Digital Logic

[CSC-151]

Nature of course: **Theory + Lab**

Text Book:

M. Morris Mao, "**Logic & Computer Design Fundamentals**", Pearson Education.

Marks Division:

Full Marks: 60+20+20

Pass Marks: 24+8+8

Internal Marks Division	**[20 Marks]**
1. Test Examination	[2+2+2+2]
2. Attendance	[5]
3. Class Activities/participation	[5]
a. Presentations	
b. Assignments	

Practical Marks Division	**[20 Marks]**
1. Practical Exam	[10]
2. Report	[5]
3. VIVA	[5]

Note: Fail in any portion will cause NQ.

Unit 1
Binary Systems

Contents:

- Digital Systems
- Binary Numbers
- Number base conversion
- Octal and hexadecimal numbers
- Binary Systems
- Integrated Circuits

Binary Systems:

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.

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**.

Examples of Digital Systems

Blu-ray Disc

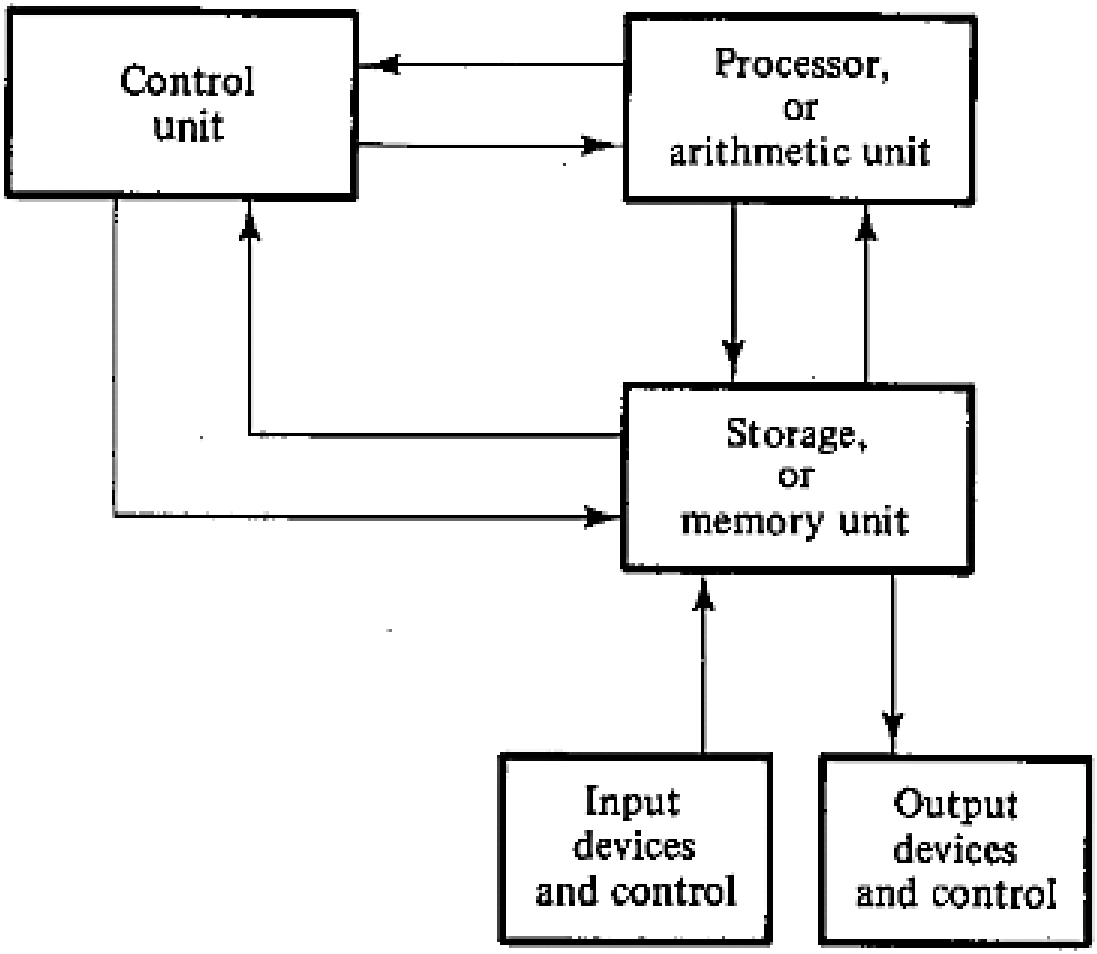


FIGURE 1-1

Block diagram of a digital computer

Generic Digital computer structure

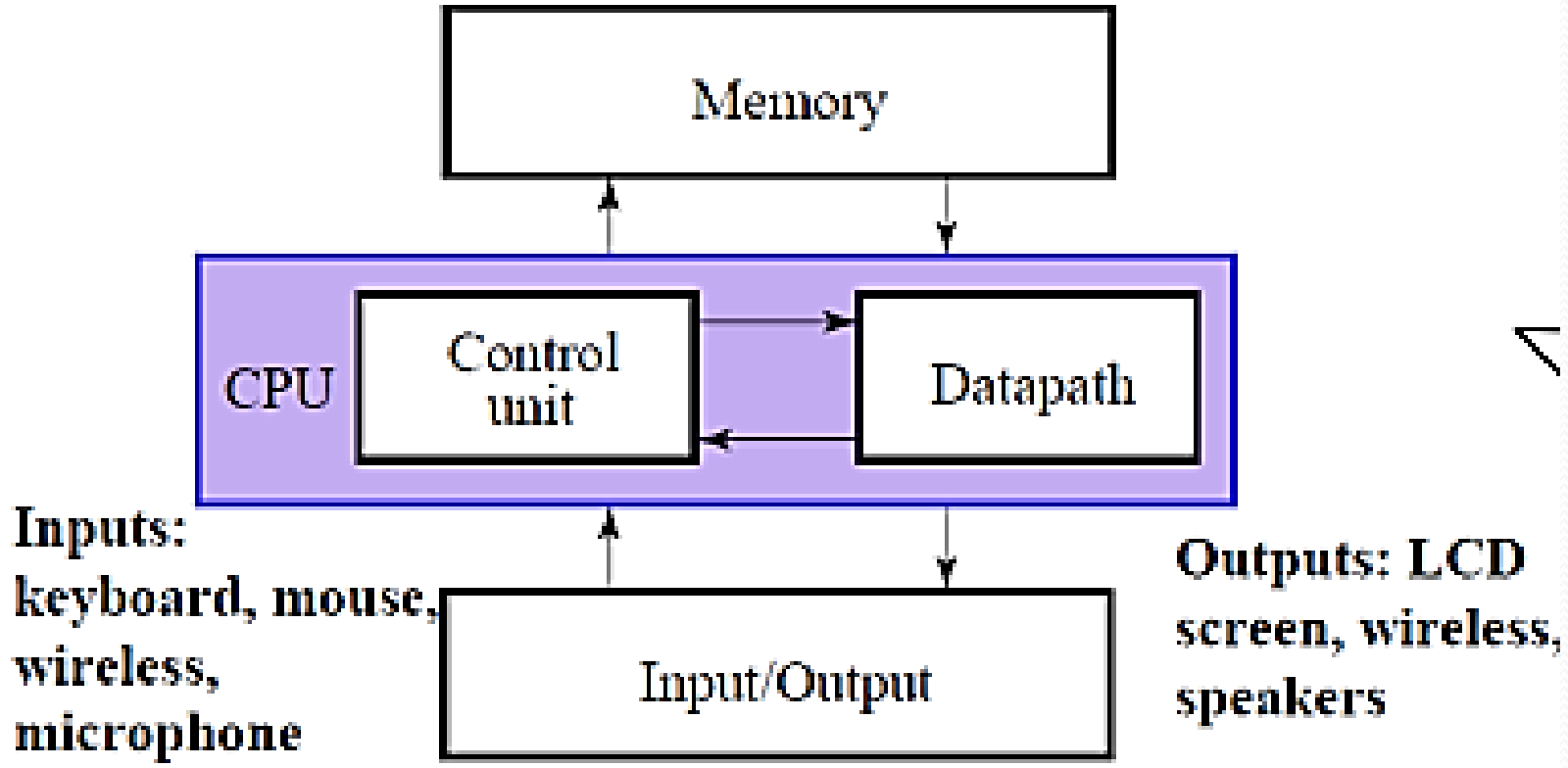


Fig: Block diagram of digital computer

Working principle of generic digital computer:

Memory stores programs as well as input, output and intermediate data. The data path 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 data path, 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.

Assignment I:

Advantages and Disadvantages of Digital System over Analog System.

Difference between Analog System and Digital System.

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.
- Quantization error during analog signal sampling.

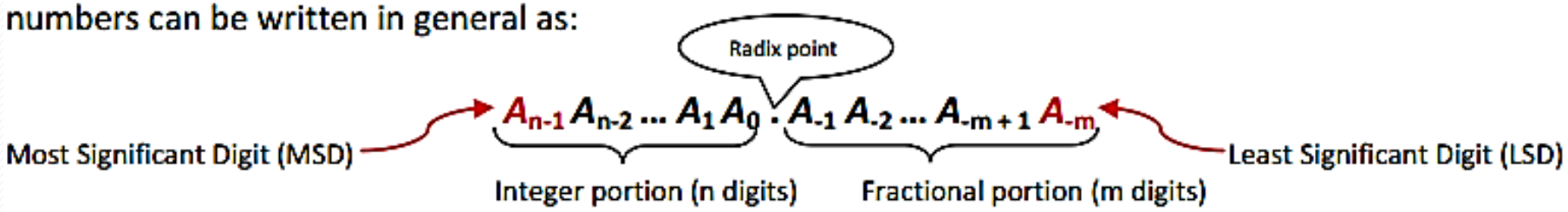
Information Representation

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.
- Binary values are represented by values or ranges of values of physical quantities

Number Systems

numbers can be written in general as:



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:

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

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

(Integer Portion) + (Fraction Portion)

Number System

- Decimal
- Binary
- Octal
- Hexadecimal

Numbering Systems		
System	Base	Digits
Binary	2	0 1
Octal	8	0 1 2 3 4 5 6 7
Decimal	10	0 1 2 3 4 5 6 7 8 9
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 A B C D E F

Decimal Number System (Base-10 system)

Radix (r) = 10

Symbols = 0 through r-1 = 0 through 10-1 = {0, 1, 2... 8, 9}

$724.5 = 7 * 10^{2} + 2 * 10^{1} + 4 * 10^{0} + 5 * 10^{-1}$

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:

$(11010.01)_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = (26.25)_{10}$

Digits in a binary number are called bits (**B**inary dig**IT**s).

Octal Number System (Base-8 system)

Radix (r) = 8

Symbols = 0 through r-1 = 0 through 8-1 = {0, 1, 2…6, 7}

Example: $(40712.56)_8$ can be interpreted using power series as:

$(40712.56)_8 = 4 \times 8^4 + 0 \times 8^3 + 7 \times 8^2 + 1 \times 8^1 + 2 \times 8^0 + 5 \times 8^{-1} + 6 \times 8^{-2} = (16842.1)_{10}$

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}

16 symbols are used in **hexadecimal**, and the 16 symbols are the **hexadecimal digits**. The hexa-decimal number system has a radix or base 16. It requires 16 symbols to represent a number in this system. The symbols are 0 to 9, A, B, C, D, E, F where the symbols A, B, C, D, E, F represent the decimal numbers 10, 11, 12, 13, 14, 15 respectively.

Example: $(4D71B.C6)_{16}$ can be interpreted using power series as:

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

TABLE 1-1
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

Number Base Conversions

Case I: *Base-r system to Decimal*:

Base-r system can be binary (r=2), octal (r=8), hexadecimal (r=16).

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.

Examples:

$\mathbf{B6A_{16}} = 11 \times 16^2 + 6 \times 16^1 + 10 \times 16^0 = 2816 + 96 + 10 = 2922_{10}$

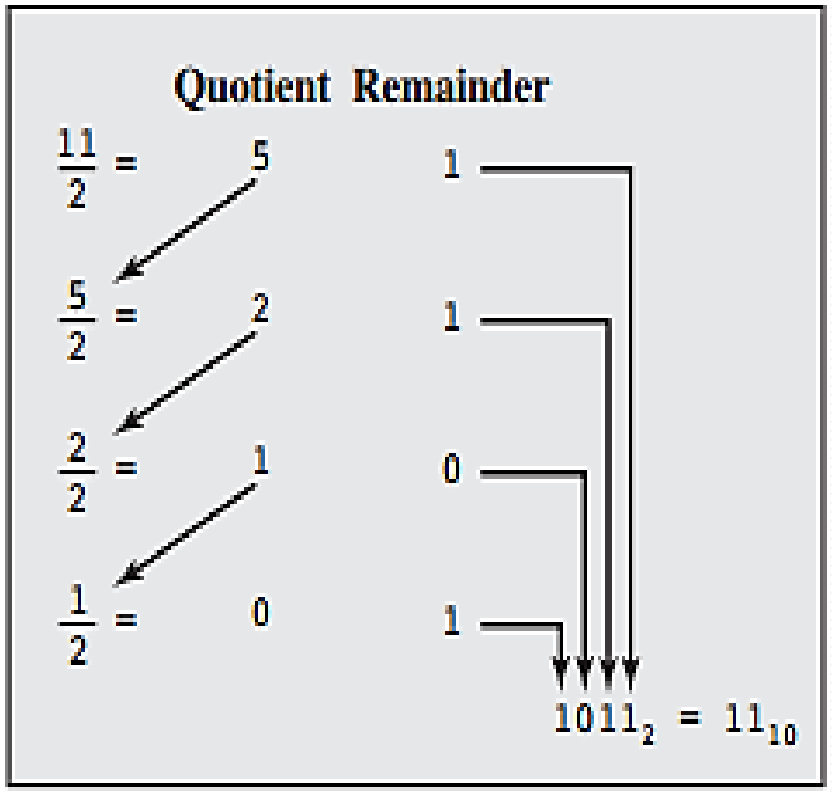
$\mathbf{273_8} = 2 \times 8^2 + 7 \times 8^1 + 3 \times 8^0 = 128 + 56 + 3 = 187_{10}$

Case II: *Decimal to Base-r system*:

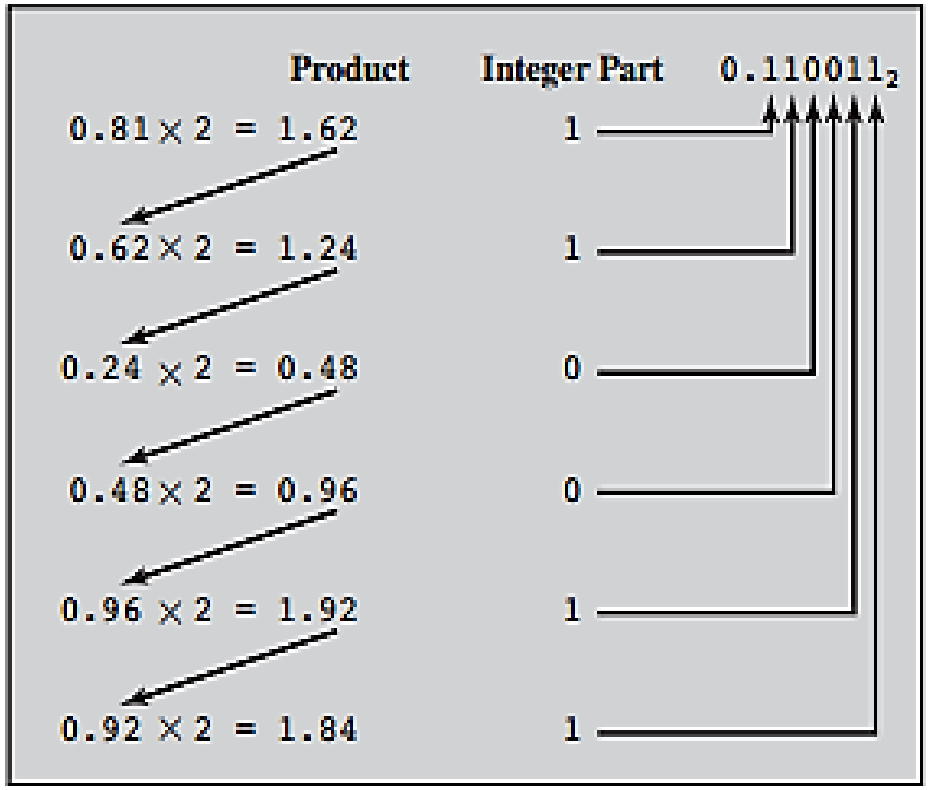
Conversion follows following algorithm.

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

Example I:Decimal to Binary(Base 2) Conversion



(a) 11_{10}



(a) $0.81_{10} = 0.110011_2$ (approximately)

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.

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.
   $(673.12)_8 = 110\ \ 111\ \ 011.\ 001\ \ 010 = (110111011.00101)_2$
   $(3A6.C)_{16} = 0011\ 1010\ 0110.\ 1100 = (1110100110.11)_2$

$$(\underbrace{10}_{2}\ \underbrace{110}_{6}\ \underbrace{001}_{1}\ \underbrace{101}_{5}\ \underbrace{011}_{3}\ .\ \underbrace{111}_{7}\ \underbrace{100}_{4}\ \underbrace{000}_{0}\ \underbrace{110}_{6})_2 = (26153.7460)_8$$

$$(\underbrace{10}_{2}\ \underbrace{1100}_{C}\ \underbrace{0110}_{6}\ \underbrace{1011}_{B}\ .\ \underbrace{1111}_{F}\ \underbrace{0010}_{2})_2 = (2C6B.F2)_{16}$$

$$(673.124)_8 = (\underbrace{110}_{6}\ \underbrace{111}_{7}\ \underbrace{011}_{3}\ .\ \underbrace{001}_{1}\ \underbrace{010}_{2}\ \underbrace{100}_{4})_2$$

$$(306.D)_{16} = (\underbrace{0011}_{3}\ \underbrace{0000}_{0}\ \underbrace{0110}_{6}\ .\ \underbrace{1101}_{D})_2$$

1. Convert the following to octal numbers:

(a) $1110101110_2 = \quad 001\underline{110}101\underline{110} = 001\ 110\ 101\ 110 = 1656_8$

(b) $111101.01101_2 = \quad 111\underline{101}.011\underline{010}_2 = 75.32_8$

2. Convert the following to their binary equivalents:

(a) $1573_8 = 001\ 101\ 111\ 011 = 1101111011_2$

(b) $64.175_8 = 110\ 100\ .\ 001\ 111\ 101 = 110100.001111101_2$

3. Convert the following to hexa-decimal numbers:

(a) $1111101101_2 = \underline{0011}1110\underline{1101} = 0011\ 1110\ 1101 = 3ED_{16}$

(b) $11110.01011_2 = \underline{1111}0.01011_2 = 0001\ 1110\ .\ 0101\ 1000 = 1E.58_{16}$

4. Convert the following to binary equivalents:

(a) $A748_{16} = 1010\ 0111\ 0100\ 1000 = 1010011101001000_2$

(b) $BA2.23C_{16} = 1011\ 1010\ 0010\ .\ 0010\ 0011\ 1100_2 = 101110100010.0010001111$

Case IV: *Octal to Hexadecimal and vice-versa*:
First of all, convert to binary and then to required form.

$1573_8 = 001101111011 = 0011\ 0111\ 1011 = 37B_{16}$

$A748_{16} = 1010\ 0111\ 0100\ 1000 = 001\ 010\ 011\ 101\ 001\ 000 = 123510_8$

Basic Arithmetic of Binary Numbers

Decimal Addition

0 + 0 = 0
0 + 1 = 1
1 + 1 = 2
1 + 2 = 3
1 + 3 = 4
1 + 4 = 5
1 + 5 = 6
1 + 6 = 7
1 + 7 = 8
1 + 8 = 9
1 + 9 = 10

Binary Addition

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10

When 1 is added to 1 in binary, the sum is 0 with a carry 1, and this carry is added to the sum of the adjacent bits.

Examples

1 0 1 0	[10]	1 0 0 1	[9]	1 1 1	[7]	1 0 1 0	[10]
+ 0 1 0 0	[+ 4]	+ 0 1 0 1	[+ 5]	+ 1 1	[+ 3]	+ 1 1 0 1	[+ 13]
1 1 1 0	[14]	1 1 1 0	[14]	1 0 1 0	[10]	1 0 1 1 1	[23]

Similarly, *while subtracting an '1' from a '0', a borrow is required.*

1 0 1 1	[11]	1 0 1 0	[10]	1 0 0 0	[8]		
- 1 0 0 1	[- 9]	- 0 1 0 1	[- 5]	+ 1 0 1	[- 5]		
0 0 1 0	[2]	0 1 0 1	[5]	0 0 1 1	[3]		

Signed and Unsigned Binary Numbers:

To represent negative integers, we need a notation for negative values. In an ordinary arithmetic, a negative number is indicated by a minus sign and a positive number by a plus sign. Because of hardware limitations, computers must represent everything with binary digits, commonly referred to as bits. 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.

For example, the string of bits 01001 can be considered as 9 (unsigned binary) or a +9 (signed binary) because the leftmost bit is 0. The string of bits 11001 represent the binary equivalent of 25 when considered as an unsigned number or as -9 when considered as a signed number because of the 1 in the leftmost position, which designates negative, and the other four bits, which represent binary 9.
The representation of the signed numbers in the last example is referred to as the ***signed-magnitude convention***.

As for ***signed-complement system***
The signed-complement system can use either the l ’s or the 2 ’s complement, but the 2’s complement is the most common.

As an example, consider the number 9 represented in binary with eight bits. +9 is represented with a sign bit of 0 in the leftmost position followed by the binary equivalent of 9 to give 00001001. Note that all eight bits must have a value and, therefore, 0's are inserted following the sign bit up to the first 1. Although there is only one way to represent +9, there are three different ways to represent -9 with eight bits:

In signed-magnitude representation:	10001001
In signed-l's-complement representation:	11110110
In signed-2's-complement representation:	11110111

In signed-magnitude, -9 is obtained from +9 by changing the sign bit in the leftmost position from 0 to 1.

In signed - l's complement, -9 is obtained by complementing all the bits of +9, including the sign bit.

The signed-2's-complement representation, of -9 is obtained by taking the 2's complement of the positive number, including the sign bit.

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 $M + (r^n - N) = M - N + r^n$
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 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

HEY! 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.

1. Binary Coded Decimal (BCD)(Also known as “8421”encoding)

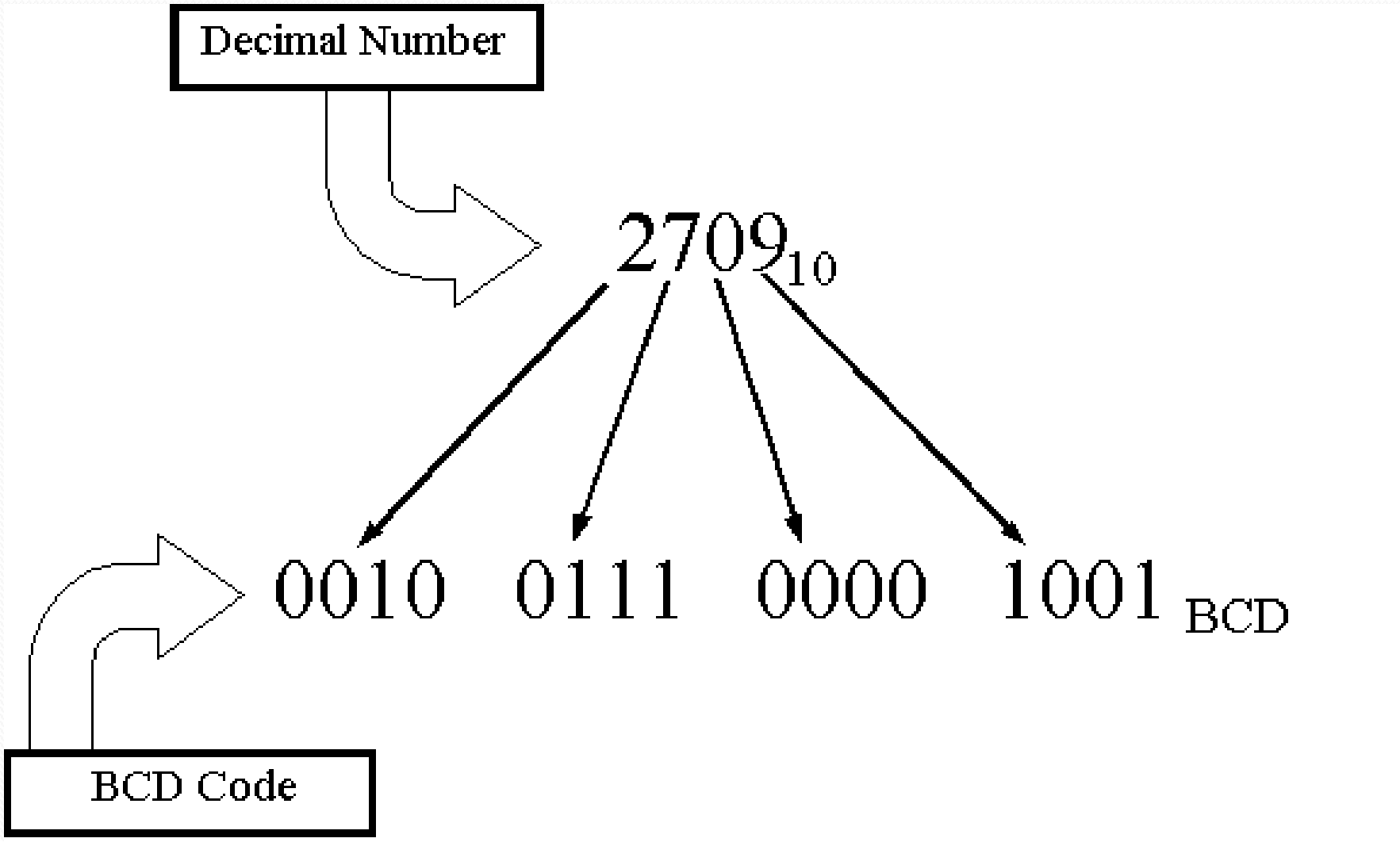
The binary number system is the most natural system for a computer, but people are accustomed to the decimal system. So, to resolve this difference, computer uses decimals in coded form which the hardware understands. A binary code that distinguishes among 10 elements of decimal digits must contain at 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.** BCD codes are used where the decimal information is directly transferred into or out of a digital system. Electronic calculators, digital voltmeters, digital clocks etc. work with BCD numbers. BCD codes have also been used in early computers, but modern computers do not use BCD numbers, due to their complexity in forming complements and calculations.

Decimal	BCD	
0	0 0 0 0	
1	0 0 0 1	
2	0 0 1 0	
3	0 0 1 1	
4	0 1 0 0	
5	0 1 0 1	
6	0 1 1 0	
7	0 1 1 1	
8	1 0 0 0	
9	1 0 0 1	
-	1 0 1 0	Unused
-	1 0 1 1	
-	1 1 0 0	
-	1 1 0 1	
-	1 1 1 0	
-	1 1 1 1	

Figure 1: Decimal to BCD mapping.

- 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:

$(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (10111001)_2$


Decimal Number
2709_{10}
0010 0111 0000 1001 BCD
BCD Code

2. Error-Detection codes

Binary information can be transmitted from one location to another by electric wires or other communication medium. 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”. Gray code was patented by Bell Labs researcher Frank Gray 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 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.

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

Binary to Gray Code Conversion:

1. MSB in Gray Code is same as MSB in Binary Number.
2. Going from left to right, add each adjacent pair of Binary code bits to get the next Gray code bit.(Discard Carries)

Eg:

10110 Binary No. to Gray Code

1	+	0	+	1	+	1	+	0	Binary Number
↓		↓		↓		↓		↓	
1		1		1		0		1	Gray Code

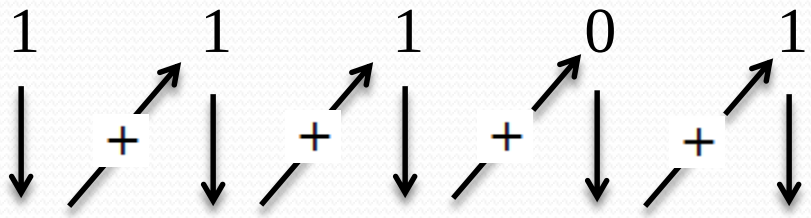
Thus, Gray Code is 11101.

Gray Code to Binary Conversion:

1. MSB in Binary Number is same as MSB in Gray Code.
2. Add each binary code bit generated to the Gray Code bit in the next adjacent position.(Discard Carries).

Eg:

Gray Code 11101 to Binary Code



Gray Code

Binary Number

Thus, Binary Number is 10110

Self complementing codes

- The 2421, the excess-3 and the 84-2-1 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 (i.e., by complementing each bit in the pattern). For example, decimal 395 is represented in the excess-3 code as 0110 1100 1000. The 9's complement of 604 is represented as 1001 0011 0111, which is obtained simply by complementing each bit of the code (as with the 1's complement of binary numbers).
- With self complementing codes, the design of the hardware is easier when it comes to say, 9's complement subtraction. It is a method of subtraction by addition, the negative number is converted to its 9's complement number. Self complementing codes give an easy way out. Suppose you have to subtract a number by say, 3(0011), its decimal 9's complement code is 1100 assigned to number 6. Similarly if 0(0000), then 9(1111). The pairs have complemented bits, which is easy to achieve in hardware, increasing the speed of operation.

BCD 8421, 2421, 8 4 -2 -1

- BCD and the 2421 code are examples of weighted 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.

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
	1010	0101	0000	0001
Unused	1011	0110	0001	0010
bit	1100	0111	0010	0011
combi-	1101	1000	1101	1100
nations	1110	1001	1110	1101
	1111	1010	1111	1110

BCD table

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

Excess-3 code

- The Excess-3 code is also called as XS-3 code. It is non-weighted code used to express decimal numbers. The Excess-3 code words are derived from the 8421 BCD code words adding (0011 in binary) or (3 in decimal)10 to each code word in 8421.

Decimal	BCD 8 4 2 1	Excess-3 BCD + 0011
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0

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.

- **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.

7-Bit ASCII code format: $B_8\ B_7\ B_6\ B_5\ B_4\ B_3\ B_2\ B_1$ *where B= 0 or 1*

Last eighth bit B_8 is the parity bit used to catch 1-bit error occurred during data transmission.

American Standard Code for Information Interchange (ASCII)

$B_4B_3B_2B_1$	$B_7B_6B_5$ = 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

Various control character symbolic notation stands for:

NULL	NULL
SOH	Start of heading
STX	Start of text
ETX	End of text
EOT	End of transmission
ENQ	Enquiry
ACK	Acknowledge
BEL	Bell
BS	Backspace
HT	Horizontal tab
LF	Line feed
VT	Vertical tab
FF	Form feed
CR	Carriage return
SO	Shift out
SI	Shift in
SP	Space

DLE	Data link escape
DC1	Device control 1
DC2	Device control 2
DC3	Device control 3
DC4	Device control 4
NAK	Negative acknowledge
SYN	Synchronous idle
ETB	End of transmission block
CAN	Cancel
EM	End of medium
SUB	Substitute
ESC	Escape
FS	File separator
GS	Group separator
RS	Record separator
US	Unit separator
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.

- **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 though 1001 as in BCD.

Binary storage and register

- Flip-flop is a 1 bit memory cell which can be used for storing the digital data. To increase the storage capacity in terms of number of bits, we have to use a group of flip-flop. Such a group of flip-flop is known as a Register. The n-bit register will consist of n number of flip-flop and it is capable of storing an n-bit word.

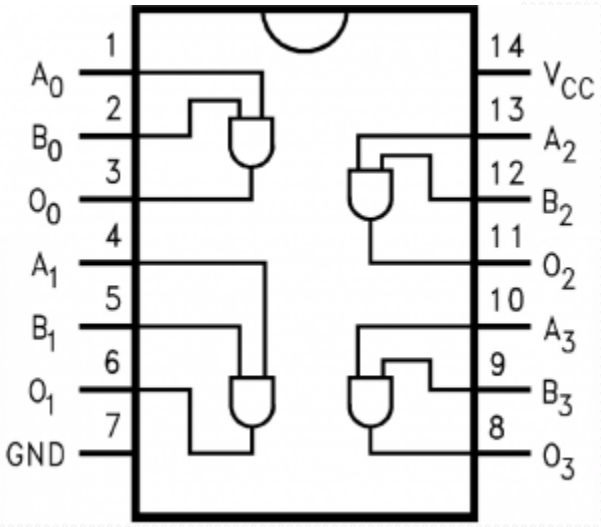
Binary logic

Binary logic deals with binary variables, which take on two discrete values, and with the operations of mathematical logic applied to these variables. The two values the variables take may be called by different names, but for our purpose, it is convenient to think in terms of binary values and assign 1 or 0 to each variables. Associated with binary variables there are three basic logical operations called AND, OR, and NOT:

- AND. This operation is represented by a dot or by the absence of an operator. For example, Z=X.Y or Z=XY.
- OR. This operation is represented by a plus symbol. For example, Z=X+Y.
- NOT. This operation is represented by a bar over the variable. For example Z=$\bar{X}$.

Introduction to Integrated Circuits

- All logic gates are available in Integrated Circuits (ICs)
- ICs are categorized in three different ways:
  - The underlying technology upon which their circuitry is based:
    - Transistor-Transistor Logic - **TTL**
    - Complementary Metal Oxide Semiconductor - **CMOS**
  - The scale of integration:
    - Small Scale Integration - **SSI**
    - Medium Scale Integration - **MSI**
    - Large Scale Integration - **LSI**
    - Very Large Scale Integration - **VLSI**



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

A particular IC is categorized as either linear (analog) or digital, depending on its intended application.

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. The fundamental building blocks of digital ICs are logic gates, which work with binary data, that is, signals that have only two different states, called low (logic 0) and high (logic 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

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: an electronic device package which has one row of connecting pins.
DIP contains two rows of pins.

Fig: SIP

Fig: DIP

- **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.

SIMM (Single In-line Memory Module) and DIMM (Dual In-line Memory Module)

Theses two terms (SIMM and DIMM) refer to a way series of dynamic random access memory integrated circuits modules are mounted on a printed circuit board and designed for use in personal computers, workstations and servers.

SIMM

Short for **Single In-line Memory Module, SIMM** is a circuit board that holds six to nine memory chips per board, the ninth chip usually an error checking chip (parity/non parity) and were commonly used with Intel Pentium or Pentium compatible motherboards. SIMMs are rarely used today and have been widely replaced by DIMMs. SIMMs are available in two flavors: 30 pin and 72 pin. 30-pin SIMMs are the older standard, and were popular on third and fourth generation motherboards. 72-pin SIMMs are used on fourth, fifth and sixth generation PCs.

4MB 72-PIN SIMM

512MB DIMM

DIMM

Short for **Dual In-line Memory Module**, **DIMM** is a circuit board that holds memory chips. DIMMs have a 64-bit path because of the Pentium Processor requirements. Because of the new bit path, DIMMs can be installed one at a time, unlike SIMMs on a Pentium that would require two to be added. Above is an example image of a 512MB DIMM memory stick.

SO-DIMM is short for **Small Outline DIMM** and is available as a 72-pin and 144-pin configuration. SODIMMs are commonly utilized in laptop computers.

Some of the advantages DIMMs have over SIMMs:

- DIMMs have separate contacts on each side of the board, thereby providing twice as much data as a single SIMM
- The command address and control signals are buffered on the DIMMs. With heavy memory requirements this will reduce the loading effort of the memory.

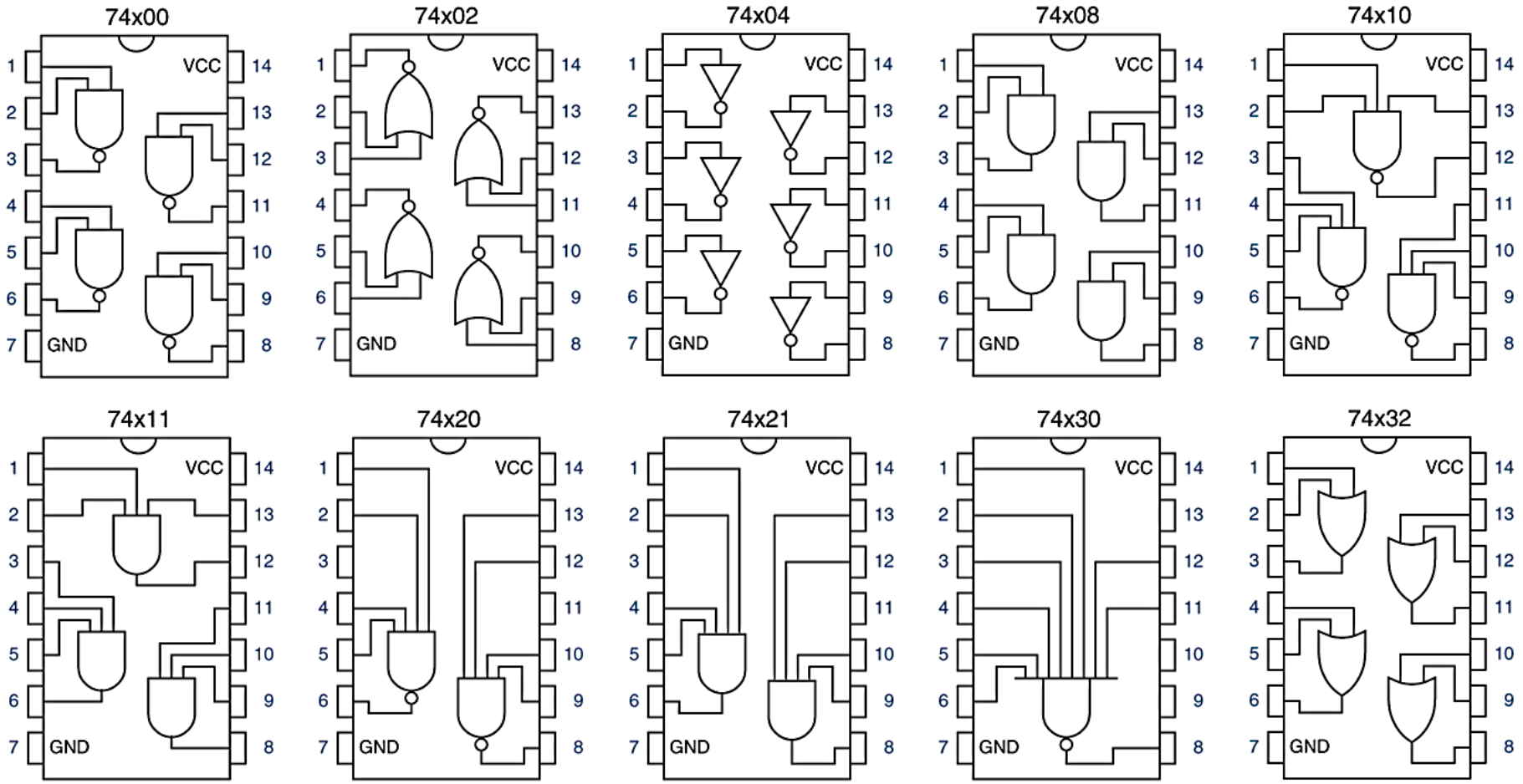


Figure 1-5

Pin diagrams for a few 7400-series SSI ICs.

Thank You!