded Decimal Interchange Code (EBCDIC).
- Five bit Baudot Code.

A. ASCII character code

The standard binary code for the alphanumeric characters is called ASCII (American Standard Code for Information Interchange). It uses seven bits to code 128 characters as shown in the table below. The seven bits of the code are designated by B_1 through B_7 with B_7 being the most significant bit.

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	NULL	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

NOTE:

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

Various control character symbolic notation stands for:

NULL	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

Example: ASCII for each symbol is ($B_7B_6B_5B_4B_3B_2B_1$)

G←100 0111, (←010 1000, h ←110 1000, >← 011 1110 and so on.

B. EBCDIC character code

EBCDIC (Extended Binary Coded Decimal Interchange Code) is another alphanumeric code used in IBM equipment. It uses **eight bits** for each character. EBCDIC has the same character symbols as ASCII, but the bit assignment for characters is different. As the name implies, the binary code for the letters and numerals is an extension of the binary-coded decimal (BCD) code. This means that the last four bits of the code range from 0000 through 1001 as in BCD.

Integrated Circuits (ICs)

An Integrated circuit is an association (or connection) of various electronic devices such as resistors, capacitors and transistors etched (or fabricated) to a semiconductor material such as silicon or germanium. It is also called as a **chip** or **microchip**. An IC can function as an amplifier, rectifier, oscillator, counter, timer and memory. Sometime ICs are connected to various other systems to perform complex functions.

Types of ICs

ICs can be categorized into two types

- Analog or Linear ICs

- Digital or logic ICs

Further there are certain ICs which can perform as a combination of both analog and digital functions.

Analog or Linear ICs: They produce continuous output depending on the input signal. From the name of the IC we can deduce that the output is a linear function of the input signal. Op-amp (operational amplifier) is one of the types of linear ICs which are used in amplifiers, timers and counters, oscillators etc.

Digital or Logic ICs: Unlike Analog ICs, Digital ICs never give a continuous output signal. Instead it operates only during defined states. Digital ICs are used mostly in microprocessor and various memory applications. Logic gates are the building blocks of Digital ICs which operate either at 0 or 1.

Advantages of ICs

- In consumer electronics, ICs have made possible the development of many new products, including personal calculators and computers, digital watches, and video games.
- They have also been used to improve or lower the cost of many existing products, such as appliances, televisions, radios, and high-fidelity equipment.
- The logic and arithmetic functions of a small computer can now be performed on a single VLSI chip called a microprocessor.
- Complete logic, arithmetic, and memory functions of a small computer can be packaged on a single printed circuit board, or even on a single chip.

Levels of Integration

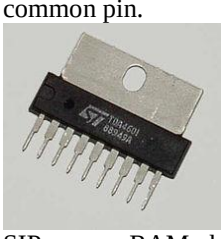
During 1959 two different scientists invented IC's. Jack Kilby from Texas Instruments made his first germanium IC during 1959 and Robert Noyce made his first silicon IC during the same year. But ICs were not the same since the day of their invention; they have evolved a long way. Integrated circuits are often classified by the number of transistors and other electronic components they contain:

- SSI (small-scale integration): Up to 100 electronic components per chip
- MSI (medium-scale integration): From 100 to 3,000 electronic components per chip
- LSI (large-scale integration): From 3,000 to 100,000 electronic components per chip
- VLSI (very large-scale integration): From 100,000 to 1,000,000 electronic components per chip
- ULSI (ultra-large-scale integration): More than 1 million electronic components per chip

SIP (Single In-line Package) and DIP (Dual In-line Package)

SIP

A **single in-line package** is an electronic device package which has one row of connecting pins. It is not as popular as the dual in-line package (DIP) which contains two rows of pins, but has been used for packaging RAM chips and multiple resistors with a common pin.



SIPs group RAM chips together on a small board. The board itself has a single row of pin-leads that resembles a comb extending from its bottom edge, which plug into a special socket on a system or system-expansion board. SIPs are commonly found in memory modules. SIP is not to be confused with SIPP which is an archaic term referring to Single In-line Pin Package which was a memory used in early computers.

DIP

Dual in-line package (DIP) is a type of semiconductor component packaging. DIPs can be installed either in sockets or permanently soldered into holes extending into the surface of the printed circuit board. DIP is relatively broadly defined as any rectangular package with two uniformly spaced parallel rows of pins pointing downward, whether it contains an IC chip or some other device(s), and whether the pins emerge from the sides of the package and bend downwards. A DIP is usually referred to as a **DIP*n***, where *n* is the total number of pins.

For example, a microcircuit package with two rows of seven vertical leads would be a DIP14. The photograph below shows three DIP14 ICs.

ASCII	EBCDIC
A character encoding standard for electronic communication	An eight-bit character encoding used mainly on IBM mainframe and IBM midrange computer operating systems
Stands for American Standard Code for Information Interchange	Stands for Extended Binary Coded Decimal Interchange Code
Uses 7 bits to represent a character	Uses 8 bits to represent a character
Represents 128 characters	Represents 256 character
Arranges the characters in consecutive order	Groups 9 characters at a time
Compatible with modern encodings such as Unicode	Not compatible with modern encodings such as Unicode
More efficient	Less efficient