Digital Fundamentals & Computer Organisation

V.B. Kirubanand M.C.A P.G.D.B.A
Lecturer & HOD Computer Science Department
Sri Krishna Arts & Science College
Coimbatore - 641 008

R.K.Publishers
Coimbatore - 641 028.

R.K.Publishers
Coimbatore - 641 028.

This book or part there of cannot be translated or reproduced in any form without the written permission of the author and the publisher

First Edition : 2002

Price Rs : 100/-

Published by
R.K.Publishers
26, Rajeswari Nagar, Nava India Road,
Coimbatore - 641 028.
Ph : 0422 - 2315727
E-Mail : publishers_rk@rediffmail.com

PREFACE

This book is an outcome of my long teaching experience to the computer science students. The subject "Digital Fundamentals & Computer Organisation" deals with system Architecture, Gates, Register etc., This book is designed for B. Sc Computer Science, B. Sc Electronics, B. C. A(Computer Applications) & M. C. A. Syllabus based on Bharathiar University, Coimbatore.

I do not claim to the originality of the subject matter which has been borrowed from various sources and has been presented in a very simple format.

I wish to acknowledge our managing Trustee Sri. S. Venkatram and our Principal Sri. Dr. D. Muthuraj who encouraged my endeavours and extended his fullest support in the preparation of this book.

My wife Smt. K. Sesha Sree, helped me while writing the information.

Staff of our college encouraged and gave me full confidence for bringing out this book.

Last but not least, I am very thankful to Dr. E. Balagurusamy , Vice Chancellor, Anna U niversity, Chennai, and my Co - brother Prof. C. Devaraj who encouraged and supported me by all means in drafting of this book.

No work is fulfilled without comments, remarks and suggestions ; I expect the same for improvement in future. I am grateful to M/s. R.K.Publishers Coimbatore, for publishing the book.

Dedicated to my Parents and Family

V.B.K

CONTENTS

Chapter - 1
NUMBER SYSTEMS

1.1 INTRODUCTION

What is Number System?

The word Number most us immediately think of the familiar decimal number system with its 10 digits.

i.e., : 0,1,2,3,4,5,6,7,8,9

In modern computer do not process decimal number. Instead, they work with binary numbers which use only the digits 0 and 1. This creates a problem. People do not like working with binary numbers because they are very long when representing larger decimal quantities. Therefore, octal and hexadecimal numbers are now widely used to compress long strings of binary numbers.

In this topic discusses binary, octal and hexadecimal numbers. Besides learning to count in these new number system, you will learn how to convert from decimal to binary octal and hexadecimal and vice versa.

1.2 DIGITAL COMPUTER AND SYSTEM

Almost all Digital computer and systems are based on binary (two-state) operation. For Instance, the switch shown in Fig. 1.1 can be open a closed; therefore, a switch is one example of a natural Binary device.

Fig. 1.1

The magnetic core shown in Fig. 1.2 is another example of two state operation. With the right-hand rule, conventional current units the wire produces clock wise flux; reverse the current and you get a counter clock wise flux. Because there are only two possible directions for the flux, the magnetic core a binary or two-state device. Earlier computers used thousands of magnetic cores to store the binary instructions and data needed during a calculation.

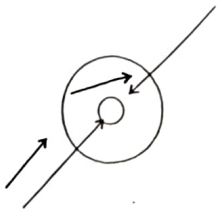


Fig. 1.2

A Punched card (Fig. 1.3) is another example of the two-state concept. A hole in the card is one possibility and no hole is the other. Using a pre-arranged code, a card punch machine with a keyboard can produce a stack of punched cards containing binary instructions and data. Some computers use punched cards to enter the instruction and data into a computer.

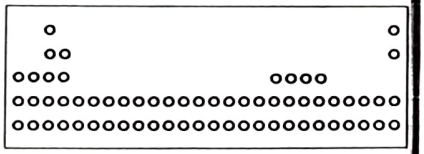


Fig. 1.3

Tape recorders can magnetize some points on the tape while leaving other points unmagnetized. When a prearranged code used, a row of points can represent either a coded instruction or data (Fig. 1.4)). In this way, a reel of tape can store thousands of binary instruction and data for later use in a computer or other digital system.

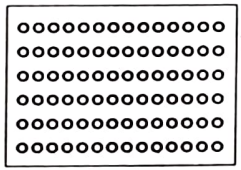


Fig. 1.4

Event the transistor circuits used in digital computer and system are designed to operate in either of two states, typically at cut off or saturation. As a result the output of a transistor circuit is either a low or a high voltage. When you look at digital signals with an oscilloscope, you find that each voltage is either low or high. These digital signals change during a computer run, so that they appear like pulses (Fig. 1.5).

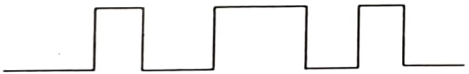


Fig. 1.5

In conclusion, switches, cores, punched cards, tape, transistor and almost all other digital components are based on binary operation. This is why it is convenient to use binary numbers when analyzing or designing digital circuits. Besides knowing how to want with binary numbers, it is necessary to learn how to convert from binary to decimal and vice versa.

1.3 NUMBER BASE CONVERSION

a) BINARY-TO-DECIMAL CONVERSION

Binary Number Table

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

The table lists the binary numbers from 0000 to 1111. But how do you convert larger binary number into their decimal values? For instance, what does binary 101001 represent in decimal numbers? This section shows how to convert a binary number quickly and easily into its decimal equivalent.

b) POSITIONAL NOTATION AND WEIGHT

We can express any decimal integer (a whole number) in units, tens, hundreds, thousands and so on. For instance, decimal number 3456

$$3\ 4\ 5\ 6 = 3000 + 400 + 50 + 6$$

In powers of 10, It becomes,

$$3456 = 3(10^3) + 4(10^2) + 5(10^1) + 6(10^0)$$

The decimal number system is an example of positional notation; each digit position has a weight or value. With decimal numbers the weight are units, tens, hundreds, thousands and so on. The sum of all the digits multiplied by their weight gives the total amount being represented. In the fore going example the 3 is multiplied by a weight of 1000 the 4 by a weight of 100 the 5 by a weight of 10 and the 6 by a weight of 1 the total us.

$$3000 + 400 + 50 + 6 = 3456$$

c) BINARY WEIGHTS

In a similar way, we can rewrite any binary number is terms of weight. For instance, binary number 110 becomes.

$$110 = 100 + 10 + 0 \qquad (\Sigma q.\ (a))$$

In decimal number, this may be rewritten as

$$6 = 4 + 2 + 0 \qquad (\Sigma q.\ (b))$$

Writing a binary number as shown in Σq (a) is the same as splitting its decimal equivalent into units, 2s and 4s as indicated by Σq. (b). In other words, each digit position is a binary number has a weight. The least significant digit (The one on the right) has a weight of 1. The second position from the right has a weight of 2. The next 4; and then 8, 16, 32 and so forth. These weight are in axending power of 2; we can write the fore going equation as

$$6 = 4 + 2 + 0$$
$$6 = 1(2^2) + 1(2^1) + 0(2^0)$$

Whenever you look at a binary number, you can find it decimal equivalent as follows. When there is a 1 in a digit position, add the weight of that position.

When there is a 0 in a digit position, disregard the weight of that position. For example, binary number 001 has a decimal equivalent of

(8421)

0001

$$0 + 0 + 1 = 1$$

As another example, binary number 0101 is equivalent to

$$0 + 4 + 0 + 1 = 5$$

Still another example 10101, which is equivalent to

$$16 + 0 + 4 + 0 + 1 = 21$$

1.4 STREAMLINED METHOD

We can streamline binary-to-decimal conversion by the following procedure :

1. Write the binary number
2. Directly under the binary number write 1,2,4,8,16,32....., working from right to left.
3. It a zero appears in a digit position, cross out the decimal weight for that position.
4. Add the remaining weight to obtain the decimal equivalent.

As an example of this approach. Let us convert binary 011 to its decimal equivalent.

Step 1	0	1	1			
Step 2	~~4~~	2	1			
Step 3		2	1			
Step 4	0	+ 2	+ 1	=	3	

As another example, notice how quickly 11001 is converted to its decimal equivalent.

1	1	~~0~~	~~0~~	1

$16 + 8 + \cancel{4} + \cancel{2} + 1 = 25$

1.5 FRACTIONS

So for, we have discussed binary integers (whole numbers). How are binary fractions converted into corresponding decimal equivalent? For instance, what is the decimal equivalent of 0.101? In this case, the weight of digit positions to the right of the binary point are give by $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}$ and so on. In powers of 2, the weight are,

2^{-1} 2^{-2} 2^{-3} 2^{-4} etc. in decimal form :

↓ ↓ ↓ ↓

0.5 0.25 0.125 0.0625 etc.

Here is an example. Binary fraction 0.101 has a decimal equivalent of

0 . 1 0 1

↓ ↓

0 . 0.5 + 0 + 0.125 = 0.625

(0.25)

Another example, the decimal equivalent of 0.1101 is

0 . 1 1 0 1

↓ ↓ ↓

0 . 0.5 + 0.25 + 0 + 0.0625 = 0.8125

(0.125)

1.6 MIXED NUMBERS

For mixed numbers (number with an integer and fractional part) handle each part acceding to the rules just development. The weight for a mixed number are :

...etc., 2^3 2^2 2^1 2^0 . 2^{-1} 2^{-2} 2^{-3} etc.,

↑

Binary Point

Power of 2 Table

Power of 2	Decimal Equivalent	Abbreviation
2^0	1	
2^1	2	
2^2	4	
2^3	8	
2^4	16	
2^5	32	
2^6	64	
2^7	128	
2^8	256	
2^9	512	
2^{10}	1,024	1k
2^{11}	2,048	2k
2^{12}	4,096	4k
2^{13}	8,192	8k
2^{14}	16,384	16k
2^{15}	32,768	32k
2^{16}	65,536	64k

For future reference, table lists power of 2 and their decimal equivalent and the number of K. The abbreviation K stands for 1024. Therefore, 1k means 1024, 2k stands for 2048, 4k represents 4096 and so on. Many persoanl computers have 64k memories this means that they can store up to 65,536 bytes in the memory section.

Example 1.

Covert binary 110.001 to a decimal number.

Solution

1	1	0	.	0	0	1		
↓	↓	↓		↓	↓	↓		
4	2	1	.	0.5	0.25	0.125	=	6.125

Example 2.

What is the decimal value of binary 10 11.11?

Solution

1	0	1	1	.	1	1		
↓	↓	↓	↓		↓	↓		
8	4	2	1	.	0.5	0.25	=	11.75

Example 3

As computer has a 256k memory. What is the decimal equivalent of 256k?

Solution

$$256 \times 1024 \rightarrow 262{,}144$$

This means that the computer can store 262,144 bytes is the memory section.

1.7 DECIMAL-TO-BINARY CONVERSION

One way to convert a decimal number into its binary equivalent is to reverse the process described in the preceding section. For instance, suppose that you want to convert decimal 10 into the corresponding binary number. All you need to do is express 10 as a sum of power of 2, and then write 1's and 0's in the appropriate digit positions:

10	=	8	0	2	0
	→	1	0	1	0

As another example :

15	=	8	4	2	1
	→	1	1	1	1

(a) DOUBLE DABBLE

A popular way to convert decimal numbers to binary numbers is the double - dabble method. In the double-dabble method you progressively divide the decimal number by 2, writing down the remainder after each division. The remainder, taken in reverse order, form the binary number. The best way to understand the method is to go through an example step by step.

Here is how to convert decimal 12 to its binary equivalent :

Step 1 : Divide 12 by 2, writing your work like this :

2	12
2	6 - 0
2	3 - 0
	1 - 1

↑ Read up

The quotient is 6 with a remainder 0.

Step 2 : Divide 6 by 2 to get 3 quotient and remainder 0.

Step 3 : Divide 3 by 2 to get 1 quotient and remainder 1.

∴ Finally 2t gives 1100

Whenever you arrive at a quotient of 1 with a remainder of 1, the conversion is finished. The remainder when read downward give the binary equivalent. In this example, binary 1100 is equivalent to decimal 12.

There is no need to keep writing down 2 before each division because you are always dividing by 2. Here is an efficient way to show the conversion of decimal 12 to its binary equivalent.

2	12	
2	6 - 0	Read up
2	3 - 0	
	1 - 1	1100

(b) FRACTIONS

As for as fractions are concerned, you multiply by 2 record a carry in the integer position. The carries read downward are the binary fraction. As an example, 0.85 converts to binary as

$0.85 \times 2 = 1.7 = 0.7$ with a carry of 1
$0.7 \times 2 = 1.4 = 0.4$ with a carry of 1
$0.4 \times 2 = 0.8 = 0.8$ with a carry of 0
$0.8 \times 2 = 1.6 = 0.6$ with a carry of 1
$0.6 \times 2 = 1.2 = 0.2$ with a carry of 1
$0.2 \times 2 = 0.4 = 0.4$ with a carry of 0

Read down

i.e., 1 1 0 1 1 0

Reading the carries downward gives binary fraction 0.110110. In this case, we stopped the conversion process after getting six binary digits. Because of this, the answer is an approximation. It more accuracy is needed, continue multiplying by 2 until you have as many digits as necessary for your application.

1.8 OCTAL NUMBERS

The base of a number system equals the number of digit it use. The decimal number system has a base of 10. Because it uses the digits 0 to 9. The binary number system has a base of 2 because it uses only the digits 0 and 1. The Octal Number has a base of 8. Although we can use any eight digit, it is customary to use the first eight decimal digits :

0, 1, 2, 3, 4, 5, 6, 7

(There is no 8 or 9 in the octal number code) These digits, 0 through 7, have exactly the same physical meaning as decimal symbols; that is 2 stands for **, 5 symbolizes ***** and so on.

(a) OCTAL-TO-DECIMAL CONVERSION.

How do we convert octal numbers to decimal numbers?

In the octal number system each digit position corresponds to a power of 8 as follows :

$8^4 \quad 8^3 \quad 8^2 \quad 8^1 \quad 8^0 \quad . \quad 8^{-1} \quad 8^{-2} \quad 8^{-3} \quad 8^{-4}$

↑

Octal Point

Therefore to convert from Octal to decimal, multiply each octal digit by its weight and add the resulting products.

For instance, Octal 33 converts to decimal like this

$3(8^1) + 3(8^0)$

$= 24 + 3 = 27.$

Here is another example Octal 3 4 5 converts to.

$3(8^2) + 4(8^1) + 5(8^0)$

$= 192 + 32 + 5 = 229$

(b) DECIMAL-TO-OCTAL CONVERSION

How do you convert in the opposite direction, that is, from decimal to Octal? Octal dabble, a method similar to double dabble, is used with octal numbers. Instead of dividing by 2 (the base of binary numbers), you divide by 8 (the base of octal numbers) writing down the remainders after each division.

As an example, convert decimal 168 as follows :

8	168
8	21 - 0
	2 - 5

Read up 2 5 0

As an another example

8	175
8	21 - 7
	2 - 5

Read up 2 5 7

(c) FRACTIONS

An the decimal fractions, multiply instead of divide, writing the carry into the integer position. An example of this is to convert decimal 0.23 into an octal fraction.

$0.23 \times 8 = 1.84 = 0.84$ with a carry of 1
$0.84 \times 8 = 6.72 = 0.72$ with a carry of 6
$0.72 \times 8 = 5.76 = 0.76$ with a carry of 5 ↓
etc., Read down

The carries read downward give the Octal Fraction 0.165. We terminated after three places : if more accuracy was required, we would continue multiplying to obtain more octal digits.

Another example.

$0.22 \times 8 = 1.76 = 0.76$ with a carry of 1
$0.76 \times 8 = 6.08 = 0.08$ with a carry of 6
$0.58 \times 8 = 0.64 = 0.64$ with a carry of 0 ↓
Read down

The comes read downward give the Octal Fraction 0.165. We terminated after three places : if more accuracy were required, we would continue multiplying to obtain more octal digits.

(d) OCTAL-TO-BINARY CONVERSION.

The base of octal number is 8. In this topic 8 is the third power of 2(the base of binary numbers), you can convert from octal to binary as follows : Change each octal digit to its binary equivalent. For instance, change octal 25 to its binary equivalent as follows :

2	5
010	101

Here, each digit coverts to its binary equivalent (2 becomes 010; and 5 becomes 101). The binary equivalent of octal 25 is 010101. Often, a space is left between groups of 3 bits: this makes it easier to read the binary number.

As another example. Octal 2775 coverts to binary as follows :

2	7	7	5
↓	↓	↓	↓
010	111	111	101

Here binary 010 111 111 101 is equivalent to octal 2775. Notice how much easier the binary number is to read if we leave as space between groups of 3 bits.

(e) MIXED OCTAL NUMBERS

Mixed octal numbers are no problem, convert each octal digit to its equivalent binary value. Octal 23.456 becomes.

2	3	·	4	5	6
↓	↓		↓	↓	↓
010	011		100	101	110

(f) BINARY-TO-OCTAL CONVERSION

Conversion from binary to octal is a reversal of the foregoing procedures. Simply remember to group the bits in threes, starting at the binary point : then convert each group of three to its octal equivalent (0s are added at each end, if necessary).

For instance, binary number 1101.10010 converts as follows :

1101.10010 →	001	101	·	100	100
	↓	↓		↓	↓
	1	5	·	4	4

Start at the binary point and, working both ways, separate the bits into group of three. When necessary, as in this case, add 0s to complete the outside group. Then convert each group of three into it binary equivalent therefore :

1101.10010 = 15.44

1.9 HEXADECIMAL NUMBERS

The Hexadecimal number most probably used extensively in microprocessor work. To begin with, they are much shorter than binary numbers. This makes then easy to write and remember.

The hexadecimal number system has a base of 16. Although any 16 digit may be used, everyone uses 0 to 9 and A to F as shown in table (a)

A
B
C
D
E
F

After reaching 9 in the hexadecimal system, you continue counting as A, B, C, D, E, F

TABLE (B) Decimal	Hexadecimal Binary	Digits Hexadecimal	
0	0000	0	
1	0001	1	
2	0010	2	
3	0011	3	
4	0100	4	
5	0101	5	
6	0110	6	
7	0111	7	
8	1000	8	
9	1001	9	
10	1010	A	→ TEN
11	1011	B	→ ELEVEN
12	1100	C	→ TWELVE
13	1101	D	→ THIRTEEN
14	1110	E	→ FOURTEEN
15	1111	F	→ FIFTEEN

(a) HEXADECIMAL-TO-BINARY CONVERSION

To convert a hexadecimal number to a binary number, convert each hexadecimal digit to its 4-bit equivalent using the code given in the table (b)

For instance, here's how 8 B C converts to binary :

8	B	C
↓	↓	↓
1000	1011	1100

As another example, F 2 A 4 converts like this,

F	2	A	4
↓	↓	↓	↓
1111	0010	1010	0100

(b) BINARY-TO-HEXADECIMAL CONVERSION

To convert in the opposite direction, from binary to hexadecimal, again use the code from the table (b). Here are two examples.

(i) Binary 1100 0101 converts as follows :

1100	0101
↓	↓
C	5

Another example

(ii) Binary 0110 1010 1110 1000 converts like this,

0110	1010	1110	1000
↓	↓	↓	↓
6	A	E	8

In both these conversion, we start with a binary number and wind up with the equivalent hexadecimal number.

(c) HEXADECIMAL-TO-DECIMAL CONVERSION

How do we convert hexadecimal numbers to decimal numbers? In the hexadecimal number system each digit position corresponds to a power of 16. The weight of digit position in a hexadecimal number as follows :

$$16^4 \ 16^3 \ 16^2 \ 16^1 \ 16^0 \ . \ 16^{-1} \ 16^{-2} \ 16^{-3} \ 16^{-4}$$

↑

Hexadecimal Point

Therefore, to convert from hexadecimal to decimal, multiply each hexadecimal digit by its weight and add the resulting products.

Here's an example. Hexadecimal F8E6.39 converts to decimal as follows :

F8E6.39 $F(16^3) + 8(16^2) + E(16^1) + 6(16^0) + 3(16^{-1}) + 9(16^{-2})$

$= 15(16^3) + 8(16^2) + 14(16^1) + 6(16^0) + 3(16^{-1}) + 9(16^{-2})$

$= 61{,}440 + 2048 + 224 + 6 + 0.1875 + 0.0352$

$= 63{,}718.2227$

(d) DECIMAL-TO-HEXADECIMAL CONVERSION

One way to convert from decimal to hexadecimal is the hex dabble. The idea is to divide successively by 16, writing down the remainders. Here's sample of how its done. To convert decimal 2479 to hexadecimal, the first division is

Final Answer is

16	2479	↑
16	154 - 15	9 10 15
	9 - 10	

9 → 9
10 → A
15 → F ↓

Other wise

15 → F ↑
10 → A
9 → 9

In this first division we get a quotient of 154 with a remainder of 15 (equivalent to F) In the second division, we get a quotient of 9 with a remainder of 10 (same as A). The final step is 15 → F, 10 → A & 9 → 9.

Hexadecimal 9AF is equivalent to decimal 2479.

1.10 COMPLEMENTS

TRUTH-TABLE

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

} Main

(a) 1's COMPLEMENT REPRESENTATION

The 1's complement of a binary number is the number that results when we complement each bit (Fig. 1.6) shows how to produce the 1's complement with logic circuits.

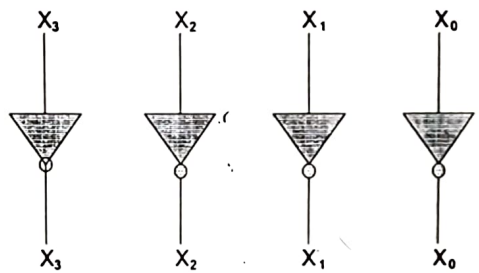


Fig. 1.6

Since each bit drives an inverter, the 4-bit output is the 1's complement of the 4-bit input. For instance of the output is

$$x_3 \quad x_2 \quad x_1 \quad x_0 = 0101$$

The 1's complement is

$$\overline{x}_3 \quad \overline{x}_2 \quad \overline{x}_1 \quad \overline{x}_0 = 1010$$

The same principle applies to binary numbers of any length, complement each bit to obtain the 1's complement. More example of 1's complement are

$$0101 \rightarrow 1010$$
$$1001 \quad 0100 \rightarrow 0110 \quad 1011$$
$$1010 \quad 0000 \quad 1100 \quad 0001 \rightarrow 0101 \quad 1111 \quad 0011 \quad 1110$$

(b) 2'S COMPLEMENT

There is a rather unusual number system that leads to the simplest logic circuits for performing arithmetic. Know s 2's complement representation, this system dominates microcomputer architecture and programming.

The 2's complement is the binary number that results when we add 1 to the 1's complement.

As a formula :

$$\text{2's complement} = \text{1's complement} + 1$$

For instance to find 2's complement of 1011, proceed like this :

$$1011 \rightarrow 0100 \quad \text{(1's complement)}$$
$$0100 + 1 = 0101 \quad \text{(2's complement)}$$

(Instead of adding 1, you can visualize the next reading on a binary Odometer) so, after obtaining the 1's complement 0100 ask yourself what comes next to binary odometer.

The answer is 0101.

Example of the 2's complements

Number	1's Complement	2's Complement
1101 0110 →	0010 1001	0010 1001 + 1

Answer is 0010 1010

Another example for 2's complement

Number	1's Complement	2's Complement
1000 0001 →	0111 1110 →	0111 1111
0011 0110 →	1100 1001 →	1100 1010

(c) 2's Complement Arithmetic

Before consider the 2's complement Arithmetic let us known about positive and negative number is binary numbers.

In other words you can take the 2's complement of a positive binary number

1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111
-8	-7	-6	-5	-4	-3	-2	-1	+0	+1	+2	+3	+4	+5	+6	+7

to find the corresponding negative binary number. For instance,

$$+5 \rightarrow 0101$$
$$-5 \rightarrow 1011$$

After taking the 2's complement of 0101, we get 1011 which represents -5. The principle also works in reverse :

$$-4 \hookrightarrow 1100$$
$$+4 \leftarrow 0100$$

After taking the 2's complement of 1100 we obtain, which represents +4.

What does the foregoing mean? It means that taking the 2's complement is equivalent to negation, changing the sign of the number. Why is this important? Because

its easy to build a logic circuits that produces the 2's complement. Whenever this circuits takes the 2's complement, the output is the negative of the input. This key idea lead to incredibly sample arithmetic circuit that can add and subtract.

In summary, here are the things to remember about 2's complement representation:-

Positive number always have a sign bit 0 and negative number always have a sing bit of 1.

- ☆ Positive numbers are stored in sign - magnitude form.
- ☆ Negative numbers are stored in 2's complements.
- ☆ Taking the 2's complement is equivalent to a sign change.

1.11 BINARY CODES

Electronic digital system use signals that have two distinct values and circuit elements that have two stable states. There is a direct 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.

A bit buy definition, is a binary digit, when used in conjunction with a binary code, it is better to thick of it as denoting a binary quantity equal to 0 or 1. A group of four distinct elements can be represented by a two-bit code, with each quantity assigned one of the following bit combination : 00, 01, 10, 11. A group of eight elements requires a three bit code, with each element assigned to one and only one the following: 000, 001, 010, 011, 100, 101, 110, 111.

(a) DECIMAL CODES

Binary codes for decimal digits require a minimum of four bits. Numerous different codes can be obtained by arranged four or more bits in 5 (five) distinct possible combinations.

A few possibilities are shown in table 1.1.

Table 1.1

Decimal Digit	(BCD) 8421	Excess-3	8 4-2-1	2 4 2 1	(BI QUI NARY) 50 4 3210
0	0000	0011	0000	0000	0100001
1	0001	0100	0111	0001	0100010
2	0010	0101	0110	0010	0100100
3	0011	0110	0101	0011	0101000
4	0100	0111	0100	0100	0110000

The BCD (Binary - Coded Decimal) is straight assignment of the binary equivalent. It is possible to assign weights to the binary bits according to their positions. The weight in the BCD code are 8, 4, 2, 1. The bit assignment 0011, for ex, can be interpreted by the weight to represent the decimal digit 6, because $0 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 = 3$. It is also possible to assign negative weight to a decimal code, as shown by the 8, 4, -2, -1 code. In this case the bit combination 0110 is interpreted as the decimal digit 2, as obtained from $0 \times 8 + 1 \times 4 + 1 \times (-2) + 0 \times (-1) = 2$. Two other weighted codes shown in the table are the 2421 and the 504 3210. A decimal code that has been used in some old computers is the excess -3 code. This is an unweighted code; its code assignment is obtained from the corresponding value of BCD after the addition of 3.

(b) THE EXCESS-3 CODE

The excess-3 code is an important 4-bit code sometimes used with binary-coded decimal (BCD) numbers. To convert any decimal numbers into its excess-3 form, add 3 to each decimal digit, and then convert the sum to a BCD number. For example, here is how to convert 12 to an excess-3 number. First, add 3 to each decimal digit.

1	2
+3	+3
4	5

Second, convert the sum to BCD form :

4	5	
↓	↓	{0100 0101) in the excess 3
0100	0101	code stands for decimal 12}

Another example

Convert 25 to an excess-3 number.

2	5		
+3	+3	5	8
5	8	↓	↓
		0101	1000

{0101 1000 in the excess 3 code stands for decimal 25}

(c) ERROR-DECTION CODES

An error-detection code can be used to detect error during transmission. The detected error cannot be corrected, but its presence is indicated.

A parity bit is an extra bit included with a message to make the total number of 1's either odd or even. A message of four bits and a parity bit, 'P' are shown is a table 1.2. In (I), P is chosen so that the sum of all 1's odd (in all five bits). In (ii) 'P' is chosen so that the sum of all 1's is even.

Table 1.2 Parity-bit generation

(I) Message	P (odd)	(Ii) Message	P(Even)
0000	1	0000	0
0001	0	0001	1
0010	0	0010	1
0011	1	0011	0
-	-	-	-
-	-	-	-
-	-	-	-
1111	1	1111	0

During transfer of information from one location to another, the parity bit is handled as follows :

In the sending end, the message (in this case the first four bits) is applied to a "Parity generation" network where the required 'P' bit is generated the message, including the parity bit is transformed to its destination. In the receiving end, all the incoming bits (in this case five) are applied to a "Parity Check" network to check the proper parity adopted. An error is detected if the checked parity does not correspond to the adopted one.

(d) REFLECTED CODE/GRAY CODE

Reflected code is nothing but gray code. The advantage of the reflected code over pure binary numbers is that a number is the reflected code changes by only are bit as it proceeds from one number to the next. To obtain a different reflected code, one can start with any bit combination and proceed to obtain the next bit combination by changing only one bit from 0 to 1 or 1 to 0 in any desired random fashion. The reflected code shown in table(a) is only one of many possible such codes.

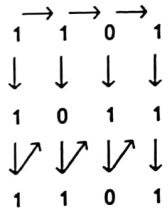
Reflected code	Decimal Equivalent
0000	0
0001	1
0011	2
0010	3
0110	4
0111	5
0101	6
0100	7
1100	8
1101	9
1111	10

1110	11
1010	12
1011	13
1001	14
1000	15

The reflected code is also known as the Gray codes.

An example for the Gray code to convert into Binary Code & Vice versa.

BINARY TO GRAY

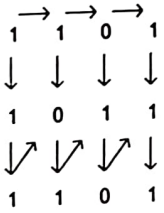


(ii) Gray to Binary

Another example

BINARY TO GRAY

(i)



(ii)

→ GRAY TO BINARY

(e) ALPHA NUMERIC CODE

To get information into and out of a computer, we need to use same kind of alphanumeric code (one for letters, numbers and other symbols). (At one time,

manufactures used their own alphanumeric codes, which led to all kind of confusion). Eventually, industry gelited on an input-output code known as the American standard code for information inter change (abbreviated ASCII). This code allows manufacturers to standardize computer hardware such as keyboard, printers - and video displays.

Another seven - and eight - bit alphanumeric codes such code is EBCDIC (Extended BCD interchange code).

Alphanumeric Character Codes

Character	7 bit ASCII Code	8 bit EBCDIC Code
A	100 0001	1100 0001
B	100 0010	1100 0010
C	100 0011	1100 0011
D	100 0100	1100 0100
-		
-		
-		
-		
0	100 1111	1101 0110
-		
-		
-		
-		
7	101 1010	1110 1001

Chapter - 2
BOOLEAN ALGEBRA

2.1 BASIC DEFENITIONS

Boolean Algebra, like and other deductive mathematical system, may be defined with a set of elements, a set of operation, and a number of unproved axioms or postulates.

A set of elementary is any collection of objects having a common property. If S is a set, and x and y are certain object then $x \in Es$ denotes that x is an element of the set S, and y s denotes that y is an not element of the set S.

The postulates of a mathematical system form the basic assumption from which it is possible to deduce the rules, theorems, and properties of the system. The most common axioms used to formulate various algebraic structure are:

1. **Closure**

A set `S' is closed with respect to a binary operator for every pair of elements of `S' the binary operator specifis a rule for obtaining a unique element of S.

For ex/. The set of natural numbers N = {1,2,3,4...} is closed with respect to the binary operator plus (+) by the rules of arithmetic addition, since for any $a,b, \in N$ we obtain a unique $C \in N$ by the operation a+b=C. The set of natural number is not closed with respect to the binary operation minus (-) by the rules of arithmetic subtraction because 2-3 = -1 and 2,3, $\in$ N which (-1) + N.

$\in$ (Belongs) ϵ

2. **Associate Law**

A binary operator • on a set `S' is said to be associative.

$$(x \bullet y) \bullet z = x \bullet (y \bullet z) \text{ for all } x, y, x \in S$$

3. **Commutative Law**

A binary operator • on a set `S' is said to be commutative whenever.

$$(x \bullet y) = y \bullet z \text{ for all } x, y, \in S$$

4. **Identity element**

A set `S' is said to have an identity element with respect to a binary operation • on `S' if there exists an element $e \in S$ with the property.

$$e \bullet x = x \bullet e = x \text{ for every } x \in S$$

ex/. The element O is an identity with respect to operation + on the set of integers

$$I = \{..., -3, -2, -1, 0, 1, 2, 3,...\} \text{ since}$$

$$x + 0 = 0 + x = x \text{ for any } x \in 1.$$

The set of Natural Number 'N' has no identity element of '0' is excluded from the set.

5. Inverse

A set 'S' having the identity element 'e' with respect to a binary operator • is said leave an inverse whenever ; for every $x \in S$ there crusts an element $y \in S$ such that :

$$x \bullet y = e$$

Example. In the set of integer I with e=0, the inverse of an element a is (-a) since

$$a + (-a) = 0.$$

6. Distributive Law

It • and are two binary operators on a set 'S' • is said to be distributive over whenever

$$x \bullet (y.z) = (x \bullet y) . (x \bullet z)$$

An ex./ Of an algebraic structures is field. A field is a set of elements, together with two binary operators, each having properties 1 to 5 and both operators combined to give property 6. The set of real numbers together with the binary operators + and form the field of real numbers is the basis for a arithmetic and ordinary algebra. The operators and postulates have the following meanings :

The binary operator + defines addition

The additive identity is 0.

The additive inverse defines subtraction

The binary operator circuit defines multiplication

The multiplication identity is 1.

The multiplicative inverse of a = 1/a division i.e., a. 1/a = 1.

The only distributive law applicable is that of over + :

$$A (b+c) = (a.b) + (a.c).$$

2.2 AXIOMATIC DEFENITION OF BOOLEAN ALGEBRA

The formal definition of Boolean algebra, we shall employ the postulates formulated by E.V. Huntington in 1904 these postulates have been used Boolean Algebra is an algebraic structure defined on a set of element B together with two binary operator + and. Provided the following postulates are satisfied.

1. (a) Closure with respect to the operator +
   (b) Closure with respect to the operator.
2. (a) An identity element with respect to +, designated by 0 : $x+ 0 = 0+x = x$.
3. (a) Commutative with respect to + : $x+y = y+x$
   (b) Commutative with respect to . : $x.y = y.x$
4. (a) . is distributive over + : $x.(y+z) = (x.y) + (x.z)$
   (b) + is distributive over . : $x+(y.z) = (x+y) . (x+z)$
5. For every element $x \in B$, there exists an element $x \in B$(called the complement of x) such that; (a) $x+x^1 = 1$ and (b) $x.x^1 = 0$.
6. There exists at least two elements x, $y \in B$ such that $x \neq y$.

Comparing Boolean algebra with arithmetic and ordinary algebra (the field of real numbers) we note the following differences :

1. Huntington postulates do not include the associate law. However, this law holds for Boolean algebra and can be derived (for both operators i.e.,+ and.) from the other postulates.
2. The distributive law of + over . i.e., $x+(y.z) = (x+y).(x+z)$ is valid for Boolean algebra, but not for ordinary algebra.
3. Boolean algebra does not have additive or multiplicative inverses : therefore, there are no subtraction or division operations.
4. Postulate 5 define un operator called complement which is not available in ordinary algebra.

5. Ordinary algebra deals with the real numbers, which constitute an infinite set of elements. B. but in the two-valued Boolean algebra defined below.
   1. The elements of the set B,
   2. The rules of operation for the Two binary operator and
   3. That the set of element B, together with the two operators, satisfied the six Huntington Postulates.

One can formulate many Boolean Algebras, depending on the choice of elements of B and the rules of operation. In out subsequent work, we deal with a two valued Boolean algebra, i.e., one with only two elements. Two-valued Boolean algebra has applications in set theory and in prepositional logic. Our interest here is with the applications of Boolean algebra to gate-type circuits.

(a) TWO-VALUED BOOLEAN ALGEBRA

A two-valued Boolean algebra is defined on a set of two elements, B = {0,1}, with rules for the two binary operators + and . as shown in the following operator tables.

(i) AND GATE TRUTH-TABLE

x	y	x . y
0	0	0
0	1	0
1	0	0
1	1	1

(ii) OR GATE TRUTH-TABLE

x	y	x + y
0	0	0
0	1	1
1	0	1
1	1	1

(ii) NOT-GATE TRUTH-TABLE

x	x'
0	1
1	0

These rules are exactly the same as the AND, OR and NOT operations, respectively, defined in the truth-table.

1. Closure is obvious from the tables since the result of each operations is either 1 or 0 and 1,0 ∈ B.
2. From the tables we see that :
   (a) 0 + 0 = 0 0 + 1 = 1 + 0 = 1
   (b) 1 . 1 = 1 1 . 0 = 0 . 1 = 0
   which establishes the two identity elements 0 for + and 1 for . as defined.
3. The commutative laws are obvious from the symmetry of the binary operator tables.
4. (a) The distributive law x.(y+z) = (x.y) + (x.z) can be shown to hold true from the operator tables by forming a truth table of all possible values of x, yand z. for each combination, we derive x.(y+z) and show that the value is the same as (x.y) + (x.z).

x	y	z	y+z	x.(y+z)	x.y	x.z	(x.y)+(x.z)
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

(b) The distributive law of + over . can be shown to hold true by means of a truth table similar to the one above.

5. From the complement table it is easily shown that :

(a) $x + x^1 = 1$, since $0 + 0^1 = 0 + 1$ and $1 + 1^1 = 1 + 0 = 1$.

(b) $x \cdot x^1 = 0$, since $0 \cdot 0^1 = 0 \cdot 1$ and $1 \cdot 1^1 = 1 \cdot 0 = 0$

which verified.

2.3 BOOLEAN FUNTIONS

A binary variable can take the value 0 or 1. A Boolean is an expression formed with binary variables, the two binary operators OR and AND, the unary operator NOT, parentheses, and equal sign.

The Boolean Function

$$F_1 = xyz^1 \quad \text{.....(2.1)}$$

The function F1 is equal to 1 if x = 1 and y = 1 and z1 = 1; otherwise F1 = 0. The above is an example of a Boolean function represented as an algebraic expression.

For ex/. = x y z

1 1 0

$F_1 = xyz^1$

$= 1.1.1 = 1.1 = 1.$

$F_1 = 1.$

Consider now the function :

$$F_2 = x + y^1z \quad \text{.....(2.2)}$$

The Boolean functions were appeared in the table. For ex/.

x y z

1 0 0

$F_2 = x + y^1 z.$

$1 + 1 .0$

$F_2 = 1 + 0 = 1.$

$F_2 = 1.$

Consider Third ex/.

x y z

1 1 0

$F_3 = x^1y^1z + x^1yz + xy^1$

$= 0.0.0 + 0.1.0 + 1.0$

$= 0 + 0 + 0$

$F_3 = 0.$

Consider another equation is : (fourth one)

x y z

1 1 1

$F_4 = xy^1 + x^1z$

$F_4 = xy^1 + x^1z$

$= 1.0 + 0.1$

$= 0 + 0$

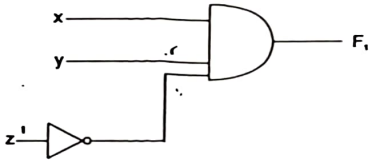
$F_4 = 0.$

The implementation of the four functions introduced is the logic diagram.

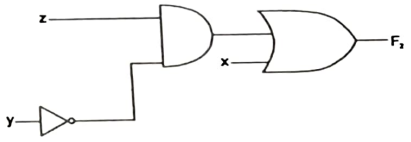
There is an AND gate fro each term in the expression, and an OR gate is used to combine two or more terms.

Let us consider the four Boolean function by the logic diagrams.

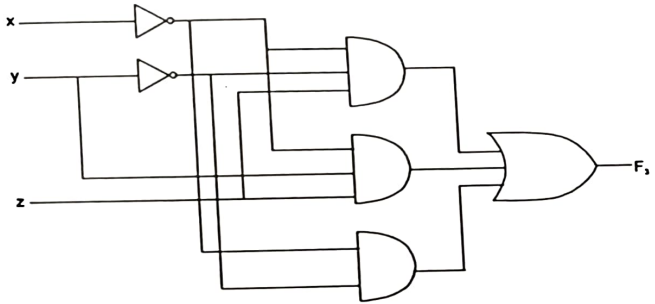
(i) $F_1 = xyz^1$



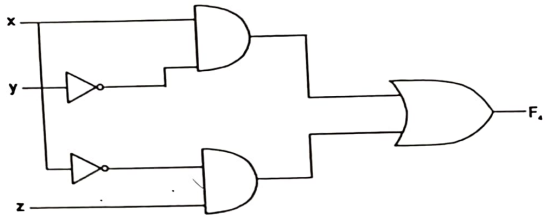
(ii) $F_2 = x+y^1z$



(iii) $F_3 = x^1y^1z = x^1 y z = xy^1$



(iv) $F_4 = xy^1 + x^1z$



2.4 CANONICAL AND STANDARD FORMS

Min terms and Max terms

A binary variable may appear either in its normal form (x) or its complement form (x^1). Now consist two binary variables x and y combined with an AND operation. Since each variable may appear in either form, there are four possible combinations:

i.e., : x^1y^1; x^1y; xy^1; xy,

Each of there four AND terms represents one of distinct areas in the min terms and max terms for three binary variables.

For AND Gate it consider in the Min term or a standard product.

x y z	Term	Designation
000	$x^1y^1z^1$	m_0
001	x^1y^1z	m_1
010	x^1yz^1	m_2
011	x^1yz	m_3
100	xy^1z^1	m_4
101	xy^1z	m_5
110	xyz^1	m_6
111	xyz	m_7

Fig. 2.1 Min terms

The above Fig. 2.1 shows of minterms. To forming an "OR" term, with each variable being primed or unprimed, provide possible combinations, called max terms or standard sums. The eight max terms for three variables, together with these symbolic designation are listed in the table.

Term	Designation
$x + y + z$	m_0
$x + y + z^I$	m_1
$x + y^I + z$	m_2
$x + y^I + z^I$	m_3
$x^I + y + z$	m_4
$x^I + y + z^I$	m_5
$x^I + y^I + z$	m_6
$x^I + y^I + z^I$	m_7

Fig. 2.2 Max terms

The above Fig. 2.2 show of Max term.

∴ Unprimed is the corresponding bit is a `0' and primed is the corresponding bit is a 1.

A Boolean function may be expressed algebraically from a given truth table by forming a min term for each combination of the variables which produces a1 in the function, and then taking the OR of all these terms.

For ex/. The function F_1 and F_2 were considered in the table of function of there variables.

x	y	z	Function F_1 Sum	Function F_2 Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Fig. 2.3

In Fig. 2.3 is determined by expressing the combinations 001, 100 and 111 as $x^Iy^Iz^Ixy^Iz^I$ and xyz^I respectively. Since each one of these min term results in $F_1 = 1$.

$F_1 = x^Iy^Iz + xy^Iz^I + xyz = m_1 + m_4 + m_7$,

similarly, it may be easily verified that :

$F_2 = x^Iy^Iz^I + x^Iyz^I + xyz^I + xy^Iz$ ++ x^Iyz

$= m_0 + m_2 + m_6 + m_5 + m_3$.

Now consider the complement of a Boolean function. It may be read from the truth table by forming a min term for each combination that produces a `O' in the function and then OR ing these terms.

$F_1 = x^Iy^Iz^I = x^Iyz^I = x^Iyz = xy^Iz = xyz^I$

If we take the complement of F_1, we obtain the function F_1 :

$F_1 = (x + y + z)\ (x + y_1 + z)\ (x^I + y + z^I)\ (x^I + y^I + z)$

$= m_0 . m_2 . m_3 . m_5 . m_6$

Similarly, it is possible to read the expression for F2 from the table :

$F_2 = (x + y + z)\ (x + y + z^I)\ (x + y^I + z)\ (x^I + y + z)$

$= m_0\ m_1\ m_2\ m_4$.

There are the important property of Boolean Algebra : Any Boolean function can be expressed as product of max terms (by "Product" is meant the AND ing of terms). (The procedure for obtaining the product of max terms directly from the truth table) the Boolean functions expressed as a sum of min term or product of max term are said to be in Canonical Form.

(a) SUM OF MIN TERM

Example 1 Express the Boolean functions $F = A+B^IC$ in a sum of min terms. The function has three variables A, B, C. The first term A is missing two variables; therefore:

$A = A(B+B^I) = AB+AB^I$

This still missing one variable :

$A = AB\ (C+C^I) + AB^I(C+C^I)$

$= ABC + ABC^I + AB^IC + AB^IC^I$

The second term B1C is missing one variable :

$$B^I C = B^I C(A + A^I) = AB^I C + A^I B^I C$$

Combining all terms, we have

$$F = A + B^I C$$
$$= ABC + ABC^I + AB^I C + AB^I C^I + AB^I C + A^I B^I C$$

But $AB^I C$ appears twice, and according to theorem 1 (x+x=x) it is possible to remove one of them. Rearranging the min terms in axending order. We finally obtain

$$F = A^I B^I C + AB^I C^I + AB^I C + ABC^I + ABC$$
$$= m_1 + m_4 + m_5 + m_6 + m_7$$

In shortation ∴ $F(A,B,C) = \Sigma (1,4,6,7)$

Σ - stands for summation symbol.

(b) PRODUCTION OF MAX TERMS

EXAMPLE : (ii) : Express the Boolean functions $F = AB+A^I C$ is a product of max term form. First convert the function into OR terms using the distributive law

$$F = AB + A^I C = (AB + A^I) (AB + C)$$
$$= (A + A^I) (B + A^I) (A + C) (B + C)$$
$$= (B) (A + C) (B + C)$$

The function has three variable : A, B and C. Each OR term is missing one variable; Therefore :

$$A^I + B = A^I + B + CC^I = (A^I+B+C) (A^I+B+C^I)$$
$$A + C = A + C + BB^I = (A+B+C) (A+B^I+C)$$
$$B + C = B + C + AA^I = (A+B+C) (A^I+B+C)$$

Combining all the terms and removing those that appear more than once, we finally obtain :

$$F = (A + B + C) (A + B^I + C) (A^I + B + C) (A^I + B^I + C)$$
$$= m_0\, m_0\, m_4\, m_5$$

A convenient way to express this function is as follows :

$$F (A,B,C) = \Pi (0,2,4,5)$$

Π, denotes the ANDing of max terms. The numbers are the max terms of the function.

(c) CONVERSION BETWEEN CANONICAL FORMS :

The complement of a function expressed as the sum of min terms equal the sum of min terms missing from the original function. This is because the original function is expressed by those min terms that make the function equal to 1. While its complement is a1 for those min terms that the function is a '0'. As an ex/. Consider the function:

$$F (A,B,C) = \Sigma (1.4.5.6.7)$$

This has a complement that can be expressed as :

$$F_1 (A,B,C) = \Sigma (0,2,3) = m_0+m_2+m_3.$$

Now if we take the complement of F1 by De Morgan's theorem, we obtain F in a different form :

$$F = (m_0 + m_0 + m_3)1 = m^I_0.m^I_0.m^I_3 = M_0M_2M_3 = \Pi (0,2,3)$$

(d) STANDARD FORMS

The Boolean algebra (express) will be provided by in Standard Form. In this configuration the term that form the functions may contain one, two or any number of literals. There are two types of standard Form :

(i) The sum of products

&

(ii) Product of Sums

(i) Sum of Product

The sum of product is a Boolean expression containing AND terms, called product terms, of one or more literals each. The sum denotes the OR ing of these terms. An example of a function expressed in sum of product is :

$$F_1 = B^I + AB + A^I BC^I.$$

The expression has three product terms of one, two and three literals each respectively their sum is in effect an OR operation.

The Product of Sums is a Boolean expression containing OR terms, called sum terms. Each term may have any number of literals. The product denotes the ANDing of these terms. An example of a function expressed in product of sum is :

$$A(B^I + C)\ (A^I + B + C^I + W)$$

$$F_2 = x\ (y^I + z)\ (x^I + y + z^I + w).$$

This expression has three sum terms of one, two and four literals each the product is an AND operation. The use of the words product and sum term from the similarity of AND operation to the arithmetic product (multiplication and the similarity of the OR operation to the arithmetic sum (addition).

A Boolean function may be expressed in a non standard form ex/.

$$F_3 = (AB + CD)\ (A^IB^I + C^ID)$$

and remove parentheses :

$$F_3 = A^IB^ICD + ABC^ID \text{ (Distributive Law)}$$

2.5 OTHER LOGIC OPERATIONS

The two binary operator 'AND' and 'OR' are placed two variable x and y they form two Boolean functions x.y and x+y respectively. The number of possible Boolean functions is 16. Therefore, the AND and OR functions one only two of a total of 16 possible function formed with two binary variables.

The truth-table for the 16 function formed with two binary variables x and y are listed in the table form.

Truth table for the 16 functions of two binary variable.

	Y			
		F_0	0000	
0	0	F_1	0001	
0	1	F_2	0010	/
1	0	F_3	0011	
1	1	F_4	0100	/
Operator		F_5	0101	
System		F_6	0110	$\oplus$
		F_7	0111	+
		F_8	1000	$\downarrow$
		F_9	1001	$\odot$
		F_{10}	1010	
		F_{11}	1011	$\subset$
		F_{12}	1100	
		F_{13}	1101	$\supset$
		F_{14}	1110	$\uparrow$
		F_{15}	1111	

The truth tables for the 16 functions, with two binary variables x and y the 16 columns F_0 to F_{15} represents a truth table of are possible function for the two given variable x and y.

The functions are determined from the 16 binary combinations that can be assigned to F. Some of the functions are shown with an operator symbol. For ex/. F_1 represents

the truth table for AND ad F_7 represents the truth table for 'OR'. The operator symbols for these functions are (-) and (+)s respectively.

The 16 functions listed in the truth table form can be expressed algebraically by means of Boolean expressions.

Boolean Functions	Operator Symbol
$F_0 = 0$	———
$F_1 = xy$	x . y
$F_2 = xy^I$	x / y
$F_3 = x$	———
$F_4 = x^Iy$	y / x
$F_5 = y$	———
$F_6 = xy^I + x^Iy$	x + y
$F_7 = x + y$	x + y
$F_8 = (x + y)^I$	x ↓ y
$F_9 = x\ y + x^Iy^I$	x • y
$F_{10} = y^I$	y^I
$F_{11} = x+y^I$	x cy
$F_{12} = x^I$	x^I
$F_{13} = x^I+y$	x ⊃ y
$F_{14} = (xy)^I$	x ↑ y
$F_{15} = 1$	———

		Name	Comments
F_0	→	Null	Binary constant 0
F_1	→	And	x any y
F_2	→	Inhibition	x but not y
F_3	→	Transfer	x
F_4	→	Inhibition	y but not x
F_5	→	Transfer	y
F_6	→	Exclusive-OR	x or y but not both
F_7	→	OR	x or y
F_8	→	NOR	Not-OR
F_9	→	Equivalent*	x equals y
F_{10}	→	Complement	Not y
F_{11}	→	Implication	If y then x
F_{12}	→	Complement	Not x
F_{13}	→	Implication If	y then x
F_{14}	→	x Ty NAND	Not-And
F_{15}	→	Identity	Binary Constant 1.

The operator symbol are listed in the second column of table form.. All the new symbols shown except for the exclusive-OR symbol ⊕ are not in common use by digital designers.

The 16 functions listed can be subdivided into three categories :

1. Two functions that produce a constant 0 or 1.
2. Four functions with unary operation complement and transfer.
3. Ten functions with binary operators that define eight different operations AND, OR, NAND, NOR exclusive-OR, equivalence* (in also known as equality, coincidence, and exclusive-NOR) inhibition and implication.

Any function can be equal to a constant but a binary function can be equal to only 1 or 0.

- The complement unction produces the complement of each of the binary variables.
- A function which is equal to use input variable has been give the name transfer because the variable x or y transformed through the gate that terms the function without changing its value.
- The eight binary operators, two inhibition and implication are used by logicians but are seldom used in computer logic.
- The AND and OR operator have been mentioned in conjunction with Boolean algebra.
- The other four functions are extensively used in the designs of digital systems. The NOR function is the complement of the OR function and its name is an abbreviation of not-OR, similarly, NAND is the complement of AND and is an abbreviation of not-AND. The exclusive-OR, abbreviated x OR a E OR is similar to OR.

The exclusive-OR and equivalence functions are the complements of each other this can be easily verified by inspecting table. The truth table for the exclusive-OR is F6 and for the equivalence is F9 and these functions in the complement of each other. For this reason the equivalence function is often called exclusive NOR i.e. exclusive-OR.NOT.

2.6 DIGITAL LOGIC GATES

Boolean function are expressed in terms of AND, OR and NOT operations, it is easier to implement a Boolean function with these types of gates.

The Graphics symbols and truth table of the eight gates are shown in fig (a),

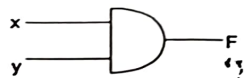
- AND GATE
- OR GATE
- INVERTER
- BUFFER
- NAND

- NOR
- EXCLUSIVE-OR(X OR)
- EXCLUSIVE-NOR (EQUIVALENCE)

Each gate has one or two (0 or 2) binary input variables designated by x and y and one binary output variable designated by F.

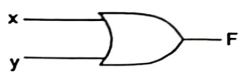
DIGITAL LOGIC GATES

BLOCK DIAGRAMS AND TRUTH TABLE

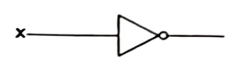
Name	Graphic Symbols	Algebraic Function	T. T.
And		F = x.y	

0

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

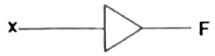
OR  F = x + y

x	y	f
0	0	0
0	1	1
1	0	1
1	1	1

Inverter  F = x^1

x	f
0	1
1	0

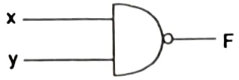
✚ Buffer



$F = x$

x	f
0	0
1	1

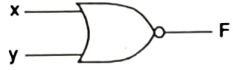
✚ Nand



$F = (xy)^{I}$

x	y	f
0	0	1
0	1	1
1	0	1
1	1	0

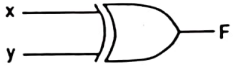
✚ Nor



$F = (x+y)^{I}$

x	y	f
0	0	1
0	1	0
1	0	0
1	1	0

✚ Exclusive-OR (X OR)



$F = xy^{I} + x^{I}y$

$= x + y$

x	y	f
0	0	0
0	1	1
1	0	1
1	1	0

✚ Exclusive - Nor (or) Equivalence

$F = xy + x^{I}y^{I}$

$= x \odot y$

x	y	f
0	0	1
0	1	0
1	0	0
1	1	1

The AND, OR and inverter circuits were defined and the inverter circuit inverts the logic sense of a binary variables. It produces the NOT, or complement, function. The small circle in the output of the graphics symbol of an inverter designates the logic complement. The triangle symbol by its self designates a buffer circuit. A buffer produces the transfer function. But does not produce any particular logic operation, since the binary value of the output is equal to the binary value of the input.

The NAND function is the complement of the AND function, as indicated by a graphic symbol which consists of an AND graphic symbol followed by a small circle. The NAND and NOR gates are extensively used as standard logic gates and are in fact far more popular than the AND and OR gates. NAND and NOR gates are easily constructed with transistor circuits. The exclusive-OR gate has a graphic symbol similar to that of the OR gate except for the additional curved line on the input side. The equivalence, or exclusive-NOR gate is the complement of the exclusive-OR, as indicated by the small circle on the outside of the graphic symbol.

2.7 IC DIGITAL LOGIC FAMILIES

Digital IC gates are classified not only by their logic operation, but also by the specific logic-circuit family to which they belong. Each logic family has its own basic electronic circuit upon which mere complex digital circuits and functions are developed. The basic circuit in each family is either a NAND or a NOR gate.

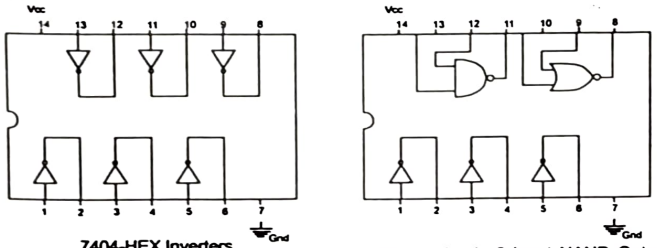
Many different logic families of digital IC's have been introduced commercially,

- ✚ TTL Transistor-Transistor Logic
- ✚ ECL Emitter-Coupled Logic
- ✚ MOS Metal-Oxide Semiconductor
- ✚ CMOS Complementary Metal-Oxide Semiconductor
- ✚ I^2L (IIL) Integrated-Injection Logic
- ✚ TTL has an extensive list of digital function and is currently the most popular logic family.

TTL IC's are usually distinguished by numerical designation as the 5400 and 7400 series. (The former has a wide operating temperature range, suitable for military use, and the latter has a narrower temperature range, suitable for industrial use).

The numeric designation of the 7400 series means that IC package are numbered as 7400, 7401, 7402 etc.,

The Fig. 2.4 shows two TTL circuits.



7404-HEX Inverters

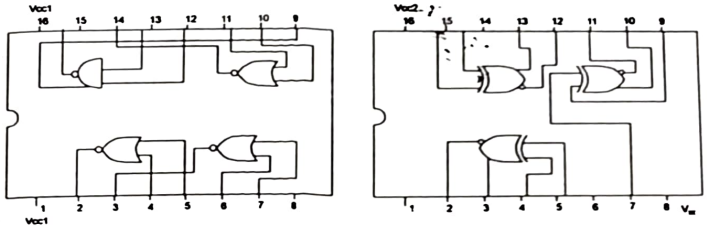
7400-Quadruple 2 input NAND Gates

Fig. 2.4 TTL GATES

- The 7404 provides six (hex) inverters in a package.
- The 7400 provides four (quadruple) 2 - input NAND gate
- The terminals marked Vcc and GND are the power supply pens which require a voltage of 5 volts for proper operation.
- ECL → Emitter Coupled Logic

ECL is used in systems requiring high - speed operations.

The most common ECL type is designated as the 10,000 series. Fig. 2.5 shows two ECL circuits.



10102-Quadruple 2 - input NOR gates

10107-Triple exclusive - OR/NOR gates

Fig 2.5 ECL GATES

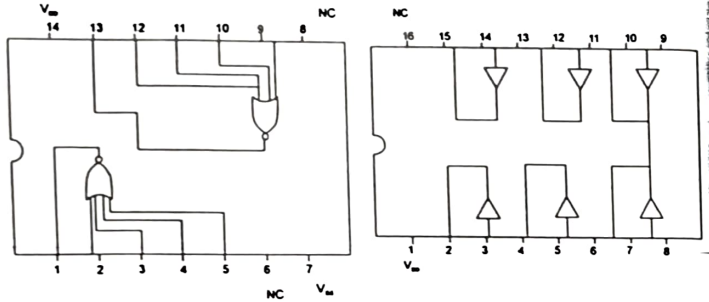
The 10102 provides four 2-input NOR gates. Note that an ECL gate may have two outputs, one or the NOR functions and another for the OR functions (Pin 9 of the 10102 IC). The 10107 IC provides three exclusive-OR gates. Here again there are outputs from each gate; the other output gives the exclusive-NOR function or equivalence. ECL gates have three terminals for power supply. V_{cc1} and V_{cc2} are usually connected to ground, and V_{EE} to a -5.2-volt supply.

- MOS → Metal-Oxide Semiconductor + I^2L Integrated-Injection Logic

  MOS and I^2L are used in circuits requiring high component density. The high density were provided with which transistors can be fabricated in MOS and I^2L.

- CMOS → Complementary Metal-Oxide Semiconductor

  CMOS is used in systems requiring low power consumption. CMOS circuits of the 4000 series are shown in Fig. 2.6.



4002-Dual 4 - input Nor gates 4050 - Hex buffers

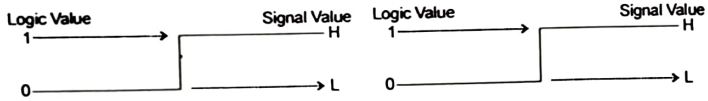
Fig. 2.6 CMOS Gates

Only two 4-input NOR gates can be accommodated in 4002 because of pin limitation. The 4050 type provider six buffer gates. Both ICs have two unused terminals marked NC (No connection). The terminal marked VDD requires a power supply voltage from 3 to 15 volts, while VSS is usually connected to ground.

(a) POSITIVE AND NEGATIVE LOGIC

The binary signal at the inputs or output of any gate can have one of two values except during transition. One signal value represents logic 1 and the other, logic-0. Since two signal values are unsigned to two logic values.

The two values of a binary signal as shown in Fig. 2.7.



(a) Positive Logic (b) Negative Logic

Fig. 2.7 Signal amplitude assignment and type of Logic

One value must be higher than the other since the two values must be different in order to distinguish between them. The 'H' designate the higher level and the 'L' designate the lower level.

There are two choice for Logic value assignment. The high-level 'H' to represent Logic '1' as shown in Fig. 2.7a defines a positive - Logic system and selecting the Low-Level 'L' to represent Logic-1 as shown in Fig. 2.7b defines a negative logic system. The terms positive and negative are same what misleading since both signal value may be positive or both may be negative.

Integrated circuits data sheets define digital functions not in term of Logic-1 or Logic-0, but rather in terms of H and L levels, were listed high and low level voltages for IC Logic formulas in truth table H and L Level in IC Logic formulas.

High Level Range	voltage (v) Typical	Low Level Range	voltage (v) Typical
2.4-5	3.5	0-0.4	0.2
-0.95- -0.7	-0.8	-1.9- -1.6	-1.8
		0-0.5	

(b) SPECIAL CHARACTERISTICS

The characteristics of IC digital logic families are usually compared by analyzing the circuit of the basic gate in each family.

We first explain the properties of these parameters and then use them to compare the IC Logic families.

- **Loading** specifies the number of standard loads that the output of a gate can drive without impairing its normal operation. A standard load is usually defined as the amount of currents needed by an input of another gate in the same IC family. Sometimes the term loading is called as Fan-out.
- The **Fan-out** is the maximum number of inputs that can be connected to the output of a gate and it is expressed by a number.
- **Power** dissipation is the supplied power required to operate the gate. The number that represents this parameter does not include the power delivered

from another gate, it represents the power delivered to the gate from the power supply.

- **Propagation delay** is the average transition delay time for a signal to propagate from input to output when the binary signals change in value, the signals through a gate take a certain amount of time to propagate from the inputs to the outputs.
- **Noise Margin** is the maximum noise voltage added to the input signal of a digital circuit that does not cause an undesirable change in the circuit output. There are two types of noise to be considered. (i) DC Noise is caused by a drift in the voltage levels of a signals. (ii) AC noise is a random pulse that may be created by other switching signals. Thus, noise is a term use to denote an undesirable signal that is the normal operating signal. Noise margin is express in volts(v) and represents the maximum noise signal that can be tolerated by the gate.

2.8 SEMI CONDUCTOR MEMORIES

(a) BIPOLAR ROM's

A Read-Only Memory (ROM) is the simplest kind of memory. we can read the word in any memory location ("Read" means to make the contents of the memory location appear at the output terminals of the ROM)

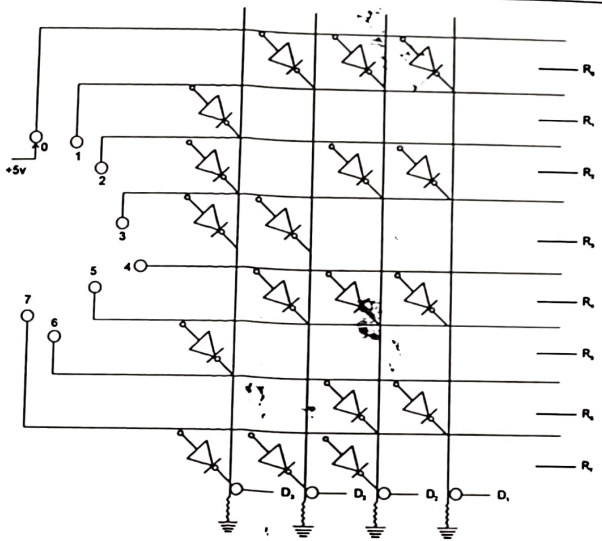


Fig. 2.8

Fig. 2.8 shows one way to build a ROM.

Each horizontal row is a register or memory location. The R0 register contains three diodes, the R1 register has one diode and so on.

The output of the ROM is the word

$$D = D_3D_2D_1D_0$$

In switch position 0, a high voltage terms on the diodes in the R0 register; all other diodes are off. This means that a high output appears at D2,D1 and D0. Therefore, the word stored at memory location 0 is,

$$D = 0111$$

What happens if the switch is moved to position 1? The diode in the R1 register conducts forcing D3 to go high. Because all other diodes are off, the output from the ROM becomes,

D = 1000.

The address and contents of a memory location are two things. As shown in table.

Register	Address	Word
R0	0	0111
R1	1	1000
R2	2	1011
R3	3	1100
R4	4	0110
R5	5	1001
R6	6	0011
R7	7	1110

(b) PROMS AND EPROMS

A programmable ROM (PROM) is different. It allows the user to store the data. An instrument called a PROM. PROM programmer, the user can burn in the program and data. Once this has been done, the programming is permanent. In other words, the stored contents cannot be erased. The Fig. 2.9 shows PROM and EPROM.

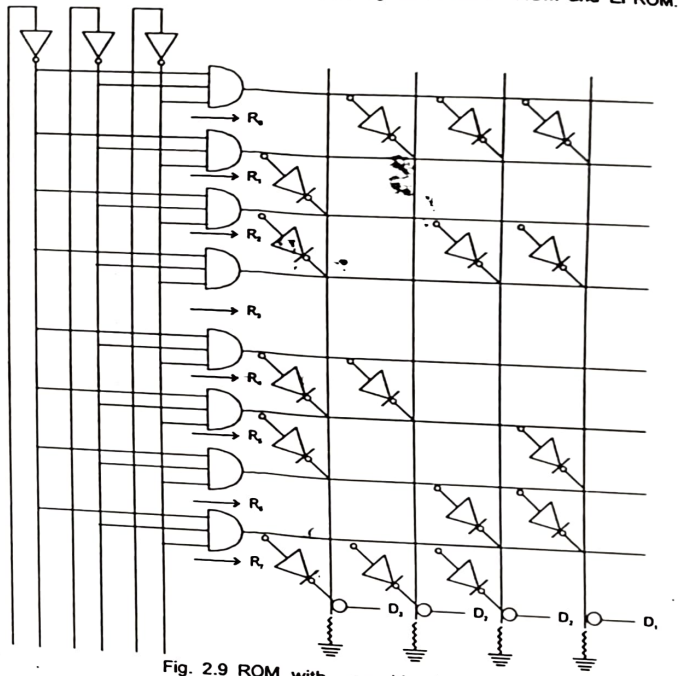


Fig. 2.9 ROM with - on chip decoding

THE ERASABLE PROM (EPROM) uses MOSFET'S:

Data is stored with a PROM programmer. Later data can be erased with ultraviolet light. The light passes through a window in the IC package to the chip, where it releases stored charges. The effect is to wipe out the stored contents. In other words, the EPROM is Ultraviolet-Light-erasable and electrically reprogrammable.

The EPROM is helpful in design and development. The user can erase and store until the program and data are perfected. Then the program and data can be sent to an IC manufactures who make for mass production. The Fig. 2.10 shows about Three - state ROM.

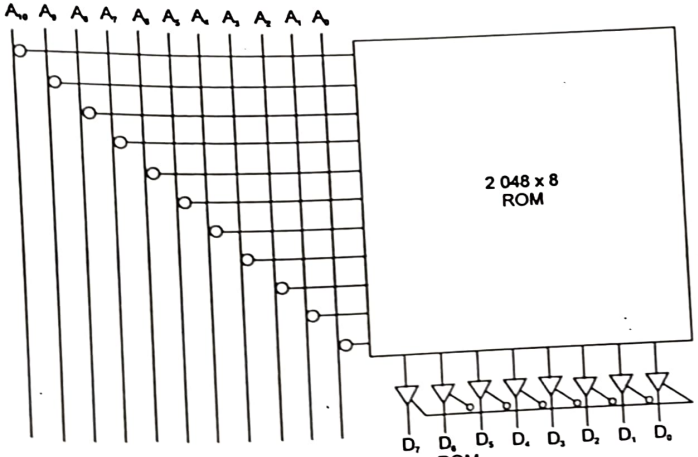


Fig. 2.10 Three-state ROM.

(c) BIPOLAR RAM's

A Random - Access Memory (RAM), or a read-write memory, is equal to a group of addressable register. After providing an address, you can read the stored contents of the memory location or write new contents into the memory location.

✦ Semiconductor RAMS

Semiconductor RAMS may be static or dynamic. The static RAM uses bipolar or MOS flip-flops; data is retained indefinitely as long as power is applied to the flip-flops.

On the other hand, a dynamic RAM uses MOSFETS and capacitor that store data. Because the capacitor charge leaks off, the stored data must be refreshed every few milliseconds. In either case, RAMS are volatile; turn off the power and you lose the stored data.

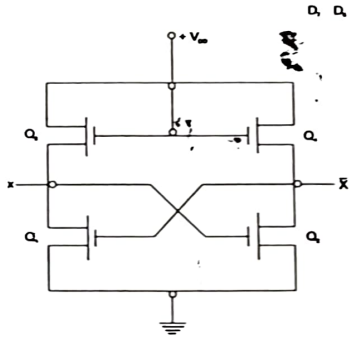


Fig. 2.11 Static Cell

The Fig. 2.11 shows one of the flip-flops used in a static MOS RAM. Q_1 and Q_2 act like switches. Q_3 and Q_4 are active loads, meaning that they behave like restores.

Either Q_1 conduct and Q_2 is cut off or vice versa. A static RAM will contain thousands of flip-flops like this, one for each stored bit. As long as power is applied, the flip-flop remain latched and can store the bit indefinitely.

DYNAMIC RAM

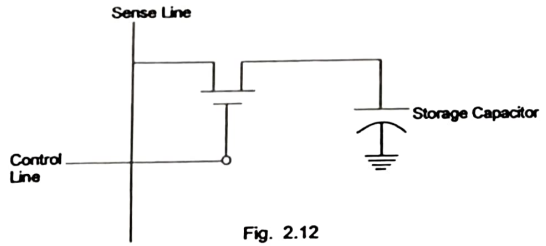


Fig. 2.12

Fig. 2.12 shows one of the memory elements (called cells) in a dynamic RAM. When the sense and control lines go high, the MOSFET conducts and charges the capacitor. When the sense and control lines go low, the MOSFET opens and the capacitor retains its charge.

A dynamic RAM may certain thousands of memory cells, sense only a single MOSFET and capacitor are needed. A dynamic RAM has more memory locations than a static RAM of the same physical size.

The disadvantages if the dynamic RAM is the need to refresh the capacitor charge every few milliseconds. This complicates the design problem because more circuitry is needed.

2.9 SIMPLIFICATION OF BOOLEAN FUNCTIONS :

(a) THE MAP METHOD

The map method provides a simple straight forward procedure for minimizing Boolean Functions. This method may be regarded either as a pictorial form of a Truth-Table or a an extension of the Venn diagram.

The map method, first proposed by Veitch (i) and slightly modified by Karnaugh (ii) is also known as the "Veitch diagram or the "Karnaugh map".

In the karnaugh map were provided in two, three and four variables map which were considered with truth-table.

KARNAUGH MAPS

The Engineers and Technicians don't simplify equations with Boolean algebra. Instead, they use a method based on Karnaugh map.

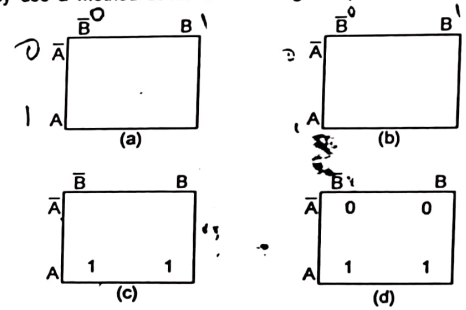


Fig. 2.13 Two-Variable Karnaugh map

NOTE

$A = 1$

$\bar{A} = 0$

$B = 1$

$\bar{B} = 0$

A	B	Y
0	0	0
0	1	0
1	0	1
1	1	1

Here we have a Truth-Table (i) and (ii) - Here's how to construct the Karnaugh map. First begin by drawing Fig. 2.13a. Note the order of the variable and their complements; the vertical column has A followed $\bar{A}$, and the horizontal row has $\bar{B}$ followed by B.

Secondly, look for output 1's in truth-table (i). The first 1 output to appear is for the input of A = 1 and B = 0. The fundamental product for this is $A\bar{B}$. Now,

enter a 1 on the Karnaugh map as shown in Fig. 2.13b. This 1 represents the product $A\overline{B}$ because the 1 is in the A row and the $\overline{B}$ column.

Similarly, Table (i) has an output 1 appearing for an input of A = 1 and B = 1. The fundamental product for this is AB. When you enter a on the karnaugh map to represent AB, you get the map of Fig. 2.13c.

The final step in the construction of the Karnaugh map is to enter 0s in the remaining spares. Fig. 2.13d shown how the karnaugh map look in its Final Form.

(c) THREE-VARIABLE MAP :

TRUTH - TABLE

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Here we have a truth table for three variable map. Begin by drawing Fig. 2.9 i.e., In the Fig. 2.14a. It is especially important to notice the order of the variables

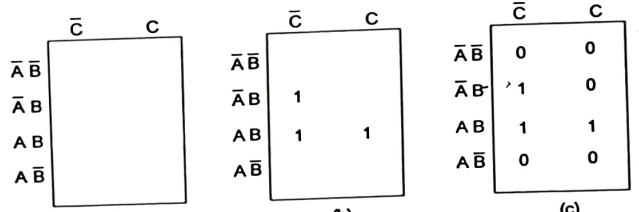


Fig. 2.14

and their complements. The vertical column is labeled $\overline{A}\,\overline{B}$, $\overline{A}B$, AB & $A\overline{B}$. This order is not a binary progression; instead 'A' follows the order & 00, 01, 11 and 01.

Note : A = 1, $\overline{A}$ = 0, B = 1 & $\overline{B}$ = 0, C = 1, $\overline{C}$ = 0.

In the truth table output 1's in the fundamental products for these 1 output are $A\overline{B}\,\overline{C}$, $AB\,\overline{C}$ and ABC. Enter these 1s on the Karnaugh map (Fig. 2.14b). The final step is to enter 0s in the remaining spares (Fig. 2.14c).

(d) FOUR-VARIABLE MAP

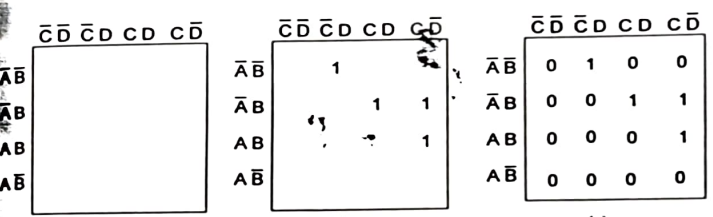


Fig. 2.15 Four-Variable Karnaugh map

Here's an example of constructing a four variable map. The truth table for the variable Karnaugh map.

TRUTH-TABLE

	A	B	C	D	Y
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	1
8	0	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	0

NOTE : A=1, $\overline{A}$ =0 B=1 $\overline{B}$ =0, C=1, $\overline{C}$ =0, D=1 $\overline{D}$ =0

The first step is to draw the blank map Fig. 2.15a. Again notice the progression the vertical column is labeled $\overline{A}\,\overline{B}$, $\overline{A}$ B, AB and A $\overline{B}$. The horizontal row is labeled $\overline{C}\,\overline{D}$, $\overline{C}$D, CD and C$\overline{D}$.

In the Truth-Table the output is have these fundamental products : $\overline{A}\ \overline{B}\ \overline{C}\ D$, $\overline{A}$ B C D, $\overline{A}$ B C D and A B C $\overline{D}$. After entering 1s on the karnaugh map you will have Fig. 2.15b. The final step of filling in 0s results in the completed map of Fig. 2.15c.

(e) PAIRS ON A KARNAUGH MAP

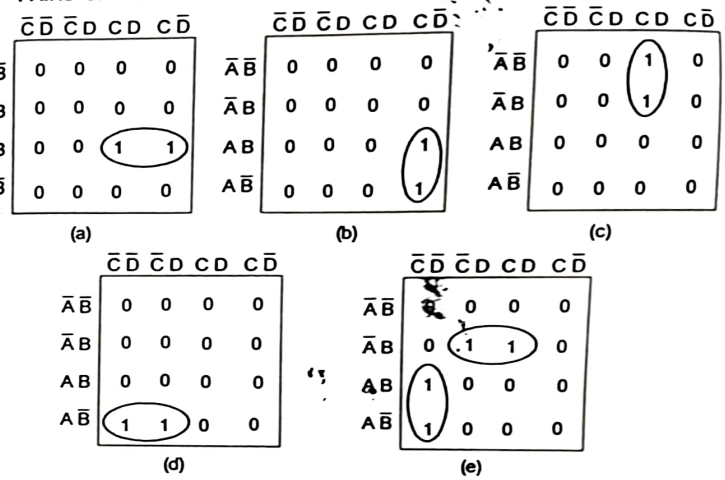


Fig. 2.16 Pairs on a Karnaugh Map

Rules → $\overline{A} + A = 0$, $\overline{B} + B = 0$, $\overline{C} + C = 0$, $\overline{D} + D = 0$ Or $A + \overline{A} = 0$, $B + \overline{B} = 0$, $C + \overline{C} = 0$, $D + \overline{D} = 0$

PAIRS

The map of Fig. 2.16a contains a pair of 1s that are horizontally adjacent. The first 1 represents the product ABCD. The second 1 stands for the product ABCD. For easy identification, it is customary to encircle a pair of adjacent 1s as shown in Fig. 2.16a, then, when you look at the map, you can tell at a glance that one variable and its complement will drop out of the Boolean equation.

Algebraic Proof

The sum-of-products equation corresponding to Fig. 2.16a is

$$Y = ABCD + ABC\overline{D}$$

Which factories into

$$Y = ABC(D + \overline{D})$$

Since D is ORed with $\overline{D}$, the equation reduces to Y = A B C

Another example. Fig. 2.16b shows a pair of 1s that are vertically adjacent. These 1s correspond to the product $ABC\overline{D}$ and $A\overline{B}C\overline{D}$.

Algebraic Proof

$$Y = ABC\overline{D} + A\overline{B}C\overline{D}$$

which factors into $Y = AC\overline{D}\ (B + \overline{B})$

$$Y = AC\overline{D}$$

A glance at Fig. 2.16c indicates that B charges form, therefore, in the Algebraic Proof for Fig. 2.16e.

Algebraic Proof

$$Y = \overline{A}\,\overline{B}CD + \overline{A}BCD$$

which factors into

$$Y = \overline{A}CD\ (\overline{B} + B)$$

$$Y = \overline{A}CD$$

A glance at Fig. 2.16d indicates that the pair 1 stands for $A\overline{B}\,\overline{C}\,\overline{D}$ and $A\overline{B}\,\overline{C}\,\overline{D}$ which is the form of Algebraic Proof.

Algebraic Proof

$$Y = A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}D$$

which factors into

$$Y = A\overline{B}\,\overline{C}\ (\overline{D} + D)$$

$$Y = A\overline{B}\,\overline{C}$$

For instance the final one pair of Fig. 2.16e represents upper or lower pair (two pairs were shown in the Fig. 2.16e).

Upper Pair =

$$Y = \overline{A}\,B\,\overline{C}\,D + \overline{A}\,B\,C\,D$$

$$Y = \overline{A}\,B\,D\,(\overline{C} + C)$$

$$Y = \overline{A}\,B\,D$$

Lower Pair =

$$Y = A\,B\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}\,\overline{D}$$

$$Y = A\,\overline{C}\,\overline{D}\ (B + \overline{B})$$

$$Y = A\,\overline{C}\,\overline{D}$$

$$\therefore\quad Y = \overline{A}\,B\,D + A\,\overline{C}\,\overline{D}$$

$$Y = B\,\overline{C}\ (A+\overline{A})\ +(D+\overline{D})$$

$$\therefore\quad Y = B\,\overline{C}$$

(f) THE QUAD ON KARNAUGH MAP

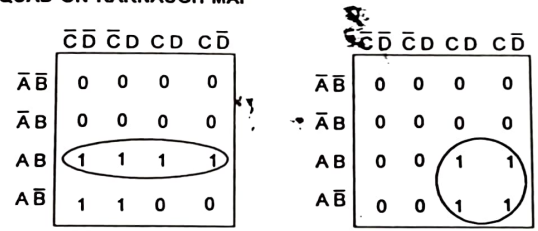


Fig 2.17 Quad on a Karnaugh Map

Quad is a group of four is that are end to end, as shown in the form of a square as shown in Fig. 2.17.

When you see a quad always encircle it because it leads to a simpler product. In fact, a quad means that two variables and their complement drop out of the Boolean equation.

Here's Fig. 2.17a shows a quad, the equation is

$$Y = AB\overline{C}\,\overline{D} + AB\overline{C}\,D + ABCD + ABC\overline{D}$$

$$Y = A\,B\,(\overline{C} + C)\,(\overline{D} + D)$$

$$Y = AB$$

Here's Fig. 2.17b shows a quad, the equation is

$$Y = A\,B\,C\,D + A\,B\,C\,\overline{D} + A\,\overline{B}\,C\,D + A\,\overline{B}\,C\,\overline{D}$$
$$Y = A\,C\,(B + \overline{B} + D + \overline{D})$$
$$Y = A\,C$$

(g) THE OCTET ON A KARNAUGH MAP

	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	0	0	0	0
$\overline{A}B$	0	0	0	0
AB	1	1	1	1
$A\overline{B}$	1	1	1	1

Fig. 2.18 Octet on a Karnaugh map

An octet is a group of eight adjacent 1s like these of Fig. 2.18. An octet always eliminates three variables and their complement.

(i) $Y = AB\overline{C}\,\overline{D} + AB\overline{C}D + ABCD + ABC\overline{D}$

$Y = AB\,(\overline{C} + C + \overline{C} + C + \overline{D} + D + D + \overline{D})$

$Y = AB$

(ii) $Y = A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}D + A\overline{B}\,CD + A\overline{B}\,C\overline{D}$

$Y = A\overline{B}\,(\overline{C} + C + \overline{C} + C + \overline{D} + D + D + \overline{D})$

$Y = A\overline{B}$

Note : $\overline{C} + C$, $\overline{D} + D = 0$.

Rules : $\overline{A} + \overline{A} = \overline{A}$, $\overline{B} + \overline{B} = \overline{B}$, $\overline{C} + \overline{C} = \overline{C}$, $\overline{D} + \overline{D} = \overline{D}$

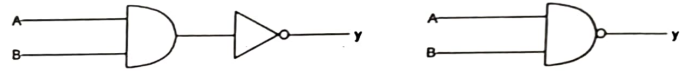
2.10 NAND AND NOR IMPLEMENTATION

(a) NAND GATES

The NAND gate has two or more input signals but only one output signal. All input signals must be high to get a low output.

TWO-INPUT GATE

Fig 2.19a shows the logical structure of NAND gate, and AND gate followed by an inverter. Therefore, the final output is NOT the AND of the inputs. Originally called a NOT-AND gate, the circuit is now referred to as a NAND gate.



(a) Logical Meaning (b) Standard Symbol

Fig. 2.19 NAND Gates

Fig. 2.19b shows the standard symbol for a NAND gate. The inverter triangle has been deleted and the bubble moved to the AND gate output. If one or more inputs are low, the result of ANDing is to low; Therefore, the final inverted output is high. Only when all inputs are high does the ANDing produce a high signal; then the final output is low.

TRUTH-TABLE

TWO-INPUT NAND GATE

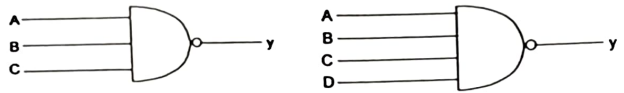
A	B	$\overline{A}\,\overline{B}$
0	0	1
0	1	1
1	0	1
1	1	0

Truth-table summarize the action of a 2-input NAND gate. As shown, the NAND gate recognizes any input word with one or more 0s, that is, one or more low inputs produce a high output. The Boolean equation for a 2-input NAND gate to

$$Y = \overline{A}\,\overline{B}$$

Read this as "Y equals NOT AB". If you use this equation, remember that the ANDing is done first then the inversion.

THREE-INPUT GATE



(a) 3-Input (b) 4-Input

Fig 2.20 NAND Gates

Fig. 2.20a shows a 3-input NAND gate. The inputs are ANDed, and the product is inverted. Therefore, the Boolean equation is

$$Y = \overline{ABC}$$

Here is the analysis of Fig. 2.20a. If one or more inputs are Low, the result of ANDing is Low; therefore, the final output is high. If all inputs are high, the ANDing gives a high signal; so the final output is Low.

TRUTH-TABLE

THREE INPUT NAND GATE

A	B	C	ABC
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Truth table for a 3-input NAND gate. As indicated, the circuit recognizes words with one or more 0s. This means that one or more low inputs produce a high output.

D FOUR-INPUT GATE

Fig. 2.20b is the symbol for a 4-input NAND gate. The inputs are ANDed, and the result os inverted. Therefore, the Boolean equation is

$$Y = \overline{ABCD}$$

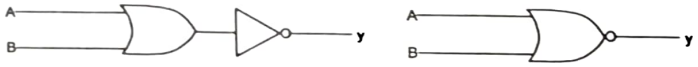
If you construct the truth table, you will have input words from 0000 to 1111. All words from 0000 through 1110 produce a 1 output; only the word 1111 gives a 0 output.

TRUTH-TABLE

A	B	C	D	ABCD
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

(b) NOR GATES IMPLEMENTATION

The NOR gate has two or more inputs signals but only one output signal. All input must be low to get a high output. In other words, the NOR gate recognizes only the input word whose bits are all 0s.



(a) Logical Meaning (b) Standard Symbol

Fig. 2.21 NOR Gate

TWO-INPUT GATE

Fig. 2.21a shows the logical structure of a NOR gate, which is an OR gate followed by an inverter, therefore, the final output is NOT the OR of the inputs. Originally called NOT-OR gate, the circuit is now referred to a NOR gate.

Fig. 2.21b shows the standard symbol for a NOR gate. Notice that the inverter triangle has been deleted and the small circle or bubble moved to the OR-gate output. The bubble is a remainder of the inversion that follows the ORing.

TRUTH-TABLE

TWO-INPUT NOR GATE

A	B	$\overline{A}+\overline{B}$
0	0	1
0	1	0
1	0	0
1	1	0

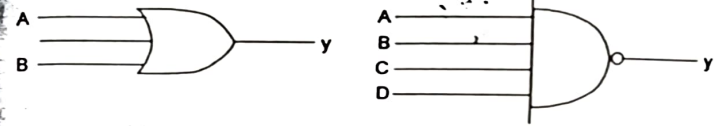
Truth-table summarizes the circuit action. As you see, the NOR gate recognizes only the input word whose bits are all 0s in other words all inputs must be low to get a high output.

Incidentally, the Boolean equation for a 2-input NOR gat is

$$Y = \overline{A}+\overline{B}$$

Read this as "Y equals NOT A OR B,". If you use this equation, remember that the ORing is done first, then the inversion.

THREE-INPUT GATE



(a) 3-Input (b) 4-Input

Fig 2.22

Fig. 2.22a shows a 3-input NOR gate. The 3-inputs are ORed, and the result is inverted. Therefore, the Boolean equation is

$$Y = \overline{A + B + C}$$

The analysis of Fig. 2.22a. The all inputs are Low, the result of ORing is low; therefore the final output is high. If one or more inputs are high, the result of ORing in high; so the final output is low.

TRUTH-TABLE

THREE-INPUT NOR GATE

A	B	C	A+B+C
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Truth-table summarize the action of a 3-input NOR gate. The circuit recognizes only the input word whose bits are 0s. In other words, all inputs must be low to get a high output.

FOUR-INPUT GATE

Fig. 2.22b is the symbol for a 4-input NOR gate. The inputs are ORed, and the result is inverted. For this reason, the Boolean equation is

$$Y = \overline{A + B + C + D}$$

The corresponding truth table has input words from 0000 to 1111. Word 0000 give a 1 output; all other words produce a 0 output.

TRUTH-TABLE

A	B	C	D	$\overline{A+B+C+D}$
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

2.11 DON'T-CARE CONDITIONS

Sometimes, it doesn't matter what the output is for a given input word. To indicate this, we use an 'X' in the truth table instead of a '0',or a'1'. For instance, look at the truth-table.

TRUTH-TABLE

	A	B	C	D	Y
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	x
9	1	0	0	1	x
10	1	0	1	0	x
11	1	0	1	1	x
12	1	1	0	0	x
13	1	1	0	1	x
14	1	1	1	0	x
15	1	1	1	1	x

The output is an 'X' for any input word from 1000 through 11 11. The X's are called don't cares because they can be treated either '0's or '1's.

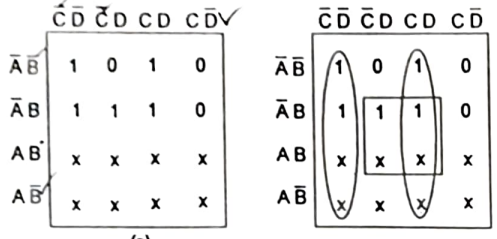


Fig. 2.23

Fig. 2.23a shows the Karnaugh map for truth-table 'X's are used for $\bar{A}\ \bar{B}\ \bar{C}\ \bar{D}$, $A\ \bar{B}\ \bar{C}\ D$, $A\ \bar{B}\ C\ \bar{D}$, $A\ \bar{B}\ C\ D$, $A\ B\ \bar{C}\ \bar{D}$, $A\ B\ \bar{C}\ D$, $A\ B\ C\ \bar{D}$ and $A\ B\ C\ D$ because these are don't cares in the truth-table.

Fig. 2.23b shows the most efficient way to encircle the groups. First, we visualize all X's as 1's and try to form the largest groups that include the real 1s. This gives as three quads. Seconds we visualize all remaining X's as '0's. In this way, the X's are used to the best advantage. We are free to do this because the don't cares can be either '0's or '1's.

THE EQUATION IS = $Y = BD + \bar{C}\bar{D} + CD$

(i) $Y = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}\bar{D} + AB\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D}$

$Y = \bar{C}\bar{D}(\bar{A} + A)(\bar{A} + A)(\bar{B} + B)(\bar{B} + B)$

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

(ii) $Y = \bar{A}\bar{B}CD + \bar{A}BCD + ABCD + A\bar{B}CD$

$Y = CD(\bar{A} + A)(\bar{A} + A)(\bar{B} + B)(\bar{B} + B)$

$Y = CD$

(iii) $Y = \bar{A}B\bar{C}D + \bar{A}BCD + AB\bar{C}D + ABCD$

$Y = BD(\bar{A} + A)(\bar{A} + A)(\bar{C} + C)(\bar{C} + C)$

$Y = BD$

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

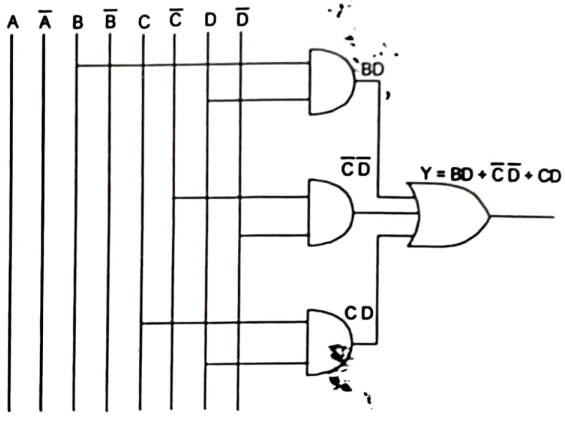


Fig. 2.24 $Y = BD + \bar{C}\bar{D} + CD$

CIRCUIT DIAGRAM OF DON'T CARE CONDITION

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

Fig. 2.24 is the simplified logic circuit.