SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING
DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

UNIT - I
Digital Computer Fundamentals-SBSA1101

1.1 NUMBER SYSTEM

A number system relates quantities and symbols. In digital system how information is represented is key and there are different radices, i.e. number bases, which a numbering system can use.

The memory unit stores programs as well as input, output and intermediate data. The processor unit performs arithmetic and other data processing tasks as specified by the program. The control unit supervises the flow of information between various units. The program and data prepared by the user are transferred into the memory unit by means of an input device such as punch card reader (or) tele typewriter. An output device, such as printer, receives the result of the computations and the printed results are presented to the user.

It can have different base values like: binary (base-2), octal (base-8), decimal (base 10) and hexadecimal (base 16), here the base number represents the number of digits used in that numbering system. As an example, in decimal numbering system the digits used are: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. Therefore the digits for binary are: 0 and 1, the digits for octal are: 0, 1, 2, 3, 4, 5, 6 and 7. For the hexadecimal numbering system, base 16, the digits are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Numbers that contain only two digit 0 and 1 are called Binary Numbers. Each 0 or 1 is called a Bit, from binary digit. A binary number of 4 bits is called a Nibble. A binary number of 8 bits is called a Byte. A binary number of 16 bits is called a Word on some systems, on others a 32-bit number is called a Word while a 16-bit number is called a Halfword.

Using 2 bit 0 and 1 to form

a binary number of 1 bit, numbers are 0 and 1

a binary number of 2 bit, numbers are 00, 01, 10, 11

a binary number of 3 bit, such numbers are 000, 001, 010, 011, 100, 101, 110, 111

a binary number of 4 bit, such numbers are 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000,
1001, 1010, 1011, 1100,1101,1110,1111

Therefore, using n bits there are 2^n binary numbers of n bits

Each digit in a binary number has a value or weight. The LSB has a value of 1. The second from the right has a value of 2, the next 4 , etc.,

Table 1.1 Binary Weights

16	*8*	*4*	*2*	*1*
2^4	2^3	2^2	2^1	2^0

Table 1.2 The binary equivalent for some decimal numbers are given below.

Decimal	0	1	2	3	4	5	6	7	8	9	10	11
Binary	0	1	10	11	100	101	110	111	1000	1001	1010	1011

1.2 NUMBER SYSTEM CONVERSIONS

There are many methods or techniques which can be used to convert numbers from one base to another. We'll demonstrate here the following −

- Decimal to Other Base System
- Other Base System to Decimal
- Other Base System to Non-Decimal
- Binary to Octal
- Octal to Binary
- Binary to Hexadecimal
- Hexadecimal to Binary

Decimal to Other Base System

- **Step 1** − Divide the decimal number to be converted by the value of the new base.
- **Step 2** − Get the remainder from Step 1 as the rightmost digit (least significant digit) of new base number.
- **Step 3** − Divide the quotient of the previous divide by the new base.
- **Step 4** − Record the remainder from Step 3 as the next digit (to the left) of the new base number.

Repeat Steps 3 and 4, getting remainders from right to left, until the quotient becomes zero in Step 3.

The last remainder thus obtained will be the Most Significant Digit (MSD) of the new base number.

Example: Calculating Binary Equivalent of Decimal Number: 2910

Table 1.3 Calculating Binary Equivalent of Decimal Number: 2910

Step	Operation	Result	Remainder
Step 1	29 / 2	14	1
Step 2	14 / 2	7	0
Step 3	7 / 2	3	1
Step 4	3 / 2	1	1
Step 5	1 / 2	0	1

As mentioned in Steps 2 and 4, the remainders have to be arranged in the reverse order so that the first remainder becomes the Least Significant Digit (LSD) and the last remainder becomes the Most Significant Digit (MSD).

Decimal Number − 29_{10} = Binary Number − 11101_2.

Other Base System to Decimal System

Steps

- Step 1 − Determine the column (positional) value of each digit (this depends on the position of the digit and the base of the number system).
- Step 2 − Multiply the obtained column values (in Step 1) by the digits in the corresponding columns.
- Step 3 − Sum the products calculated in Step 2. The total is the equivalent value in decimal.

Example

Binary Number − 11101_2

Calculating Decimal Equivalent –

Table 1.4 Calculating Decimal Equivalent of Binary Number − 11101_2

Step	Binary Number	Decimal Number
Step 1	11101_2	$((1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0))_{10}$
Step 2	11101_2	$(16 + 8 + 4 + 0 + 1)_{10}$
Step 3	11101_2	29_{10}

Other Base System to Non-Decimal System

Steps

- Step 1 − Convert the original number to a decimal number (base 10).
- Step 2 − Convert the decimal number so obtained to the new base number.

Example

Octal Number − 25_8

Calculating Binary Equivalent −

Table 1.5 Calculating Decimal Equivalent of Octal Number − 258

Step 1 − Convert to Decimal

Step	Octal Number	Decimal Number
Step 1	25_8	$((2 \times 8^1) + (5 \times 8^0))_{10}$
Step 2	25_8	$(16 + 5)_{10}$
Step 3	25_8	21_{10}

Octal Number 25_8 = Decimal Number 21_{10}

Table 1.6 Decimal to Binary Conversion

Step 2 − Convert Decimal to Binary

Step	Operation	Result	Remainder
Step 1	21 / 2	10	1
Step 2	10 / 2	5	0
Step 3	5 / 2	2	1
Step 4	2 / 2	1	0
Step 5	1 / 2	0	1

Decimal Number − 21_{10} = Binary Number − 10101_2

Octal Number − 25_8 = Binary Number − 10101_2

Binary to Octal

Steps

- Step 1 − Divide the binary digits into groups of three (starting from the right).
- Step 2 − Convert each group of three binary digits to one octal digit.

Example

Binary Number − 10101_2

Calculating Octal Equivalent –

Table 1.7 Calculating Octal Equivalent of Binary Number − 101012

Step	Binary Number	Octal Number
Step 1	10101_2	010 101
Step 2	10101_2	$2_8\ 5_8$
Step 3	10101_2	25_8

Binary Number − 10101_2 = Octal Number − 25_8

Octal to Binary

Steps

- Step 1 − Convert each octal digit to a 3 digit binary number (the octal digits may be treated as decimal for this conversion).
- Step 2 − Combine all the resulting binary groups (of 3 digits each) into a single binary number.

Example

Octal Number − 25_8

Calculating Binary Equivalent –

Table 1.8 Calculating the binary equivalent of the Octal number - 25_8

Step	Octal Number	Binary Number
Step 1	25_8	$2_{10}\ 5_{10}$
Step 2	25_8	$010_2\ 101_2$
Step 3	25_8	010101_2

Octal Number − 25_8 = Binary Number − 10101_2

Binary to Hexadecimal

Steps

- Step 1 − Divide the binary digits into groups of four (starting from the right).
- Step 2 − Convert each group of four binary digits to one hexadecimal symbol.

Example

Binary Number − 10101_2

Calculating hexadecimal Equivalent –

Table 1.9 Calculating the Hexadecimal equivalent of the Binary number - 10101_2

Step	Binary Number	Hexadecimal Number
Step 1	10101_2	0001 0101
Step 2	10101_2	1_{10} 5_{10}
Step 3	10101_2	15_{16}

Binary Number − 10101_2 = Hexadecimal Number − 15_{16}

Hexadecimal to Binary

Steps

- Step 1 − Convert each hexadecimal digit to a 4 digit binary number (the hexadecimal digits may be treated as decimal for this conversion).
- Step 2 − Combine all the resulting binary groups (of 4 digits each) into a single binary number.

Example

Hexadecimal Number − 15_{16}

Calculating Binary Equivalent –

Table 1.10 Calculating Binary Equivalent of the Hexadecimal Number − 15_{16}

Step	Hexadecimal Number	Binary Number
Step 1	15_{16}	1_{10} 5_{10}
Step 2	15_{16}	0001_2 0101_2
Step 3	15_{16}	00010101_2

Hexadecimal Number − 15_{16} = Binary Number − 10101

1.3 COMPLEMENTS

1's complement

The 1's complement of a number is found by changing all 1's to 0's and all 0's to 1's. This is called as taking complement or 1's complement. Example of 1's Complement is as follows.

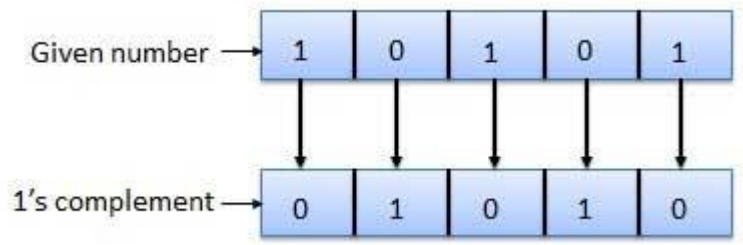


Figure 1.1 Finding 1's complement

2's complement

The 2's complement of binary number is obtained by adding 1 to the Least Significant Bit (LSB) of 1's complement of the number.

2's complement = 1's complement + 1

Example of 2's Complement is as follows.

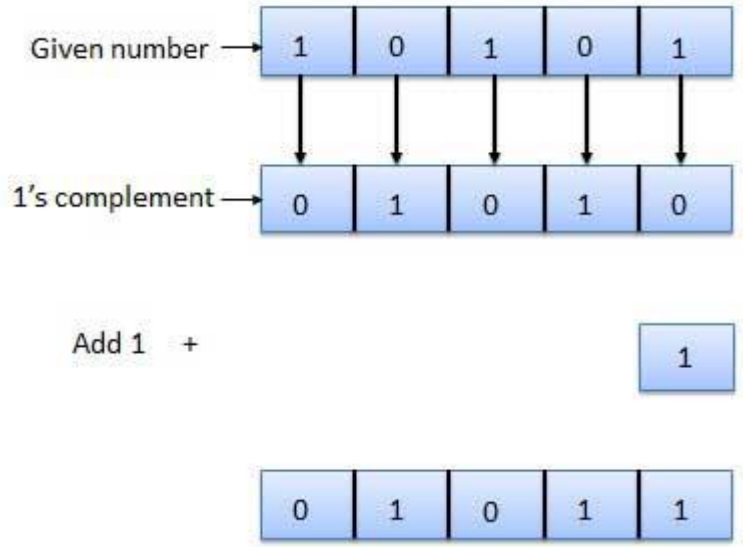


Figure 1.2 Finding 2's complement

1.4 **BINARY CODES**

In the coding, when numbers, letters or words are represented by a specific group of symbols, it is said that the number, letter or word is being encoded. The group of symbols is called as a code. The digital data is represented, stored and transmitted as group of binary bits. This group is also called as binary code. The binary code is represented by the number as well as alphanumeric letter.

Advantages of Binary Code

Following is the list of advantages that binary code offers.

- Binary codes are suitable for the computer applications.
- Binary codes are suitable for the digital communications.

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

Classification of binary codes

The codes are broadly categorized into following four categories.

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

Weighted Codes

Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the number represents a specific weight. Several systems of the codes are used to express the decimal digits 0 through 9. In these codes each decimal digit is represented by a group of four bits.

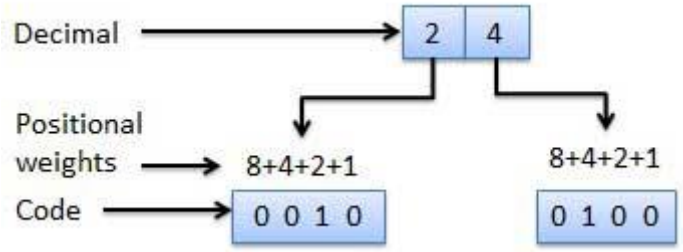


Figure 1.3 Weighted codes

Non-Weighted Codes

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

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 $(0011)_2$ or (3)10 to each code word in 8421. The excess-3 codes are obtained as follows −

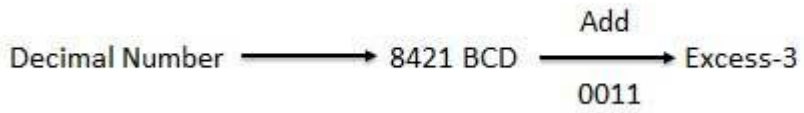


Table 1.11 BCD to Excess-3 Code

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

Gray Code

It is the non-weighted code and it is not arithmetic codes. That means there are no specific weights assigned to the bit position. It has a very special feature that, only one bit will change each time the decimal number is incremented as shown in fig. As only one bit changes at a time, the gray code is called as a unit distance code. The gray code is a cyclic code. Gray code cannot be used for arithmetic operation.

Table 1.12 BCD to Gray Code

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

Application of Gray code

- Gray code is popularly used in the shaft position encoders.

- A shaft position encoder produces a code word which represents the angular position of the shaft.

Binary Coded Decimal (BCD) code

In this code each decimal digit is represented by a 4-bit binary number. BCD is a way to express each of the decimal digits with a binary code. In the BCD, with four bits we can represent sixteen numbers (0000 to 1111). But in BCD code only first ten of these are used (0000 to 1001). The remaining six code combinations i.e. 1010 to 1111 are invalid in BCD.

Table 1.13 Representation of BCD numbers

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

Advantages of BCD Codes

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

Disadvantages of BCD Codes

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

Alphanumeric codes

A binary digit or bit can represent only two symbols as it has only two states '0' or '1'. But this is not enough for communication between two computers because there we need many more symbols for communication. These symbols are required to represent 26 alphabets with capital and small letters, numbers from 0 to 9, punctuation marks and other symbols.

The alphanumeric codes are the codes that represent numbers and alphabetic characters. Mostly such codes also represent other characters such as symbol and various instructions necessary for conveying information. An alphanumeric code should at least represent 10 digits and 26 letters of alphabet i.e. total 36 items. The following three alphanumeric codes are very commonly used for the data representation.

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

ASCII code is a 7-bit code whereas EBCDIC is an 8-bit code. ASCII code is more commonly used worldwide while EBCDIC is used primarily in large IBM computers.

Error detection codes − are used to detect the errors present in the received bit stream. These codes contain some bits, which are included appended to the original bit stream. These codes detect the error, if it is occurred during transmission of the original data bit stream. Example − Parity code, Hamming code.

Error correction codes − are used to correct the errors present in the received data bit stream so that, we will get the original data. Error correction codes also use the similar strategy of error detection codes. Example − Hamming code.
Therefore, to detect and correct the errors, additional bits are appended to the data bits at the time of transmission.

Parity Code
It is easy to include one parity bit either to the left of MSB or to the right of LSB of original bit stream. There are two types of parity codes, namely even parity code and odd parity code based on the type of parity being chosen.

Even Parity Code

The value of even parity bit should be zero, if even number of ones present in the binary code. Otherwise, it should be one. So that, even number of ones present in even parity code. Even parity code contains the data bits and even parity bit.

The following table shows the even parity codes corresponding to each 3-bit binary code. Here, the even parity bit is included to the right of LSB of binary code.

Table 1.14 Even Parity

Binary Code	Even Parity bit	Even Parity Code
000	0	0000
001	1	0011
010	1	0101
011	0	0110
100	1	1001
101	0	1010
110	0	1100
111	1	1111

Here, the number of bits present in the even parity codes is 4. So, the possible even number of ones in these even parity codes are 0, 2 & 4.

- If the other system receives one of these even parity codes, then there is no error in the received data. The bits other than even parity bit are same as that of binary code.

- If the other system receives other than even parity codes, then there will be an errorss in the received data. In this case, we can't predict the original binary code because we don't know the bit positions of error.

Therefore, even parity bit is useful only for detection of error in the received parity code. But, it is not sufficient to correct the error.

Odd Parity Code

The value of odd parity bit should be zero, if odd number of ones present in the binary code. Otherwise, it should be one. So that, odd number of ones present in odd parity code. Odd parity code contains the data bits and odd parity bit.

The following table shows the odd parity codes corresponding to each 3-bit binary code. Here, the odd parity bit is included to the right of LSB of binary code.

Table 1.15 ODD PARITY

Binary Code	Odd Parity bit	Odd Parity Code
000	1	0001
001	0	0010
010	0	0100
011	1	0111
100	0	1000
101	1	1011
110	1	1101
111	0	1110

Here, the number of bits present in the odd parity codes is 4. So, the possible odd number of ones in these odd parity codes are 1 & 3.

- If the other system receives one of these odd parity codes, then there is no error in the received data. The bits other than odd parity bit are same as that of binary code.
- If the other system receives other than odd parity codes, then there is an errorss in the received data. In this case, we can't predict the original binary code because we don't know the bit positions of error.

Therefore, odd parity bit is useful only for detection of error in the received parity code. But, it is not sufficient to correct the error.

Hamming Code

Hamming code is useful for both detection and correction of error present in the received data. This code uses multiple parity bits and we have to place these parity bits in the positions of powers of 2.

The minimum value of 'k' for which the following relation is correct valid is nothing but the required number of parity bits.

$$2^k \geq n+k+1$$

Where,

'n' is the number of bits in the binary code information

'k' is the number of parity bits

Therefore, the number of bits in the Hamming code is equal to n + k.

Let the Hamming code is bn+kbn+k−1.....b3b2b1& parity bits pk,pk−1,....p1. We can place the 'k' parity bits in powers of 2 positions only. In remaining bit positions, we can place the 'n' bits of binary code.

Based on requirement, we can use either even parity or odd parity while forming a Hamming code. But, the same parity technique should be used in order to find whether any error present in the received data.

Follow this procedure for finding parity bits.

- Find the value of p_1, based on the number of ones present in bit positions b_3, b_5, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^0.
- Find the value of p_2, based on the number of ones present in bit positions b_3, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^1.
- Find the value of p_3, based on the number of ones present in bit positions b_5, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^2.
- Similarly, find other values of parity bits.

Follow this procedure for finding check bits.

- Find the value of c_1, based on the number of ones present in bit positions b_1, b_3, b_5, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^0.
- Find the value of c_2, based on the number of ones present in bit positions b_2, b_3, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^1.
- Find the value of c_3, based on the number of ones present in bit positions b_4, b_5, b_6, b_7 and so on. All these bit positions suffixes in their equivalent binary have '1' in the place value of 2^2.
- Similarly, find other values of check bits.

The decimal equivalent of the check bits in the received data gives the value of bit position, where the error is present. Just complement the value present in that bit position. Therefore, we will get the original binary code after removing parity bits.

Example

Let us find the Hamming code for binary code, $d_4d_3d_2d_1$ = 1000. Consider even parity bits.

The number of bits in the given binary code is n=4.

We can find the required number of parity bits by using the following mathematical relation.

$2k \geq n+k+1$

Substitute, n=4 in the above mathematical relation.

$\Rightarrow 2k \geq 4+k+1$
$\Rightarrow 2k \geq 5+k$

The minimum value of k that satisfied the above relation is 3. Hence, we require 3 parity bits p_1, p_2, and p_3. Therefore, the number of bits in Hamming code will be 7, since there are 4 bits in binary code and 3 parity bits. We have to place the parity bits and bits of binary code in the Hamming code as shown below.

The 7-bit Hamming code is b7b6b5b4b3b2b1=d4d3d2p3d1p2bp1

By substituting the bits of binary code, the Hamming code will be b7b6b5b4b3b2b1=100p3Op2p. Now, let us find the parity bits.

p1=b7⊕b5⊕b3=1⊕0⊕0=1
p2=b7⊕b6⊕b3=1⊕0⊕0=1
p3=b7⊕b6⊕b5=1⊕0⊕0=1

By substituting these parity bits, the Hamming code will be b7b6b5b4b3b2b1=1001011.

1.5 LOGIC GATES

Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a **certain logic**. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc.

AND Gate

A circuit which performs an AND operation is shown in figure. It has n input (n >= 2) and one output.

Y	=	A AND B AND C ..... N
Y	=	A.B.C ..... N
Y	=	ABC ..... N

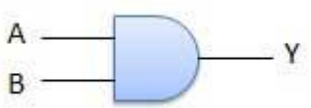


Figure 1.4 Logic diagram of AND gate

Table 1.16 Truth Table of AND Gate

Inputs		Output
A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

A circuit which performs an OR operation is shown in figure. It has n input (n >= 2) and one output.

Y = A OR B OR C ..... N
Y = A + B + C ..... N

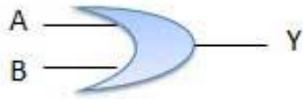


Figure 1.5 Logic diagram of OR gate

Table 1.17 Truth Table of OR gate

Inputs		Output
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate

NOT gate is also known as **Inverter**. It has one input A and one output Y.

Y = NOT A

Y = $\overline{A}$

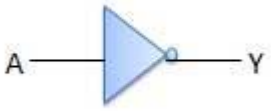


Figure 1.6 Logic diagram of NOT gate

Table 1.18 Truth Table of NOT gate

Inputs	Output
A	B
0	1
1	0

NAND Gate

A NOT-AND operation is known as NAND operation. It has n input (n >= 2) and one output.

Y = A NOT AND B NOT AND C ..... N

Y = A NAND B NAND C ..... N

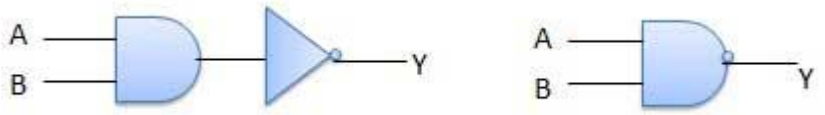


Figure 1.7 Logic diagram of NAND gate

Table 1.19 Truth Table of NAND gate

Inputs		Output
A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

A NOT-OR operation is known as NOR operation. It has n input (n >= 2) and one output.

Y = A NOT OR B NOT OR C ..... N
Y = A NOR B NOR C ..... N

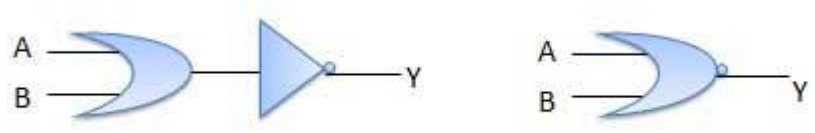


Figure 1.8 Logic diagram of NOR gate

Table 1.20 7Truth Table of NOR gate

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

XOR Gate

XOR or Ex-OR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-OR gate is abbreviated as EX-OR gate or sometime as X-OR gate. It has n input (n >= 2) and one output.

Y = A XOR B XOR C ..... N
Y = $A \oplus B \oplus C$ ..... N
Y = $\overline{A}B + A\overline{B}$

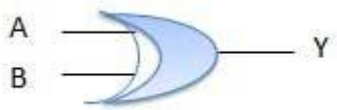


Figure 1.9 Logic diagram of XOR gate

Table 1.21 Truth Table of XOR gate

Inputs		Output
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

XNOR Gate

XNOR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-NOR gate is abbreviated as EX-NOR gate or sometime as X-NOR gate. It has n input (n >= 2) and one output.

$Y = A \text{ XOR } B \text{ XOR } C \ldots\ldots N$

$Y = A \odot B \odot C \ldots\ldots N$

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

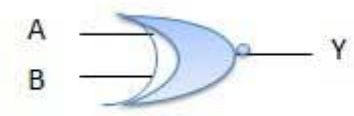


Figure 1.10 Logic diagram of XNOR gate

Table 1.22 Truth Table of XNOR gate

Inputs		Output
A	B	$A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

References :

1. Moris Mano, “Digital Computer Fundamentals” TMH 3rd Edition
2. http://www.tutorialspoint.com/computer_logical_organization/number_system_conversion.htm
3. http://www.electronics-tutorials.ws/binary/signed-binary-numbers.html
4. HAMMING, R. →. “Error Detecting and Error Correcting Codes.” Bell System Tech. Jour., 29 (1950): 147–160.
5. A.P GODSE,D.A.GODSE .”Digital Systems”.Technical Publications.Pune.
6. http://www.tutorialspoint.com/computer_logical_organization/binary_codes.htm
7. http://nptel.ac.in/courses/Webcourse-contents/IIScBANG/Digital%20Systems/Digital%20Systems.pdf
8. Digital Logic Circuits by D.A.Godse A.P.Godse

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING
DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

UNIT - II
Digital Computer Fundamentals-SBSA1101

2.1 BOOLEAN ALGEBRA

Boolean algebra is used to analyse and simplify the digital (logic) circuits. It uses only the binary numbers i.e. 0 and 1. It is also called as Binary Algebra or logical Algebra. Boolean algebra was invented by George Boole in 1854.

Rule in Boolean algebra

Following are the important rules used in Boolean algebra.

- Variable used can have only two values. Binary 1 for HIGH and Binary 0 for LOW.
- Complement of a variable is represented by an overbar (-). Thus, complement of variable B is represented as $\overline{B}$. Thus if B = 0 then $\overline{B}$ = 1 and B = 1 then $\overline{B}$ = 0.
- ORing of the variables is represented by a plus (+) sign between them. For example ORing of A, B, C is represented as A + B + C.
- Logical ANDing of the two or more variable is represented by writing a dot between them such as A.B.C. Sometime the dot may be omitted like ABC.

Boolean Laws

There are six types of Boolean Laws.

Commutative law

Any binary operation which satisfies the following expression is referred to as commutative operation.

(i) $A.B = B.A$ (ii) $A + B = B + A$

Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

Associative law

This law states that the order in which the logic operations are performed is irrelevant as their effect is the same.

(i) $(A.B).C = A.(B.C)$ (ii) $(A + B) + C = A + (B + C)$

Distributive law

Distributive law states the following condition.

$A.(B + C) = A.B + A.C$

AND law

These laws use the AND operation. Therefore they are called as AND laws.

(i) $A.0 = 0$ (ii) $A.1 = A$

(iii) $A.A = A$ (iv) $A.\overline{A} = 0$

OR law

These laws use the OR operation. Therefore they are called as OR laws.

(i) $A + 0 = A$ (ii) $A + 1 = 1$

(iii) $A + A = A$ (iv) $A + \overline{A} = 1$

Inversion law

This law uses the NOT operation. The inversion law states that double inversion of a variable results in the original variable itself.

$\overline{\overline{A}} = A$

Theorems

De Morgan has suggested two theorems which are extremely useful in Boolean Algebra. The two theorems are discussed below.

Theorem 1

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

NAND = Bubbled OR

- The left hand side (LHS) of this theorem represents a NAND gate with inputs A and B, whereas the right hand side (RHS) of the theorem represents an OR gate with inverted inputs.
- This OR gate is called as Bubbled OR.

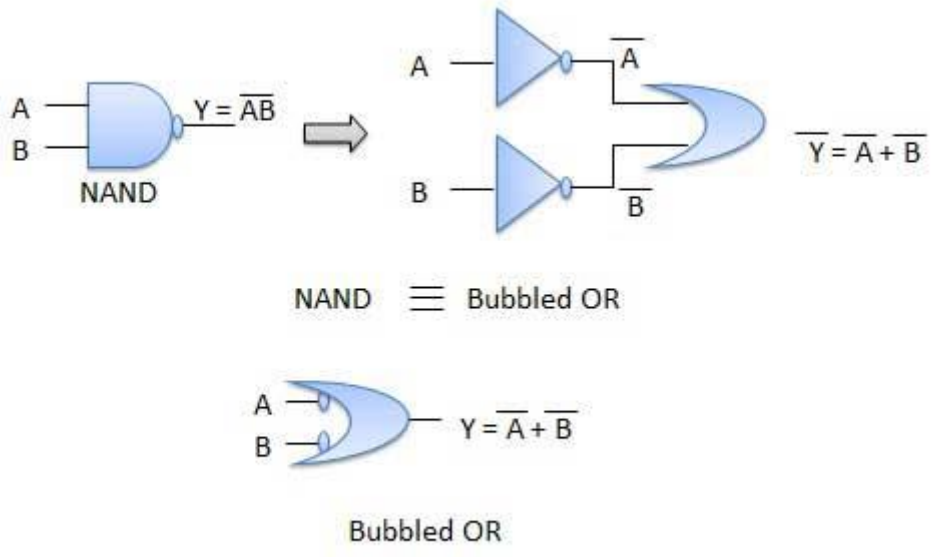


Table showing verification of the De Morgan's first theorem −

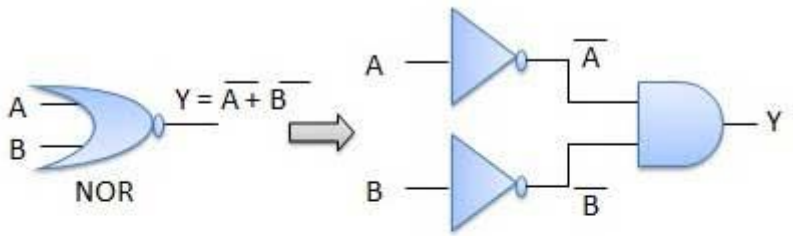
A	B	$\overline{AB}$	$\overline{A}$	$\overline{B}$	$\overline{A} + \overline{B}$
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

Theorem 2

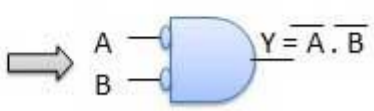
$\overline{A + B} = \overline{A} . \overline{B}$

NOR = Bubbled AND

- The LHS of this theorem represents a NOR gate with inputs A and B, whereas the RHS represents an AND gate with inverted inputs.
- This AND gate is called as Bubbled AND.



NOR ☰ Bubbled AND



Bubbled AND

Table showing verification of the De Morgan's second theorem −

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

Consensus Theorem

```
AB + A C + BC = AB + A C
```

Proof:

```
LHS      = AB + A C + BC
         = AB + A C + BC (A+A)
         = AB + A C + ABC + ABC
         = AB (1+C) + A C (1+C)
         = AB + A C = RHS
```

2.2 AXIOMS

There are some set of logical expressions which we accept as true and upon which we can build a set of useful theorems. These sets of logical expressions are known as Axioms or postulates of Boolean algebra. An axiom is nothing more than the definition of three basic logic operations (AND, OR and NOT). All axioms defined in Boolean algebra are the results of an operation that is performed by a logical gate.

```
Axiom 1: 0.0 = 0     Axiom 6: 0+1 = 1
Axiom 2: 0.1 = 0     Axiom 7: 1+0 = 1
Axiom 3: 1.0 = 0     Axiom 8: 1+1 = 1
Axiom 4: 1.1 = 1     Axiom 9: 0′ = 1
Axiom 5: 0+0 = 0     Axiom 10: 1′ = 0
```

2.3 TRUTH TABLE SIMPLIFICATION OF BOOLEAN FUNCTION

Boolean algebra deals with binary variables and logic operation. A Boolean Function is described by an algebraic expression called Boolean expression which consists of binary variables, the constants 0 and 1, and the logic operation symbols. Consider the following example.

$F(A, B, C, D) = A + \overline{BC} + ADC$ Equation No. 1

Boolean Function — Boolean Expression

Here the left side of the equation represents the output Y. So we can state equation no. 1

$Y = A + \overline{BC} + ADC$

Truth Table Formation

A truth table represents a table having all combinations of inputs and their corresponding result.

It is possible to convert the switching equation into a truth table. For example, consider the following switching equation.

$F(A, B, C) = A + BC$

The output will be high (1) if A = 1 or BC = 1 or both are 1. The truth table for this equation is shown by Table (a). The number of rows in the truth table is 2^n where n is the number of input variables (n=3 for the given equation). Hence there are $2^3 = 8$ possible input combination of inputs.

Inputs			Output
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Canonical and Standard Forms

Logical functions are generally expressed in terms of different combinations of logical variables with their true forms as well as the complement forms. Binary logic values obtained by the logical functions and logic variables are in binary form. An arbitrary logic function can be expressed in the following forms.

(i) Sum of the Products (SOP)
(ii) Product of the Sums (POS) Product Term:

In Boolean algebra, the logical product of several variables on which a function depends is considered to be a product term. In other words, the AND function is referred to as a product term or standard product. The variables in a product term can be either in true form or in complemented form. For example, ABC′ is a product term.

Sum Term:

An OR function is referred to as a sum term. The logical sum of several variables on which a function depends is considered to be a sum term. Variables in a sum term can also be either in true form or in complemented form. For example, A + B + C′ is a sum term.

Sum of Products (SOP):

The logical sum of two or more logical product terms is referred to as a sum of products expression. It is basically an OR operation on AND operated variables. For example, Y = AB + BC + AC or Y = A′B + BC + AC′ are sum of products expressions.

Product of Sums (POS):

Similarly, the logical product of two or more logical sum terms is called a product of sums expression. It is an AND operation on OR operated variables. For example, Y = (A + B + C)(A + B′ + C)(A + B + C′) or Y = (A + B + C)(A′ + B′ + C′) are product of sums expressions.

Standard form:

The standard form of the Boolean function is when it is expressed in sum of the products or product of the sums fashion. The examples stated above, like Y =AB + BC + AC or Y = (A + B + C)(A + B′ + C)(A + B + C′) are the standard forms. However, Boolean functions are also sometimes expressed in nonstandard forms like F = (AB + CD)(A′B′ + C′D′), which is neither a sum of products form nor a product of sums form. However, the same expression can be converted to a standard form with help of various Boolean properties, as:
F = (AB + CD)(A′B′ + C′D′) = A′B′CD + ABC′D′

Minterm

A product term containing all n variables of the function in either true or complemented form is called the minterm. Each minterm is obtained by an AND operation of the variables in their true form or complemented form. For a two-variable function, four different combinations are possible, such as, A′B′, A′B, AB′, and AB. These product terms are called the fundamental products or standard products or minterms. In the minterm, a variable will possess the value 1 if it is in true or uncomplemented form, whereas, it contains the value 0 if it is in complemented form. For three variables function, eight minterms are possible as listed in the following table

A	B	C	Minterm
0	0	0	A’B’C’
0	0	1	A’B’C
0	1	0	A’BC’
0	1	1	A’BC
1	0	0	AB’C’
1	0	1	AB’C
1	1	0	ABC’
1	1	1	ABC

So, if the number of variables is n, then the possible number of minterms is 2n. The main property of a minterm is that it possesses the value of 1 for only one combination of n input variables and the rest of the 2n – 1 combinations have the logic value of 0. This means, for the above three variables example, if A = 0, B = 1, C = 1 i.e., for input combination of 011, there is only one combination A′BC that has the value 1, the rest of the seven combinations have the value 0.

Canonical Sum of Product Expression:

When a Boolean function is expressed as the logical sum of all the minterms from the rows of a truth table, for which the value of the function is 1, it is referred to as the canonical sum of product expression. The same can be expressed in a compact form by listing the corresponding decimal-equivalent codes of the minterms containing a function value of 1.
For example, if the canonical sum of product form of a three-variable logic function F has the minterms A′BC, AB′C, and ABC′, this can be expressed as the sum of the decimal codes corresponding to these minterms as below.
F (A,B,C) = (3,5,6)
= m3 + m5 + m6
= A′BC + AB′C + ABC′
where Σ (3,5,6) represents the summation of minterms corresponding to decimal codes 3, 5, and
6. The canonical sum of products form of a logic function can be obtained by using the following procedure:
1. Check each term in the given logic function. Retain if it is a minterm, continue to examine the next term in the same manner.

2. Examine for the variables that are missing in each product which is not a minterm. If the missing variable in the minterm is X, multiply that minterm with (X+X′).

2. Multiply all the products and discard the redundant terms.

Maxterm

A sum term containing all n variables of the function in either true or complemented form is called the maxterm. Each maxterm is obtained by an OR operation of the variables in their true form or complemented form. Four different combinations are possible for a two-variable function, such as, A′ + B′, A′ + B, A + B′, and A + B. These sum terms are called the standard sums or maxterms. Note that, in the maxterm, a variable will possess the value 0, if it is in true or uncomplemented form, whereas, it contains the value 1, if it is in complemented form. Like minterms, for a three-variable function, eight maxterms are also possible as listed in the following table

A	B	C	Maxterm
0	0	0	A+B+C
0	0	1	A+B+C’
0	1	0	A+B’+C
0	1	1	A+B’+C’
1	0	0	A’+B+C
1	0	1	A’+B+C’
1	1	0	A’+B’+C
1	1	1	A’+B’+C’

So, if the number of variables is n, then the possible number of maxterms is 2n. The main property of a maxterm is that it possesses the value of 0 for only one combination of n input variables and the rest of the 2n –1 combinations have the logic value of 1. This means, for the above three variables example, if A = 1, B = 1, C = 0 i.e., for input combination of 110, there is only one combination A′ + B′ + C that has the value 0, the rest of the seven combinations have the value 1.

Canonical Product of Sum Expression:

When a Boolean function is expressed as the logical product of all the maxterms from the rows of a truth table, for which the value of the function is 0, it is referred to as the canonical product of sum expression. The same can be expressed in a compact form by listing the corresponding decimal equivalent codes of the maxterms containing a function value of 0. For example, if the canonical product of sums form of a three-variable logic function F has the maxterms A + B + C, A + B′ + C, and A′ + B + C′, this can be expressed as the product of the decimal codes corresponding to these maxterms as below,

F (A,B,C) = Π (0,2,5)

= M0 M2 M5

= (A + B + C) (A + B′ + C) (A′ + B + C′)

where Π (0,2,5) represents the product of maxterms corresponding to decimal codes 0, 2, and 5. The canonical product of sums form of a logic function can be obtained by using the following procedure.

1. Check each term in the given logic function. Retain it if it is a maxterm, continue to examine the next term in the same manner.

2. Examine for the variables that are missing in each sum term that is not a maxterm. If the missing variable in the maxterm is X, add that maxterm with (X.X′).

3. Expand the expression using the properties and postulates as described earlier and discard the redundant terms. Some examples are given here to explain the above procedure.

Methods to simplify the Boolean function

The methods used for simplifying the Boolean function are as follows −

- Karnaugh-map or K-map, and
- . Mc-Clausky (or) Tabular Method

2.4 K MAP

In previous chapters, we have simplified the Boolean functions using Boolean postulates and theorems. It is a time consuming process and we have to re-write the simplified expressions after each step.

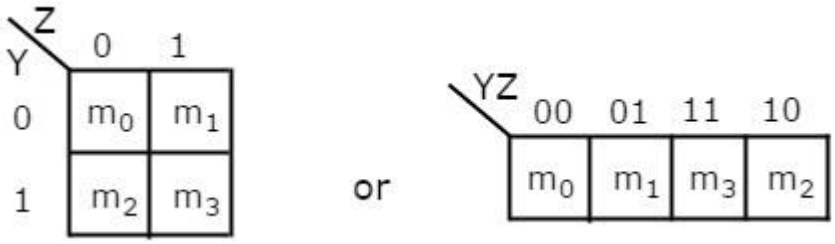
To overcome this difficulty, Karnaugh introduced a method for simplification of Boolean functions in an easy way. This method is known as Karnaugh map method or K-map method. It is a graphical method, which consists of 2^n cells for 'n' variables. The adjacent cells are differed only in single bit position.

K-Maps for 2 to 5 Variables

K-Map method is most suitable for minimizing Boolean functions of 2 variables to 5 variables. Now, let us discuss about the K-Maps for 2 to 5 variables one by one.

2 Variable K-Map

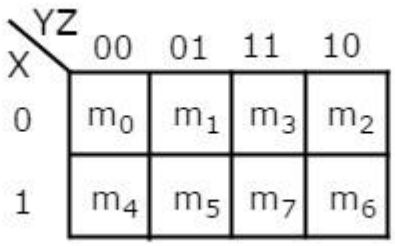
The number of cells in 2 variable K-map is four, since the number of variables is two. The following figure shows 2 variable K-Map.



- There is only one possibility of grouping 4 adjacent min terms.
- The possible combinations of grouping 2 adjacent min terms are {(m_0, m_1), (m_2, m_3), (m_0, m_2) and (m_1, m_3)}.

3 Variable K-Map

The number of cells in 3 variable K-map is eight, since the number of variables is three. The following figure shows 3 variable K-Map.

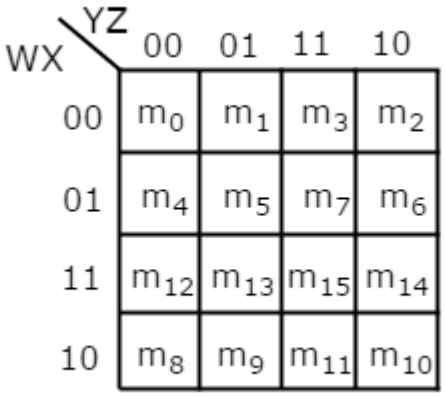


- There is only one possibility of grouping 8 adjacent min terms.
- The possible combinations of grouping 4 adjacent min terms are {(m_0, m_1, m_3, m_2), (m_4, m_5, m_7, m_6), (m_0, m_1, m_4, m_5), (m_1, m_3, m_5, m_7), (m_3, m_2, m_7, m_6) and (m_2, m_0, m_6, m_4)}.

- The possible combinations of grouping 2 adjacent min terms are {(m_0, m_1), (m_1, m_3), (m_3, m_2), (m_2, m_0), (m_4, m_5), (m_5, m_7), (m_7, m_6), (m_6, m_4), (m_0, m_4), (m_1, m_5), (m_3, m_7) and (m_2, m_6)}.
- If x=0, then 3 variable K-map becomes 2 variable K-map.

4 Variable K-Map

The number of cells in 4 variable K-map is sixteen, since the number of variables is four. The following figure shows 4 variable K-Map.

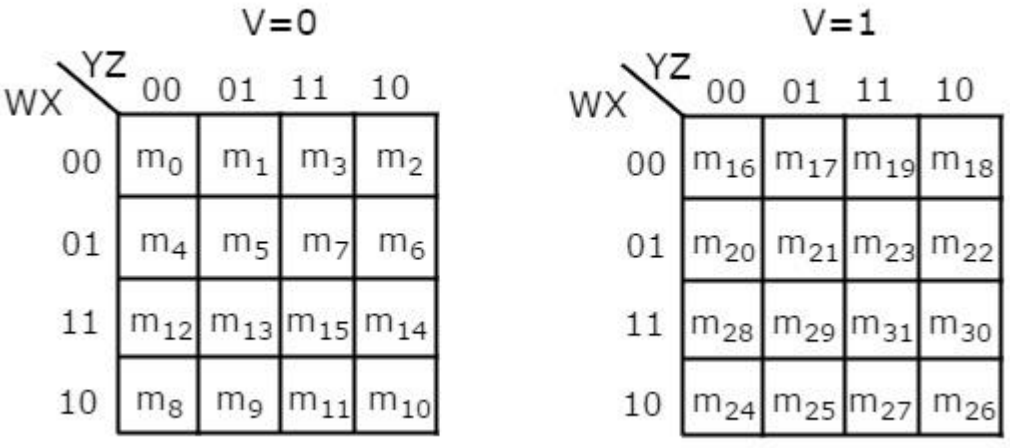


WX \ YZ	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

- There is only one possibility of grouping 16 adjacent min terms.
- Let R_1, R_2, R_3 and R_4 represents the min terms of first row, second row, third row and fourth row respectively. Similarly, C_1, C_2, C_3 and C_4 represents the min terms of first column, second column, third column and fourth column respectively. The possible combinations of grouping 8 adjacent min terms are {(R_1, R_2), (R_2, R_3), (R_3, R_4), (R_4, R_1), (C_1, C_2), (C_2, C_3), (C_3, C_4), (C_4, C_1)}.
- If w=0, then 4 variable K-map becomes 3 variable K-map.

5 Variable K-Map

The number of cells in 5 variable K-map is thirty-two, since the number of variables is 5. The following figure shows 5 variable K-Map.



V=0

WX \ YZ	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

V=1

WX \ YZ	00	01	11	10
00	m_{16}	m_{17}	m_{19}	m_{18}
01	m_{20}	m_{21}	m_{23}	m_{22}
11	m_{28}	m_{29}	m_{31}	m_{30}
10	m_{24}	m_{25}	m_{27}	m_{26}

- There is only one possibility of grouping 32 adjacent min terms.
- There are two possibilities of grouping 16 adjacent min terms. i.e., grouping of min terms from m_0 to m_{15} and m_{16} to m_{31}.
- If v=0, then 5 variable K-map becomes 4 variable K-map.

In the above all K-maps, we used exclusively the min terms notation. Similarly, you can use exclusively the Max terms notation.

Minimization of Boolean Functions using K-Maps

If we consider the combination of inputs for which the Boolean function is '1', then we will get the Boolean function, which is in standard sum of products form after simplifying the K-map.

Similarly, if we consider the combination of inputs for which the Boolean function is '0', then we will get the Boolean function, which is in standard product of sums form after simplifying the K-map.

Follow these rules for simplifying K-maps in order to get standard sum of products form.

- Select the respective K-map based on the number of variables present in the Boolean function.
- If the Boolean function is given as sum of min terms form, then place the ones at respective min term cells in the K-map. If the Boolean function is given as sum of products form, then place the ones in all possible cells of K-map for which the given product terms are valid.
- Check for the possibilities of grouping maximum number of adjacent ones. It should be powers of two. Start from highest power of two and upto least power of two. Highest power is equal to the number of variables considered in K-map and least power is zero.
- Each grouping will give either a literal or one product term. It is known as prime implicant. The prime implicant is said to be essential prime implicant, if atleast single '1' is not covered with any other groupings but only that grouping covers.
- Note down all the prime implicants and essential prime implicants. The simplified Boolean function contains all essential prime implicants and only the required prime implicants.

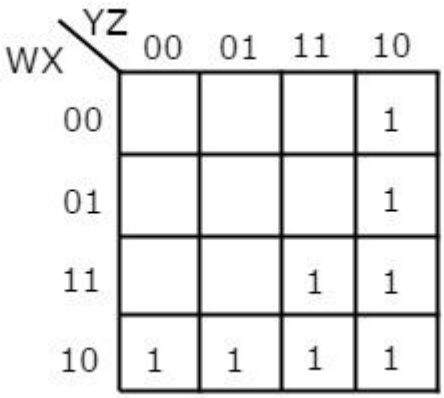
Note 1 − If outputs are not defined for some combination of inputs, then those output values will be represented with don't care symbol 'x'. That means, we can consider them as either '0' or '1'.

Note 2 − If don't care terms also present, then place don't cares 'x' in the respective cells of K-map. Consider only the don't cares 'x' that are helpful for grouping maximum number of adjacent ones. In those cases, treat the don't care value as '1'.

Example

Let us simplify the following Boolean function, fW,X,Y,ZW,X,Y,Z= WX'Y' + WY + W'YZ' using K-map.

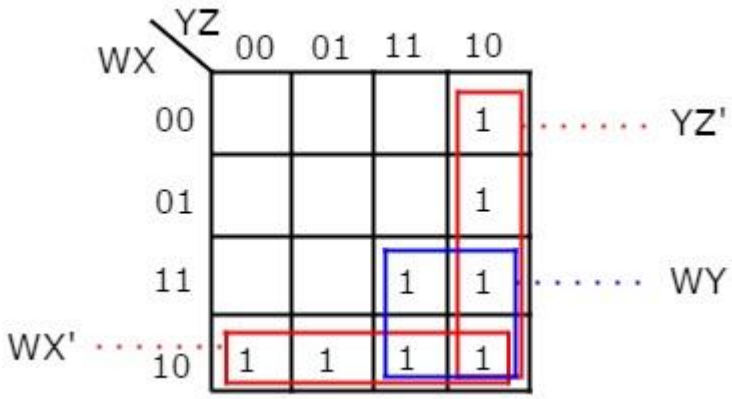
The given Boolean function is in sum of products form. It is having 4 variables W, X, Y & Z. So, we require 4 variable K-map. The 4 variable K-map with ones corresponding to the given product terms is shown in the following figure.



Here, 1s are placed in the following cells of K-map.

- The cells, which are common to the intersection of Row 4 and columns 1 & 2 are corresponding to the product term, WX'Y'.
- The cells, which are common to the intersection of Rows 3 & 4 and columns 3 & 4 are corresponding to the product term, WY.
- The cells, which are common to the intersection of Rows 1 & 2 and column 4 are corresponding to the product term, W'YZ'.

There are no possibilities of grouping either 16 adjacent ones or 8 adjacent ones. There are three possibilities of grouping 4 adjacent ones. After these three groupings, there is no single one left as ungrouped. So, we no need to check for grouping of 2 adjacent ones. The 4 variable K-map with these three groupings is shown in the following figure.



Here, we got three prime implicants WX', WY & YZ'. All these prime implicants are essential because of following reasons.

- Two ones (m_8 & m_9) of fourth row grouping are not covered by any other groupings. Only fourth row grouping covers those two ones.
- Single one (m_{15}) of square shape grouping is not covered by any other groupings. Only the square shape grouping covers that one.

- Two ones (m_2 & m_6) of fourth column grouping are not covered by any other groupings. Only fourth column grouping covers those two ones.

Therefore, the simplified Boolean function is

f = WX' + WY + YZ'

Follow these rules for simplifying K-maps in order to get standard product of sums form.

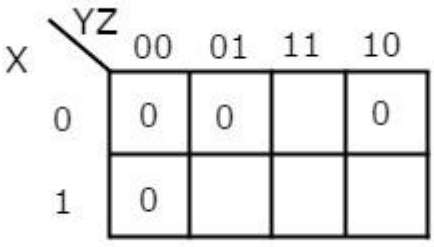
- Select the respective K-map based on the number of variables present in the Boolean function.
- If the Boolean function is given as product of Max terms form, then place the zeroes at respective Max term cells in the K-map. If the Boolean function is given as product of sums form, then place the zeroes in all possible cells of K-map for which the given sum terms are valid.
- Check for the possibilities of grouping maximum number of adjacent zeroes. It should be powers of two. Start from highest power of two and upto least power of two. Highest power is equal to the number of variables considered in K-map and least power is zero.
- Each grouping will give either a literal or one sum term. It is known as prime implicant. The prime implicant is said to be essential prime implicant, if atleast single '0' is not covered with any other groupings but only that grouping covers.
- Note down all the prime implicants and essential prime implicants. The simplified Boolean function contains all essential prime implicants and only the required prime implicants.

Note − If don't care terms also present, then place don't cares 'x' in the respective cells of K-map. Consider only the don't cares 'x' that are helpful for grouping maximum number of adjacent zeroes. In those cases, treat the don't care value as '0'.

Example

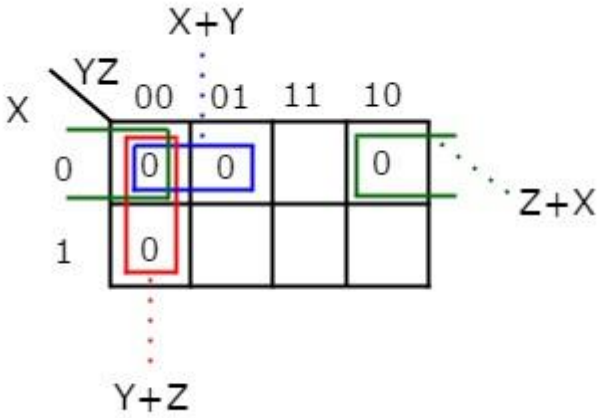
Let us simplify the following Boolean function, f(X,Y,Z)=∏M(0,1,2,4)f(X,Y,Z)=∏M(0,1,2,4) using K-map.

The given Boolean function is in product of Max terms form. It is having 3 variables X, Y & Z. So, we require 3 variable K-map. The given Max terms are M_0, M_1, M_2 & M_4. The 3 variable K-map with zeroes corresponding to the given Max terms is shown in the following figure.



There are no possibilities of grouping either 8 adjacent zeroes or 4 adjacent zeroes. There are three possibilities of grouping 2 adjacent zeroes. After these three groupings, there is no single

zero left as ungrouped. The 3 variable K-map with these three groupings is shown in the following figure.



Here, we got three prime implicants X + Y, Y + Z & Z + X. All these prime implicants are essential because one zero in each grouping is not covered by any other groupings except with their individual groupings.

Therefore, the simplified Boolean function is

f = X+YX+Y.Y+ZY+Z.Z+XZ+X

In this way, we can easily simplify the Boolean functions up to 5 variables using K-map method. For more than 5 variables, it is difficult to simplify the functions using K-Maps. Because, the number of cells in K-map gets doubled by including a new variable.

2.5 Mc-CLUSKEY

Quine-McClukey tabular method is a tabular method based on the concept of prime implicants. We know that prime implicant is a product or sum term, which can't be further reduced by combining with any other product or sum terms of the given Boolean function.

This tabular method is useful to get the prime implicants by repeatedly using the following Boolean identity.

xy + xy' = xy+y′= x.1 = x

Procedure of Quine-McCluskey Tabular Method

Follow these steps for simplifying Boolean functions using Quine-McClukey tabular method.

Step 1 − Arrange the given min terms in an ascending order and make the groups based on the number of ones present in their binary representations. So, there will be at most 'n+1' groups if there are 'n' Boolean variables in a Boolean function or 'n' bits in the binary equivalent of min terms.

Step 2 − Compare the min terms present in successive groups. If there is a change in only one-bit position, then take the pair of those two min terms. Place this symbol '_' in the differed bit position and keep the remaining bits as it is.

Step 3 − Repeat step2 with newly formed terms till we get all prime implicants.

Step 4 − Formulate the prime implicant table. It consists of set of rows and columns. Prime implicants can be placed in row wise and min terms can be placed in column wise. Place '1' in the cells corresponding to the min terms that are covered in each prime implicant.

Step 5 − Find the essential prime implicants by observing each column. If the min term is covered only by one prime implicant, then it is essential prime implicant. Those essential prime implicants will be part of the simplified Boolean function.

Step 6 − Reduce the prime implicant table by removing the row of each essential prime implicant and the columns corresponding to the min terms that are covered in that essential prime implicant. Repeat step 5 for reduced prime implicant table. Stop this process when all min terms of given Boolean function are over.

Example

Let us simplify the following Boolean function, $f(W,X,Y,Z)=\sum m(2,6,8,9,10,11,14,15)$ using Quine-McClukey tabular method.

The given Boolean function is in sum of min terms form. It is having 4 variables W, X, Y & Z. The given min terms are 2, 6, 8, 9, 10, 11, 14 and 15. The ascending order of these min terms based on the number of ones present in their binary equivalent is 2, 8, 6, 9, 10, 11, 14 and 15. The following table shows these min terms and their equivalent binary representations.

Group Name	Min terms	W	X	Y	Z
GA1	2	0	0	1	0
	8	1	0	0	0
GA2	6	0	1	1	0
	9	1	0	0	1
	10	1	0	1	0
GA3	11	1	0	1	1
	14	1	1	1	0
GA4	15	1	1	1	1

The given min terms are arranged into 4 groups based on the number of ones present in their binary equivalents. The following table shows the possible merging of min terms from adjacent groups.

Group Name	Min terms	W	X	Y	Z
GB1	2,6	0	-	1	0
	2,10	-	0	1	0
	8,9	1	0	0	-
	8,10	1	0	-	0
GB2	6,14	-	1	1	0
	9,11	1	0	-	1
	10,11	1	0	1	-
	10,14	1	-	1	0
GB3	11,15	1	-	1	1
	14,15	1	1	1	-

The min terms, which are differed in only one-bit position from adjacent groups are merged. That differed bit is represented with this symbol, '-'. In this case, there are three groups and each group contains combinations of two min terms. The following table shows the possible merging of min term pairs from adjacent groups.

Group Name	Min terms	W	X	Y	Z
GB1	2,6,10,14	-	-	1	0
	2,10,6,14	-	-	1	0
	8,9,10,11	1	0	-	-
	8,10,9,11	1	0	-	-
GB2	10,11,14,15	1	-	1	-
	10,14,11,15	1	-	1	-

The successive groups of min term pairs, which are differed in only one-bit position are merged. That differed bit is represented with this symbol, '-'. In this case, there are two groups and each group contains combinations of four min terms. Here, these combinations of 4 min terms are available in two rows. So, we can remove the repeated rows. The reduced table after removing the redundant rows is shown below.

Group Name	Min terms	W	X	Y	Z
GC1	2,6,10,14	-	-	1	0
	8,9,10,11	1	0	-	-
GC2	10,11,14,15	1	-	1	-

Further merging of the combinations of min terms from adjacent groups is not possible, since they are differed in more than one-bit position. There are three rows in the above table. So, each row will give one prime implicant. Therefore, the prime implicants are YZ', WX' & WY.

The prime implicant table is shown below.

Min terms / Prime Implicants	2	6	8	9	10	11	14	15
YZ’	1	1			1		1	
WX’			1	1	1	1		
WY					1	1	1	1

The prime implicants are placed in row wise and min terms are placed in column wise. 1s are placed in the common cells of prime implicant rows and the corresponding min term columns.

The min terms 2 and 6 are covered only by one prime implicant YZ’. So, it is an essential prime implicant. This will be part of simplified Boolean function. Now, remove this prime implicant row and the corresponding min term columns. The reduced prime implicant table is shown below.

Min terms / Prime Implicants	8	9	11	15
WX’	1	1	1	
WY			1	1

The min terms 8 and 9 are covered only by one prime implicant WX’. So, it is an essential prime implicant. This will be part of simplified Boolean function. Now, remove this prime implicant row and the corresponding min term columns. The reduced prime implicant table is shown below.

Min terms / Prime Implicants	15
WY	1

The min term 15 is covered only by one prime implicant WY. So, it is an essential prime implicant. This will be part of simplified Boolean function.

In this example problem, we got three prime implicants and all the three are essential. Therefore, the simplified Boolean function is

fW,X,Y,Z = YZ’ + WX’ + WY.

References :

1. Moris Mano, “Digital Computer Fundamentals” TMH 3rd Edition
2. http://www.tutorialspoint.com/computer_logical_organization/number_system_conversion.htm
3. http://www.electronics-tutorials.ws/binary/signed-binary-numbers.html
4. HAMMING, R. →. “Error Detecting and Error Correcting Codes.” Bell System Tech. Jour., 29 (1950): 147–160.
5. A.P GODSE,D.A.GODSE .”Digital Systems”.Technical Publications.Pune.
6. http://www.tutorialspoint.com/computer_logical_organization/binary_codes.htm
7. http://nptel.ac.in/courses/Webcourse-contents/IIScBANG/Digital%20Systems/Digital%20Systems.pdf
8. Digital Logic Circuits by D.A.Godse A.P.Godse

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

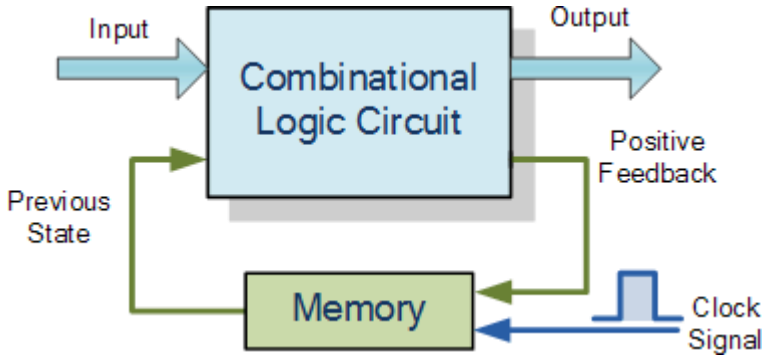
DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

UNIT - III

Digital Computer Fundamentals-SBSA1101

3.1 SEQUENTIAL LOGIC CIRCUITS

Unlike Combinational Logic circuits that change state depending upon the actual signals being applied to their inputs at that time, Sequential Logic circuits have some form of inherent "Memory" built in to them as they are able to take into account their previous input state as well as those actually present, a sort of "before" and "after" effect is involved with sequential logic circuits.



In other words, the output state of a "sequential logic circuit" is a function of the following three states, the "present input", the "past input" and/or the "past output". Sequential Logic circuits remember these conditions and stay fixed in their current state until the next clock signal changes one of the states, giving sequential logic circuits "Memory".

Sequential logic circuits are generally termed as two state or Bistable devices which can have their output or outputs set in one of two basic states, a logic level "1" or a logic level "0" and will remain "latched" (hence the name latch) indefinitely in this current state or condition until some other input trigger pulse or signal is applied which will cause the bistable to change its state once again.

The word "Sequential" means that things happen in a "sequence", one after another and in Sequential Logic circuits, the actual clock signal determines when things will happen next. Simple sequential logic circuits can be constructed from standard Bistable circuits such as: Flip-flops, Latches and Counters and which themselves can be made by simply connecting together universal NAND Gates and/or NOR Gates in a particular combinational way to produce the required sequential circuit.

Flip-Flop

In electronics, a flip-flop or latch is a circuit that has two stable states and can be used to store state information. Flip-flops and latches are used as data storage elements. A flip-flop stores a single *bit* (binary digit) of data; one of its two states represents a "one" and the other represents a "zero". Such data storage can be used for storage of *state*, and such a circuit is described as sequential logic. When used in a finite-state machine, the output and next state depend not only on its current input, but also on its current state (and hence, previous inputs). It can also be used for counting of pulses, and for synchronizing variably-timed input signals to some reference timing signal.

Flip-flops can be either simple (transparent or opaque) or clocked (synchronous or edge-triggered). Although the term flip-flop has historically referred generically to both simple and clocked circuits, in modern usage it is common to reserve the term flip-flop exclusively for discussing clocked circuits; the simple ones are commonly called latches.

Using this terminology, a latch is level-sensitive, whereas a flip-flop is edge- sensitive. That is, when a latch is enabled it becomes transparent, while a flip flop's output only changes on a single type (positive going or negative going) of clock edge.

Flip-flop types

Flip-flops can be divided into common types

SR ("set-reset")

D ("data" or "delay")

T ("toggle")

JK types are the common ones.

3.2 SR Flip-Flop

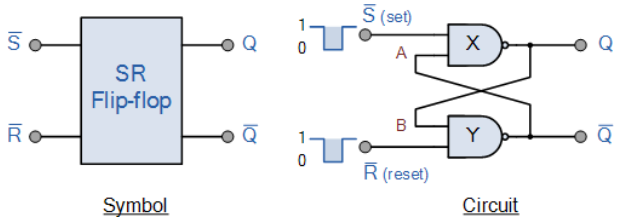
The SR flip-flop, also known as a SR Latch, can be considered as one of the most basic sequential logic circuit possible. This simple flip-flop is basically a one- bit memory bistable device that has two inputs, one which will “SET” the device (meaning the output = “1”), and is labelled S and another which will “RESET” the device (meaning the output = “0”), labelled R.

Then the SR description stands for “Set-Reset”. The reset input resets the flip-flop back to its original state with an output Q that will be either at a logic level “1” or logic “0” depending upon this set/reset condition.

A basic NAND gate SR flip-flop circuit provides feedback from both of its outputs back to its opposing inputs and is commonly used in memory circuits to store a single data bit. Then the SR flip-flop actually has three inputs, Set, Reset and its current output Q relating to it’s

current state or history. The term “Flip- flop” relates to the actual operation of the device, as it can be “flipped” into one logic Set state or “flopped” back into the opposing logic Reset state.

The Basic SR Flip-flop



The Set State

Consider the circuit shown above. If the input R is at logic level “0” (R = 0) and input S is at logic level “1” (S = 1), the NAND gate Y has at least one of its inputs at logic “0” therefore, its output Q must be at a logic level “1” (NAND Gate principles). Output Q is also fed back to input “A” and so both inputs to NAND gate X are at logic level “1”, and therefore its output Q must be at logic level “0”.

Again NAND gate principals. If the reset input R changes state, and goes HIGH to logic “1” with S remaining HIGH also at logic level “1”, NAND gate Y inputs are now R = “1” and B = “0”. Since one of its inputs is still at logic level “0” the output at Q still remains HIGH at logic level “1” and there is no change of state. Therefore, the flip-flop circuit is said to be “Latched” or “Set” with Q = “1” and Q = “0”.

Reset State

In this second stable state, Q is at logic level “0”, (not Q = “0”) its inverse output at Q is at logic level “1”, (Q = “1”), and is given by R = “1” and S = “0”. As gate X has one of its inputs at logic “0” its output Q must equal logic level “1” (again NAND gate principles). Output Q is fed back to input “B”, so both inputs to NAND gate Y are at logic “1”, therefore, Q = “0”.

If the set input, S now changes state to logic “1” with input R remaining at logic “1”, output Q still remains LOW at logic level “0” and there is no change of state. Therefore, the flip-flop circuits “Reset” state has also been latched and we can define this “set/reset” action in the following truth table.

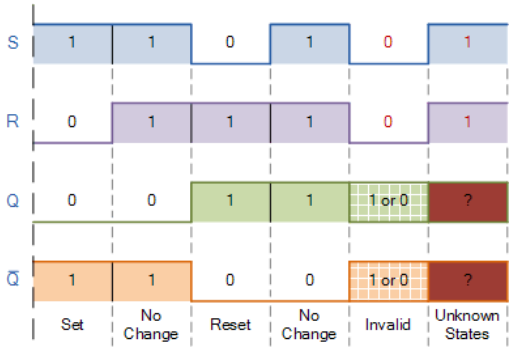
State	S	R	Q		Description
Set	1	0	0	1	Set $\overline{Q} \gg 1$
	1	1	0	1	no change
Reset	0	1	1	0	Reset $\overline{Q} \gg 0$
	1	1	1	0	no change
Invalid	0	0	1	1	Invalid Condition

Truth Table for this Set-Reset Function

It can be seen that when both inputs S = "1" and R = "1" the outputs Q and Q can be at either logic level "1" or "0", depending upon the state of the inputs S or R BEFORE this input condition existed. Therefore the condition of S = R = "1" does not change the state of the outputs Q and Q.

However, the input state of S = "0" and R = "0" is an undesirable or invalid condition and must be avoided. The condition of S = R = "0" causes both outputs Q and Q to be HIGH together at logic level "1" when we would normally want Q to be the inverse of Q. The result is that the flip-flop looses control of Q and Q, and if the two inputs are now switched "HIGH" again after this condition to logic "1", the flip-flop becomes unstable and switches to an unknown data state based upon the unbalance as shown in the following switching diagram.

S-R Flip-flop Switching Diagram



This unbalance can cause one of the outputs to switch faster than the other resulting in the flip-flop switching to one state or the other which may not be the required state and data corruption will exist. This unstable condition is generally known as its Meta-stable state.

Then, a simple NAND gate SR flip-flop or NAND gate SR latch can be set by applying a logic "0", (LOW) condition to its Set input and reset again by then applying a logic "0" to its Reset input. The SR flip-flop is said to be in an "invalid" condition (Meta-stable) if both the set and reset inputs are activated simultaneously.

Latch Flip Flop

The R-S (Reset Set) flip flop is the simplest flip flop of all and easiest to understand. It is basically a device which has two outputs one output being the inverse or complement of the other, and two inputs. A pulse on one of the inputs to take on a particular logical state. The outputs will then remain in this state until a similar pulse is applied to the other input. The two inputs are called the Set and Reset input (sometimes called the preset and clear inputs).

Such flip flop can be made simply by cross coupling two inverting gates either NAND or NOR gate could be used Figure 1(a) shows on RS flip flop using NAND gate and Figure 1(b) sh ows the same circuit using NOR gate.

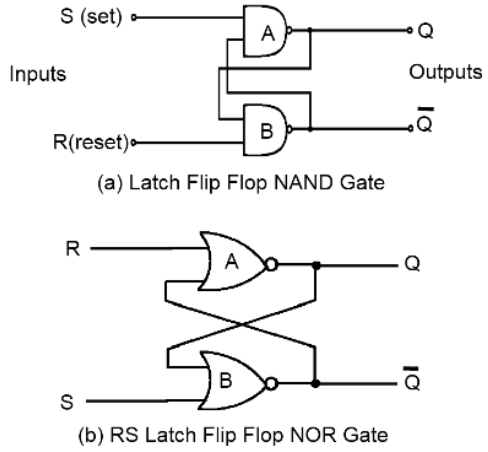


Figure 1: Latch R-S Flip Flop Using NAND and NOR Gates

To describe the circuit of Figure 1(a), assume that initially both R and S are at the logic 1 state and that output is at the logic 0 state.

Now, if Q = 0 and R = 1, then these are the states of inputs of gate B, therefore the outputs of gate B is at 1 (making it the inverse of Q i.e. 0). The output of gate B is connected to an input of gate A so if S = 1, both inputs of gate A are at the logic 1 state. This means that the output of gate A must be 0 (as was originally specified). In other words, the 0 state at Q is continuously disabling gate B so that

any change in R has no effect. Also the 1 state at $\overline{Q}$ is continuously enabling gate A so that any change S will be transmitted through to Q. The above conditions constitute one of the stable states of the device referred to as the Reset state since Q = 0.

Now suppose that the R-S flip flop in the Reset state, the S input goes to 0. The output of gate A i.e. Q will go to 1 and with Q = 1 and R = 1, the output of gates B ($\overline{Q}$) will go to 0 with $\overline{Q}$ now 0 gate A is disabled keeping Q at 1. Consequently, when S returns to the 1 state it has no effect on the flip flop whereas a change in R will cause a change in the output of gate B. The above conditions constitute the other stable state of the device, called the Set state since Q = 1. Note that the change of the state of S from 1 to 0 has caused the flip flop to change from the Reset state to the Set state.

There is another input condition which has not yet been considered. That is when both the R and S inputs are taken to the logic state 0. When this happens both Q and $\overline{Q}$ will be forced to 1 and will remain so far as long as R and S are kept at 0. However when both inputs return to 1 there is no way of knowing whether the flip flop will latch in the Reset state or the Set state. The condition is said to be indeterminate because of this indeterminate state great care must be taken when using R-S flip flop to ensure that both inputs are not instructed simultaneously.

Initial Conditions	Inputs (Pulsed)		Final Output	
Q	S	R	Q	$\overline{Q}$
1	0	0	indeterminate	
1	0	1	1	0
1	1	0	0	1
1	1	1	1	0
0	0	0	indeterminate	
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1

Table 1: The truth table for the NAND R-S flip flop

Table 2: Simple NAND R-S Flip Flop Truth Table

S	R	Q
0	0	indeterminate
0	1	Set (1)
1	0	Reset(0)
1	1	No Change

Table 3: NOR Gate R-S Flip Flop Truth Table

S	R	Q
0	0	No Change
0	1	Reset (0)
1	0	Set (1)
1	1	Indeterminate

Clocked RS Flip Flop

The RS latch flip flop required the direct input but no clock. It is very use full to add clock to control precisely the time at which the flip flop changes the state of its output.

In the clocked R-S flip flop the appropriate levels applied to their inputs are blocked till the receipt of a pulse from an other source called clock. The flip flop changes state only when clock pulse is applied depending upon the inputs. The basic circuit is shown in Figure 2. This circuit is formed by adding two AND gates at inputs to the R-S flip flop. In addition to control inputs Set (S) and Reset (R), there is a clock input (C) also.

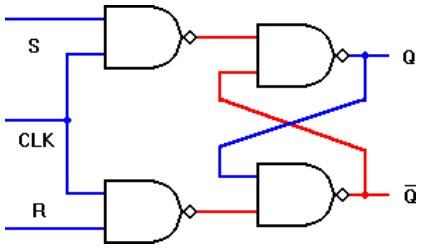


Figure 2: Clocked RS Flip Flop

Table 4: The truth table for the Clocked R-S flip flop

Initial Conditions	Inputs (Pulsed)		Final Output
Q	S	R	Q (t + 1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	indeterminate
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	indeterminate

S	R	Q
0	0	No Change
0	1	Reset (0)
1	0	Set (1)
1	1	Indeterminate

Table 5: Excitation table for R-S Flip Flop

3.3 D FLIP FLOP

A D type (Data or delay flip flop) has a single data input in addition to the clock input as shown in Figure 3.

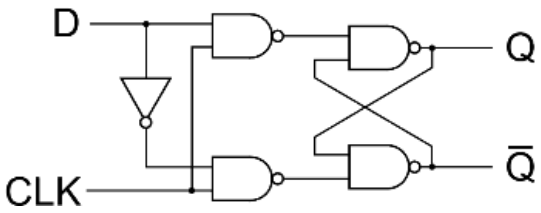


Figure 3: D Flip Flop

Basically, such type of flip flop is a modification of clocked RS flip flop gates from a basic Latch flip flop and NOR gates modify it in to a clock RS flip flop. The D input goes directly to S input and its complement through NOT gate, is applied to the R input.

This kind of flip flop prevents the value of D from reaching the output until a clock pulse occurs. The action of circuit is straight forward as follows.

When the clock is low, both AND gates are disabled, there fore D can change values with out affecting the value of Q. On the other hand, when the clock is high, both AND gates are enabled. In this case, Q is forced equal to D when the clock again goes low, Q retains or stores the last value of D. The truth table for such a flip flop is as given below in table 6.

S	R	Q(t + 1)
0	0	0
0	1	1
1	0	0
1	1	1

Table 6: Truth table for D Flip Flop

S	Q
0	0
1	1

Table 7: Excitation table for D Flip Flop

3.4 JK FLIP FLOP

One of the most useful and versatile flip flop is the JK flip flop the unique features of a JK flip flop are:

If the J and K input are both at 1 and the clock pulse is applied, then the output will change state, regardless of its previous condition.

If both J and K inputs are at 0 and the clock pulse is applied there will be no change in the output. There is no indeterminate condition, in the operation of JK flip flop i.e. it has no ambiguous state. The circuit diagram for a JK flip flop is shown in Figure 4.

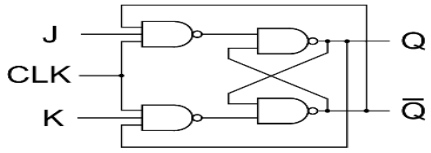


Figure 4: JK Flip Flop

When J = 0 and K = 0

These J and K inputs disable the NAND gates, therefore clock pulse have no effect on the flip flop. In other words, Q returns it last value.

When J = 0 and K = 1,

The upper NAND gate is disabled the lower NAND gate is enabled if Q is 1 therefore, flip flop will be reset (Q = 0 , $\overline{Q}$ =1)if not already in thatstate.

When J = 1 and K = 0

The lower NAND gate is disabled and the upper NAND gate is enabled if $\overline{Q}$ is at 1, As a result we will be able to set the flip flop (Q = 1, $\overline{Q}$ = 0) if not alreadyset

When J = 1 and K = 1

If Q = 0 the lower NAND gate is disabled the upper NAND gate is enabled. This will set the flip flop and hence Q will be 1. On the other hand if Q = 1, the lower NAND gate is enabled and flip flop will be reset and hence Q will be 0. In other words , when J and K are both high, the clock pulses cause the JK flip flop to toggle. Truth table for JK flip flop is shown in table 8.

Table 8: The truth table for the JK flip flop

Initial Conditions	Inputs (Pulsed)		Final Output
Q	S	R	Q (t + 1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

S	R	Q
0	0	No Change
0	1	0
1	0	0
1	1	Toggle

Table 6: Excitation table for JK Flip Flop

3.5 T FLIP FLOP

A method of avoiding the indeterminate state found in the working of RS flip flop is to provide only one input (the T input) such, flip flop acts as a toggle switch. Toggle means to change in the previous stage i.e. switch to opposite state. It can be constructed from clocked RS flip flop be incorporating feedback from output to input as shown in Figure 5.

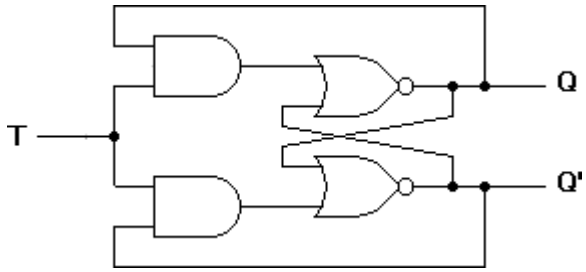


Figure 5: T Flip Flop

Such a flip flop is also called toggle flip flop. In such a flip flop a train of extremely narrow triggers drives the T input each time one of these triggers, the output of the flip flop changes stage. For instance Q equals 0 just before the trigger. Then the upper AND gate is enable and the lower AND gate is disabled. When the trigger arrives, it results in a high S input.

This sets the Q output to 1. When the next trigger appears at the point T, the lower AND gate is enabled and the trigger passes through to the R input this forces the flip flop to reset.

Since each incoming trigger is alternately changed into the set and reset inputs the flip flop toggles. It takes two triggers to produce one cycle of the output waveform. This means the output has half the frequency of the input stated another way, a T flip flop divides the input frequency by two. Thus such a circuit is also called a divide by two circuits.

A disadvantage of the toggle flip flop is that the state of the flip flop after a trigger pulse has been applied is only known if the previous state is known. The truth table for a T flip flop is as given table 7.

Table 7: Truth table for T Flip Flop

Q_n	T	$Q_n + 1$
0	0	0
0	1	1
1	0	1
1	1	0

Table 8: Excitation table for T Flip Flop

T	Q
0	Q_n
1	$\overline{Q}_n$

Generally T flip flop ICs are not available. It can be constructed using JK, RS or D flip flop. Figure 6 shows the relation of T flip flop using JK flip flop.

Figure 6: T Flip Flop Using JK Flip Flop

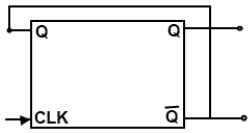


Figure 7: D-type Flip Flop connected as toggle stage

A D-type flip flop may be modified by external connection as a T-type stage as shown in Figure 7. Since the Q logic is used as D-input the opposite of the Q output is transferred into the stage each clock pulse. Thus the stage having Q - 0 transistors $\overline{Q}$ = 1, Providing a toggle action, if the stage had Q = 1 the clock pulse would result in Q = 0 being transferred, again providing the toggle operation. The D-type flip flop connected as in Figure 6 will thus operate as a T-type stage, complementing each clock pulse.

Master Slave Flip Flop

Figure 8 shows the schematic diagram of master sloave J-K flip flop

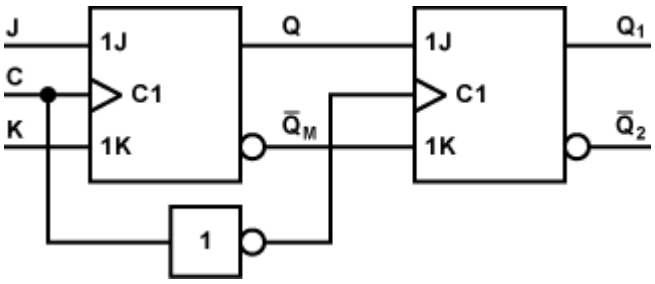


Figure 8: Master Slave JK Flip Flop

A master slave flip flop contains two clocked flip flops. The first is called master and the second slave. When the clock is high the master is active. The output of the master is set or reset according to the state of the input. As the slave is inactive during this period its output remains in the previous state. When clock becomes low the output of the slave flip flop changes because it become active during low clock period. The final output of master slave flip flop is the output of the slave flip flop. So the output of master slave flip flop is available at the end of a clock pulse.

3.6 SHIFT REGISTER

In digital circuits, a shift register is a cascade of flip flops, sharing the same clock, in which the output of each flip-flop is connected to the "data" input of the next flip-flop in the chain, resulting in a circuit that shifts by one position the "bit array" stored in it, "shifting in" the data present at its input and 'shifting out' the last bit in the array, at each transition of the clock input.

What is Shift Register:

Shift Registers are sequential logic circuits, capable of storage and transfer of data. They are made up of Flip Flops which are connected in such a way that the output of one flip flop could serve as the input of the other flip-flop, depending on the type of shift registers being created.

Types of Shift Registers

Shift registers are categorized into types majorly by their mode of operation, either serial or parallel. There are six (6) basic types of shift registers which are listed below although some of them can be further divided based on direction of data flow either shift right or shift left.

Serial in – Serial out Shift Register (SISO)

Serial In – Parallel out shift Register (SIPO)

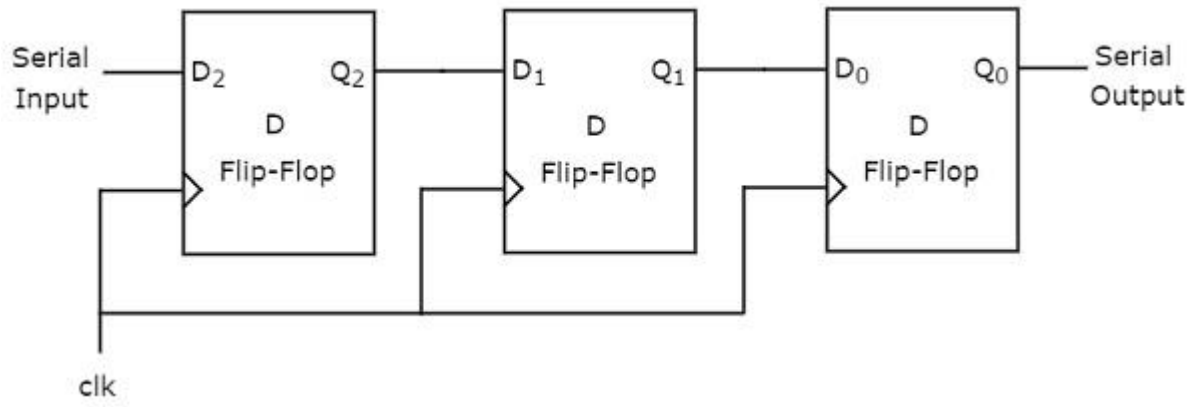
Parallel in – Parallel out Shift Register (PIPO)

Parallel in – Serial out Shift Register (PISO)

Bidirectional Shift Registers

Serial In − Serial Out SISO Shift Register

The shift register, which allows serial input and produces serial output is known as Serial In – Serial Out SISOSISO shift register. The block diagram of 3-bit SISO shift register is shown in the following figure.



This block diagram consists of three D flip-flops, which are **cascaded**. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can send the bits serially from the input of left most D flip-flop. Hence, this input is also called as **serial input**. For every positive edge triggering of clock signal, the data shifts from one stage to the next. So, we can receive the bits serially from the output of right most D flip-flop. Hence, this output is also called as **serial output**.

Example

Let us see the working of 3-bit SISO shift register by sending the binary information **“011”** from LSB to MSB serially at the input.

Assume, initial status of the D flip-flops from leftmost to rightmost is Q2Q1Q0=000. We can understand the working of 3-bit SISO shift register from the following table.

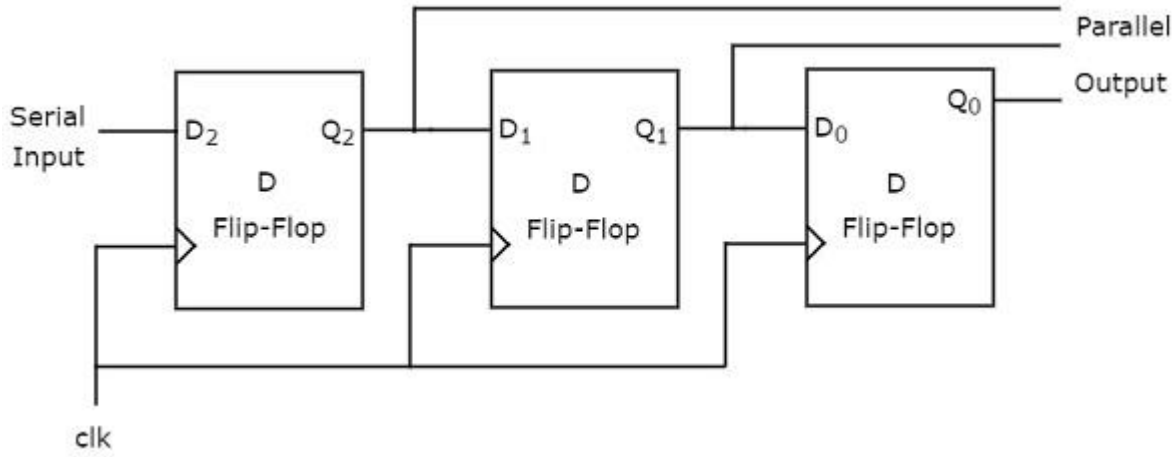
No of positive edge of Clock	Serial Input	Q_2	Q_1	Q_0
0	-	0	0	0
1	1 LSB	1	0	0
2	1	1	1	0
3	0 MSB	0	1	1 LSB
4	-	-	0	1
5	-	-	-	0 MSB

The initial status of the D flip-flops in the absence of clock signal is Q2Q1Q0=000. Here, the serial output is coming from Q0. So, the LSB 11 is received at 3rd positive edge of clock and the MSB 0 is received at 5th positive edge of clock.

Therefore, the 3-bit SISO shift register requires five clock pulses in order to produce the valid output. Similarly, the **N-bit SISO shift register** requires **2N-1** clock pulses in order to shift 'N' bit information.

Serial In - Parallel Out SIPOSIPO Shift Register

The shift register, which allows serial input and produces parallel output is known as Serial In – Parallel Out SIPO SIPO shift register. The block diagram of 3-bit SIPO shift register is shown in the following figure.



This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can send the bits serially from the input of left most D flip-flop. Hence, this input is also called as serial input. For every positive edge triggering of clock signal, the data shifts from one stage to the next. In this case, we can access the outputs of each D flip-flop in parallel. So, we will get parallel outputs from this shift register.

Example

Let us see the working of 3-bit SIPO shift register by sending the binary information "011" from LSB to MSB serially at the input.

Assume, initial status of the D flip-flops from leftmost to rightmost is Q2Q1Q0=000. Here, Q2Q2 & Q0Q0 are MSB & LSB respectively. We can understand the working of 3-bit SIPO shift register from the following table.

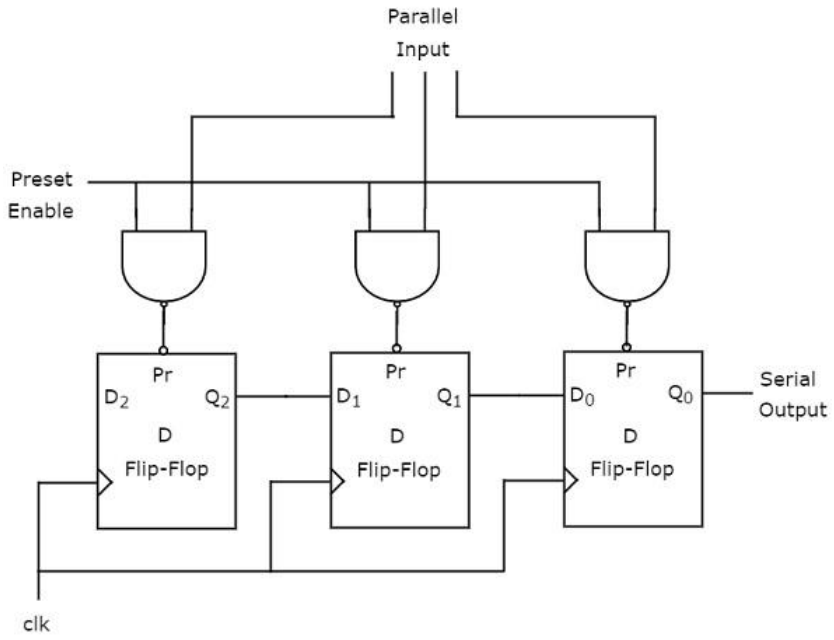
No of positive edge of Clock	Serial Input	Q_2MSBMSB	Q_1	Q_0LSBLSB
0	-	0	0	0
1	1LSBLSB	1	0	0
2	1	1	1	0
3	0MSBMSB	0	1	1

The initial status of the D flip-flops in the absence of clock signal is Q2Q1Q0=000. The binary information "011" is obtained in parallel at the outputs of D flip-flops for third positive edge of clock.

So, the 3-bit SIPO shift register requires three clock pulses in order to produce the valid output. Similarly, the N-bit SIPO shift register requires N clock pulses in order to shift 'N' bit information.

Parallel In − Serial Out PISOPISO Shift Register

The shift register, which allows parallel input and produces serial output is known as Parallel In − Serial Out PISOPISO shift register. The block diagram of 3-bit PISO shift register is shown in the following figure.



This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can apply the parallel inputs to each D flip-flop by making Preset Enable to 1. For every positive edge triggering of clock signal, the data shifts from one stage to the next. So, we will get the serial output from the right most D flip-flop.

Example

Let us see the working of 3-bit PISO shift register by applying the binary information "011" in parallel through preset inputs.

Since the preset inputs are applied before positive edge of Clock, the initial status of the D flip-flops from leftmost to rightmost will be Q2Q1Q0=011. We can understand the working of 3-bit PISO shift register from the following table.

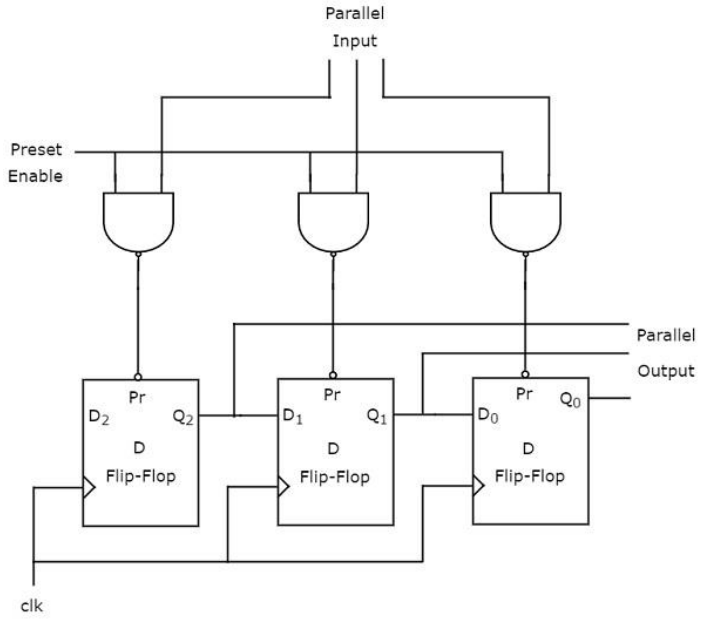
No of positive edge of Clock	Q_2	Q_1	Q_0
0	0	1	1LSBLSB
1	-	0	1
2	-	-	0LSBLSB

Here, the serial output is coming from Q0. So, the LSB 1 is received before applying positive edge of clock and the MSB 00 is received at 2^{nd} positive edge of clock.

Therefore, the 3-bit PISO shift register requires two clock pulses in order to produce the valid output. Similarly, the N-bit PISO shift register requires N-1 clock pulses in order to shift 'N' bit information.

Parallel In - Parallel Out PIPOPIPO Shift Register

The shift register, which allows parallel input and produces parallel output is known as Parallel In − Parallel Out PIPO PIPO shift register. The block diagram of 3-bit PIPO shift register is shown in the following figure.



This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can apply the parallel inputs to each D flip-flop by making Preset Enable to 1. We can apply the parallel inputs through preset or clear. These two are asynchronous inputs. That means, the flip-flops produce the corresponding outputs, based on the values of asynchronous inputs. In this case, the effect of outputs is independent of clock transition. So, we will get the parallel outputs from each D flip-flop.

Example

Let us see the working of 3-bit PIPO shift register by applying the binary information “011” in parallel through preset inputs.

Since the preset inputs are applied before positive edge of Clock, the initial status of the D flip-flops from leftmost to rightmost will be Q2Q1Q0=011. So, the binary information “011” is obtained in parallel at the outputs of D flip-flops before applying positive edge of clock.

Therefore, the 3-bit PIPO shift register requires zero clock pulses in order to produce the valid output. Similarly, the N-bit PIPO shift register doesn’t require any clock pulse in order to shift ‘N’ bit information.

3.7 COUNTERS

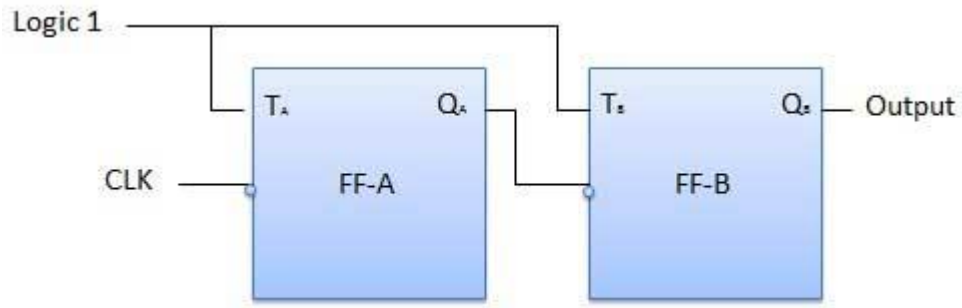
Counter is a sequential circuit. A digital circuit which is used for a counting pulses is known counter. Counter is the widest application of flip-flops. It is a group of flip-flops with a clock signal applied. Counters are of two types.

- Asynchronous or ripple counters.
- Synchronous counters

3.8 RIPPLE COUNTER

The logic diagram of a 2-bit ripple up counter is shown in figure. The toggle (T) flip-flop are being used. But we can use the JK flip-flop also with J and K connected permanently to logic 1. External clock is applied to the clock input of flip-flop A and Q_A output is applied to the clock input of the next flip-flop i.e. FF-B.

Logical Diagram



Operation

S.N.	Condition	Operation
1	**Initially let both the FFs be in the reset state**	Q_BQ_A = 00 initially
2	**After 1st negative clock edge**	As soon as the first negative clock edge is applied, FF-A will toggle and Q_A will be equal to 1. Q_A is connected to clock input of FF-B. Since Q_A has changed from 0 to 1, it is treated as the positive clock edge by FF-B. There is no change in Q_B because FF-B is a negative edge triggered FF. Q_BQ_A = 01 after the first clock pulse.
3	**After 2nd negative clock edge**	On the arrival of second negative clock edge, FF-A toggles again and Q_A = 0. The change in Q_A acts as a negative clock edge for FF-B. So it will also toggle, and Q_B will be 1. Q_BQ_A = 10 after the second clock pulse.

4	**After 3rd negative clock edge**	On the arrival of 3rd negative clock edge, FF-A toggles again and Q_A become 1 from 0. Since this is a positive going change, FF-B does not respond to it and remains inactive. So Q_B does not change and continues to be equal to 1. Q_BQ_A = 11 after the third clock pulse.
5	**After 4th negative clock edge**	On the arrival of 4th negative clock edge, FF-A toggles again and Q_A becomes 1 from 0. This negative change in Q_A acts as clock pulse for FF-B. Hence it toggles to change Q_B from 1 to 0. Q_BQ_A = 00 after the fourth clock pulse.

Truth Table

Clock	Counter output		State number	Deciimal Counter output
	Q_B	Q_A		
Initially	0	0	—	0
1st	0	1	1	1
2nd	1	0	2	2
3rd	1	1	3	3
4th	0	0	4	0

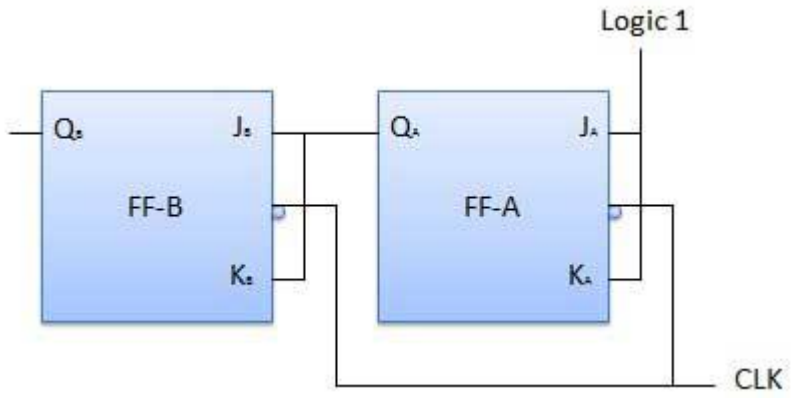
3.9 SYNCHRONOUS COUNTER

If the "clock" pulses are applied to all the flip-flops in a counter simultaneously, then such a counter is called as synchronous counter.

2-bit Synchronous up counter

The J_A and K_A inputs of FF-A are tied to logic 1. So FF-A will work as a toggle flip-flop. The J_B and K_B inputs are connected to Q_A.

Logical Diagram



Operation

S.N.	Condition	Operation
1	Initially let both the FFs be in the reset state	Q_BQ_A = 00 initially.
2	After 1st negative clock edge	As soon as the first negative clock edge is applied, FF-A will toggle and Q_A will change from 0 to 1. But at the instant of application of negative clock edge, Q_A , $J_B = K_B = 0$. Hence FF-B will not change its state. So Q_B will remain 0. Q_BQ_A = 01 after the first clock pulse.
3	After 2nd negative clock edge	On the arrival of second negative clock edge, FF-A toggles again and Q_A changes from 1 to 0. But at this instant Q_A was 1. So $J_B = K_B = 1$ and FF-B will toggle. Hence Q_B changes from 0 to 1. Q_BQ_A = 10 after the second clock pulse.
4	After 3rd negative clock edge	On application of the third falling clock edge, FF-A will toggle from 0 to 1 but there is no change of state for FF-B. Q_BQ_A = 11 after the third clock pulse.

5	After 4th negative clock edge	On application of the next clock pulse, Q_A will change from 1 to 0 as Q_B will also change from 1 to 0. Q_BQ_A = 00 after the fourth clock pulse.

References :

1. Moris Mano, "Digital Computer Fundamentals" TMH 3rd Edition
2. http://www.tutorialspoint.com/computer_logical_organization/number_system_conversion.htm
3. http://www.electronics-tutorials.ws/binary/signed-binary-numbers.html
4. HAMMING, R. →. "Error Detecting and Error Correcting Codes." Bell System Tech. Jour., 29 (1950): 147–160.
5. A.P GODSE,D.A.GODSE ."Digital Systems".Technical Publications.Pune.
6. http://www.tutorialspoint.com/computer_logical_organization/binary_codes.htm
7. http://nptel.ac.in/courses/Webcourse-contents/IIScBANG/Digital%20Systems/Digital%20Systems.pdf
8. Digital Logic Circuits by D.A.Godse A.P.Godse

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

UNIT - IV

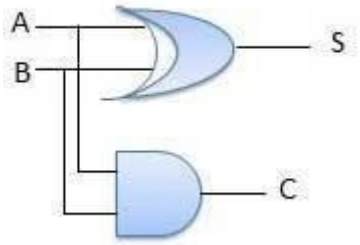
Digital Computer Fundamentals-SBSA1101

4.1 ADDERS

Half adder

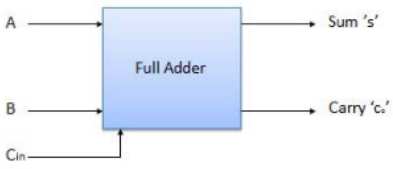
Half adder is a combinational logic circuit with two inputs and two outputs. The half adder circuit is designed to add two single bit binary number A and B. It is the basic building block for addition of two single bit numbers. This circuit has two outputs carry and sum.

Inputs		Output
A	B	S C
0	0	0 0
0	1	1 0
1	0	1 0
1	1	0 1

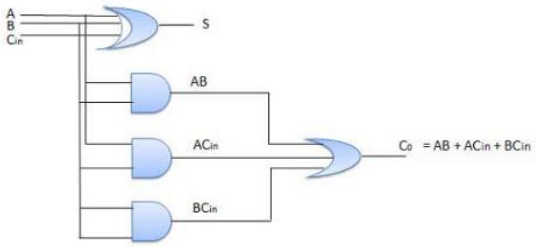


Full Adder

Full adder is developed to overcome the drawback of Half Adder circuit. It can add two one-bit numbers A and B, and carry c. The full adder is a three input and two output combinationalcircuit.



Inputs			Output	
A	B	C_{in}	S	C_o
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

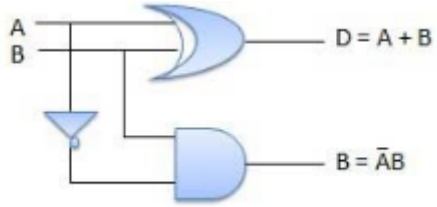


4.2 SUBTRACTORS

Half Subtractors

Half subtractor is a combination circuit with two inputs and two outputs (difference and borrow). It produces the difference between the two binary bits at the input and also produces an output (Borrow) to indicate if a 1 has been borrowed. In the subtraction (A-B), A is called as Minuend bit and B is called as Subtrahend bit.

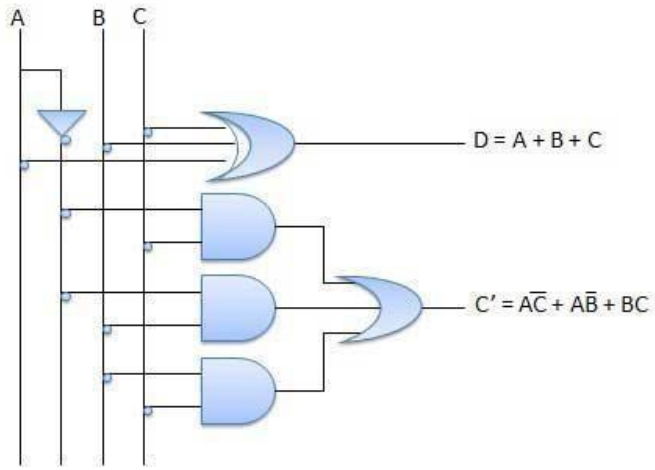
Inputs		Output	
A	B	(A – B)	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



Full Subtractors

The disadvantage of a half subtractor is overcome by full subtractor. The full subtractor is a combinational circuit with three inputs A,B,C and two output D and C'. A is the 'minuend', B is 'subtrahend', C is the 'borrow' produced by the previous stage, D is the difference output and C' is the borrow output.

Inputs			Output	
A	B	C	(A-B-C)	C'
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



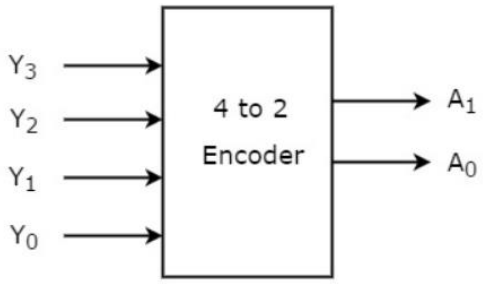
The circuit for subtracting A - B consists of an adder with inverters placed between each data input B and the corresponding input of the full adder. The input carry C0 must be equal to 1 when subtraction is performed. The operation thus performed becomes A, plus the 1's complement of B, plus 1. This is equal to A plus the 2's complement of B.

4.3 ENCODER

An Encoder is a combinational circuit that performs the reverse operation of Decoder. It has maximum of 2^n input lines and 'n' output lines. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes 2^n input lines with 'n' bits. It is optional to represent the enable signal in encoders.

4 to 2 Encoder

Let 4 to 2 Encoder has four inputs Y_3, Y_2, Y_1 & Y_0 and two outputs A_1 & A_0. The block diagram of 4 to 2 Encoder is shown in the following figure.



At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output. The Truth table of 4 to 2 encoder is shown below.

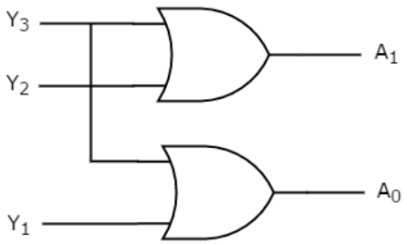
Inputs				Outputs	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

From Truth table, we can write the Boolean functions for each output as

$$A1=Y3+Y2$$

$$A0=Y3+Y1$$

We can implement the above two Boolean functions by using two input OR gates. The circuit diagram of 4 to 2 encoder is shown in the following figure.



Priority Encoder

4 to 2 priority encoder has four inputs Y_3, Y_2, Y_1 & Y_0 and two outputs A_1 & A_0. Here, the input, Y_3 has the highest priority, whereas the input, Y_0 has the lowest priority. In this case, even if more than one input is '1' at the same time, the output will be the binarybinary code corresponding to the input, which is having higher priority.

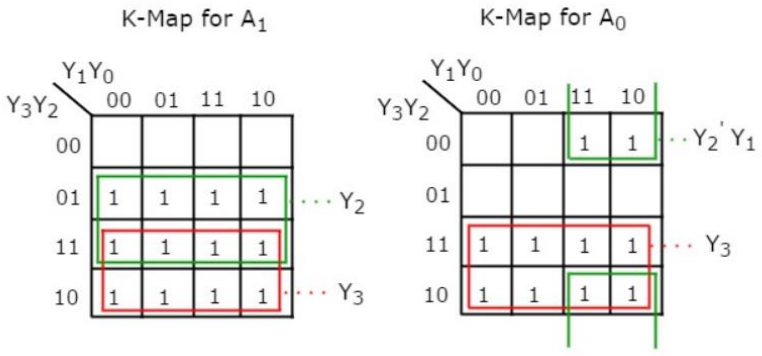
We considered one more output, V in order to know, whether the code available at outputs is valid or not.

- If at least one input of the encoder is '1', then the code available at outputs is a valid one. In this case, the output, V will be equal to 1.
- If all the inputs of encoder are '0', then the code available at outputs is not a valid one. In this case, the output, V will be equal to 0.

The Truth table of 4 to 2 priority encoder is shown below.

Inputs				Outputs		
Y_3	Y_2	Y_1	Y_0	A_1	A_0	V
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

Use 4 variable K-maps for getting simplified expressions for each output.



The simplified Boolean functions are

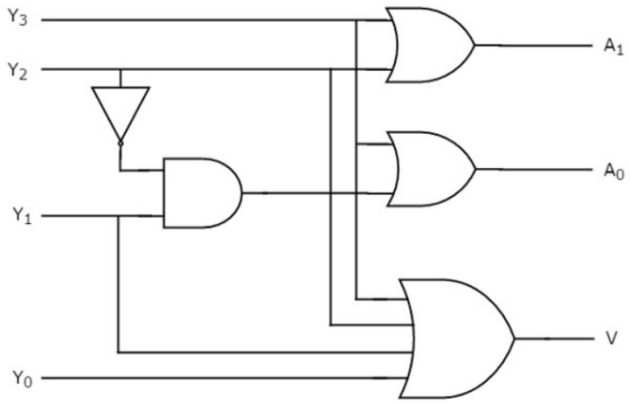
$$A1=Y3+Y2A1=Y3+Y2$$

$$A0=Y3+Y2'Y1$$

milarly, we will get the Boolean function of output, V as

$$V=Y3+Y2+Y1+Y0V=Y3+Y2+Y1+Y0$$

We can implement the above Boolean functions using logic gates. The circuit diagram of 4 to 2 priority encoder is shown in the following figure.



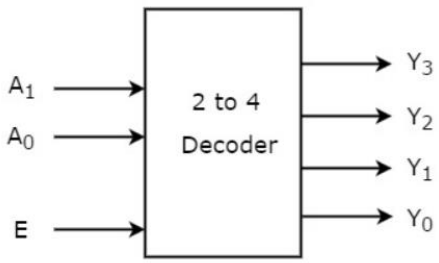
The above circuit diagram contains two 2-input OR gates, one 4-input OR gate, one 2input AND gate & an inverter. Here AND gate & inverter combination are used for producing a valid code at the outputs, even when multiple inputs are equal to '1' at the same time. Hence, this circuit encodes the four inputs with two bits based on the priority assigned to each input.

4.4 DECODER

Decoder is a combinational circuit that has 'n' input lines and maximum of 2^n output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but the **min terms** of 'n' input variables lines lines, when it is enabled.

2 to 4 Decoder

Let 2 to 4 Decoder has two inputs A_1 & A_0 and four outputs Y_3, Y_2, Y_1 & Y_0. The block diagram of 2 to 4 decoder is shown in the following figure.



One of these four outputs will be '1' for each combination of inputs when enable, E is '1'. The Truth table of 2 to 4 decoder is shown below.

Enable	Inputs		Outputs			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

From Truth table, we can write the Boolean functions for each output as

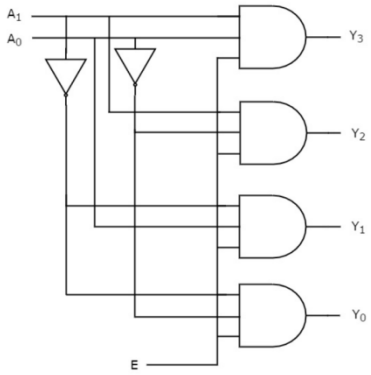
$$Y3=E.A1.A0Y3=E.A1.A0$$

$$Y2=E.A1.A0'Y2=E.A1.A0'$$

$$Y1=E.A1'.A0$$

$$Y0=E.A1'.A0'$$

Each output is having one product term. So, there are four product terms in total. We can implement these four product terms by using four AND gates having three inputs each & two inverters. The circuit diagram of 2 to 4 decoder is shown in the following figure.



Therefore, the outputs of 2 to 4 decoder are nothing but the min terms of two input variables A_1 & A_0, when enable, E is equal to one. If enable, E is zero, then all the outputs of decoder will be equal to zero.

Similarly, 3 to 8 decoder produces eight min terms of three input variables A_2, A_1 & A_0 and 4 to 16 decoder produces sixteen min terms of four input variables A_3, A_2, A_1 & A_0.

Implementation of Higher-order Decoders

3 to 8 Decoder

In this section, let us implement 3 to 8 decoder using 2 to 4 decoders. We know that 2 to 4 Decoder has two inputs, A_1 & A_0 and four outputs, Y_3 to Y_0. Whereas, 3 to 8 Decoder has three inputs A_2, A_1 & A_0 and eight outputs, Y_7 to Y_0.We can find the number of lower order decoders required for implementing higher order decoder using the following formula.

Required number of lower order decoders= m2/m1

Where,

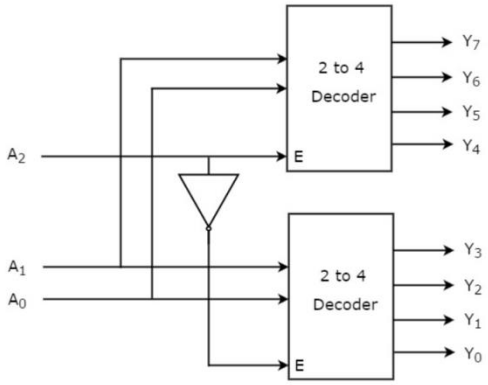
m1is the number of outputs of lower order decoder.

m2 is the number of outputs of higher order decoder.

Here, m1 = 4 and m2 = 8. Substitute, these two values in the above formula.

Required number of 2to4 decoders = 8/4

Therefore, we require two 2 to 4 decoders for implementing one 3 to 8 decoder. The block diagram of 3 to 8 decoder using 2 to 4 decoders is shown in the following figure.



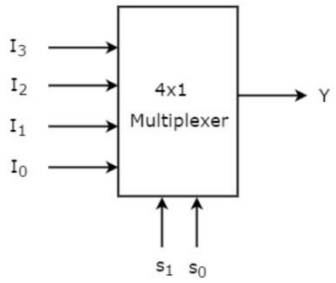
The parallel inputs A_1 & A_0 are applied to each 2 to 4 decoder. The complement of input A_2 is connected to Enable, E of lower 2 to 4 decoder in order to get the outputs, Y_3 to Y_0. These are the lower four min terms. The input, A_2 is directly connected to Enable, E of upper 2 to 4 decoder in order to get the outputs, Y_7 to Y_4. These are the higher four min terms.

4.5 MULTIPLEXER

Multiplexer is a combinational circuit that has maximum of 2^n data inputs, 'n' selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines.Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination will select only one data input. Multiplexer is also called as Mux.

4x1 Multiplexer

4x1 Multiplexer has four data inputs I_3, I_2, I_1 & I_0, two selection lines s_1 & s_0 and one output Y. The block diagram of 4x1 Multiplexer is shown in the following figure.



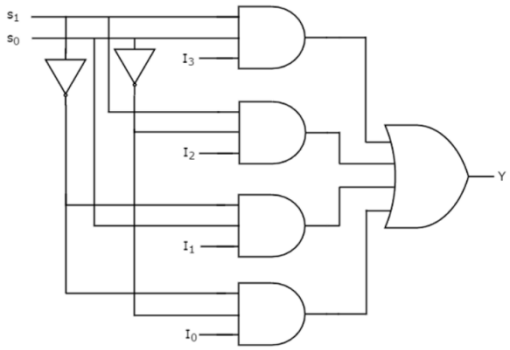
One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. Truth table of 4x1 Multiplexer is shown below.

Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

From Truth table, we can directly write the Boolean function for output, Y as

$$Y=S1'S0'I0+S1'S0I1+S1S0'I2+S1S0I3Y=S1'S0'I0+S1'S0I1+S1S0'I2+S1S0I3$$

We can implement this Boolean function using Inverters, AND gates & OR gate. The circuit diagram of 4x1 multiplexer is shown in the following figure.



we can easily understand the operation of the above circuit. Similarly, you can implement 8x1 Multiplexer and 16x1 multiplexer by following the same procedure.

Implementation of Higher-order Multiplexers

Now, let us implement the following two higher-order Multiplexers using lower-order Multiplexers.

- 8x1 Multiplexer
- 16x1 Multiplexer

8x1 Multiplexer

In this section, let us implement 8x1 Multiplexer using 4x1 Multiplexers and 2x1 Multiplexer. We know that 4x1 Multiplexer has 4 data inputs, 2 selection lines and one output. Whereas, 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output.So, we require two 4x1 Multiplexers in first stage in order to get the 8 data inputs. Since, each 4x1 Multiplexer produces one output, we require a 2x1 Multiplexer in second stage by considering the outputs of first stage as inputs and to produce the final output.

Let the 8x1 Multiplexer has eight data inputs I_7 to I_0, three selection lines s_2, s_1 & s0 and one output Y. The Truth table of 8x1 Multiplexer is shown below.

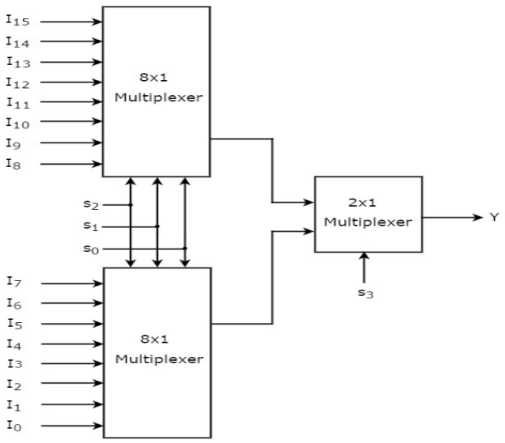
Selection Inputs			Output
S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

16x1 Multiplexer

In this section, let us implement 16x1 Multiplexer using 8x1 Multiplexers and 2x1 Multiplexer. We know that 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output. Whereas, 16x1 Multiplexer has 16 data inputs, 4 selection lines and one output. So, we require two 8x1 Multiplexers in first stage in order to get the 16 data inputs. Since, each 8x1 Multiplexer produces one output, we require a 2x1 Multiplexer in second stage by considering the outputs of first stage as inputs and to produce the final output.

Let the 16x1 Multiplexer has sixteen data inputs I_{15} to I_0, four selection lines s_3 to s_0 and one output Y. The Truth table of 16x1 Multiplexer is shown below. We can implement 16x1 Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 16x1 Multiplexer is shown in the following figure.

Selection Inputs				Output
S_3	S_2	S_1	S_0	Y
0	0	0	0	I_0
0	0	0	1	I_1
0	0	1	0	I_2
0	0	1	1	I_3
0	1	0	0	I_4
0	1	0	1	I_5
0	1	1	0	I_6
0	1	1	1	I_7
1	0	0	0	I_8
1	0	0	1	I_9
1	0	1	0	I_{10}
1	0	1	1	I_{11}
1	1	0	0	I_{12}
1	1	0	1	I_{13}
1	1	1	0	I_{14}
1	1	1	1	I_{15}



The same selection lines, s_2, s_1 & s_0 are applied to both 8x1 Multiplexers. The data inputs of upper 8x1 Multiplexer are I_{15} to I_8 and the data inputs of lower 8x1 Multiplexer are I_7 to I_0. Therefore, each 8x1 Multiplexer produces an output based on the values of selection lines, s_2, s_1 & s_0.

The outputs of first stage 8x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other selection line, s_3 is applied to 2x1 Multiplexer.

- If s_3 is zero, then the output of 2x1 Multiplexer will be one of the 8 inputs Is_7 to I_0 based on the values of selection lines s_2, s_1 & s_0.
- If s_3 is one, then the output of 2x1 Multiplexer will be one of the 8 inputs I_{15} to I_8 based on the values of selection lines s_2, s_1 & s_0.

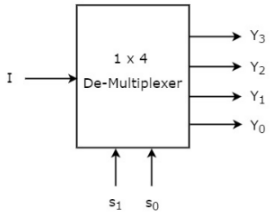
Therefore, the overall combination of two 8x1 Multiplexers and one 2x1 Multiplexer performs as one 16x1 Multiplexer.

4.6 De-MULTIPLEXER

De-Multiplexer is a combinational circuit that performs the reverse operation of Multiplexer. It has single input, 'n' selection lines and maximum of 2^n outputs. The input will be connected to one of these outputs based on the values of selection lines. Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination can select only one output. De-Multiplexer is also called as De-Mux.

1x4 De-Multiplexer

1x4 De-Multiplexer has one input I, two selection lines, s_1 & s_0 and four outputs Y_3, Y_2, Y_1 &Y_0. The block diagram of 1x4 De-Multiplexer is shown in the following figure.



The single input 'I' will be connected to one of the four outputs, Y_3 to Y_0 based on the values of selection lines s_1 & s0. The **Truth table** of 1x4 De-Multiplexer is shown below.

Selection Inputs		Outputs			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

From the above Truth table, we can directly write the Boolean functions for each output as

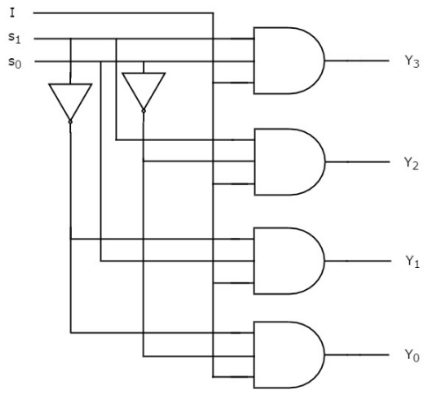
$$Y3=s1s0IY3=s1s0I$$

$$Y2=s1s0′IY2=s1s0′I$$

$$Y1=s1′s0IY1=s1′s0I$$

$$Y0=s1′s0′I$$

We can implement these Boolean functions using Inverters & 3-input AND gates. The circuit diagram of 1x4 De-Multiplexer is shown in the following figure.



We can easily understand the operation of the above circuit. Similarly, you can implement 1x8 De-Multiplexer and 1x16 De-Multiplexer by following the same procedure.

Implementation of Higher-order De-Multiplexers

Now, let us implement the following two higher-order De-Multiplexers using lower-order De-Multiplexers.

- 1x8 De-Multiplexer
- 1x16 De-Multiplexer

1x8 De-Multiplexer

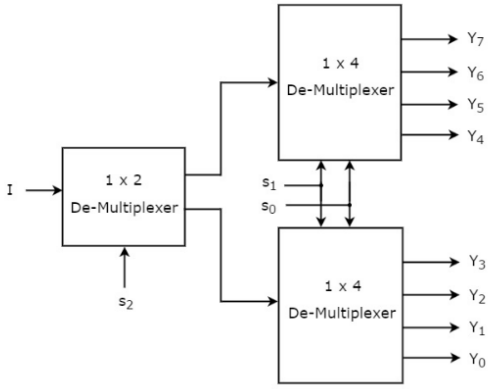
In this section, let us implement 1x8 De-Multiplexer using 1x4 De-Multiplexers and 1x2 De-Multiplexer. We know that 1x4 De-Multiplexer has single input, two selection lines and four outputs. Whereas, 1x8 De-Multiplexer has single input, three selection lines and eight outputs.

So, we require two 1x4 De-Multiplexers in second stage in order to get the final eight outputs. Since, the number of inputs in second stage is two, we require 1x2 DeMultiplexer in first stage so that the outputs of first stage will be the inputs of second stage. Input of this 1x2 De-Multiplexer will be the overall input of 1x8 De-Multiplexer.

Let the 1x8 De-Multiplexer has one input I, three selection lines s_2, s_1 & s_0 and outputs Y_7 to Y_0. The Truth table of 1x8 De-Multiplexer is shown below.

Selection Inputs			Outputs							
s_2	s_1	s_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	I
0	0	1	0	0	0	0	0	0	I	0
0	1	0	0	0	0	0	0	I	0	0
0	1	1	0	0	0	0	I	0	0	0
1	0	0	0	0	0	I	0	0	0	0
1	0	1	0	0	I	0	0	0	0	0
1	1	0	0	I	0	0	0	0	0	0
1	1	1	I	0	0	0	0	0	0	0

We can implement 1x8 De-Multiplexer using lower order Multiplexers easily by considering the above Truth table. The block diagram of 1x8 De-Multiplexer is shown in the following figure.

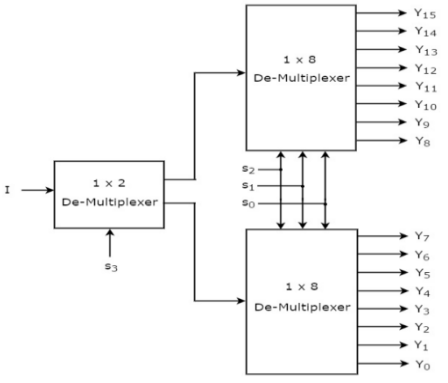


The common selection lines, s_1 & s_0 are applied to both 1x4 De-Multiplexers. The outputs of upper 1x4 De-Multiplexer are Y_7 to Y_4 and the outputs of lower 1x4 De-Multiplexer are Y_3 to Y_0.The other selection line, s_2 is applied to 1x2 De-Multiplexer. If s_2 is zero, then one of the four outputs of lower 1x4 De-Multiplexer will be equal to input, I based on the values of selection lines s_1 & s_0. Similarly, if s_2 is one, then one of the four outputs of upper 1x4 DeMultiplexer will be equal to input, I based on the values of selection lines s_1 & s_0.

1x16 De-Multiplexer

In this section, let us implement 1x16 De-Multiplexer using 1x8 De-Multiplexers and 1x2 De-Multiplexer. We know that 1x8 De-Multiplexer has single input, three selection lines and eight outputs. Whereas, 1x16 De-Multiplexer has single input, four selection lines and sixteen outputs.

So, we require two 1x8 De-Multiplexers in second stage in order to get the final sixteen outputs. Since, the number of inputs in second stage is two, we require 1x2 DeMultiplexer in first stage so that the outputs of first stage will be the inputs of second stage. Input of this 1x2 De-Multiplexer will be the overall input of 1x16 De-Multiplexer. Let the 1x16 De-Multiplexer has one input I, four selection lines s_3, s_2, s_1 & s_0 and outputs Y_{15} to Y_0. The block diagram of 1x16 De-Multiplexer using lower order Multiplexers is shown in the following figure.



The common selection lines s_2, s_1 & s_0 are applied to both 1x8 De-Multiplexers. The outputs of upper 1x8 De-Multiplexer are Y_{15} to Y_8 and the outputs of lower 1x8 DeMultiplexer are Y_7 to Y_0.The other selection line, s_3 is applied to 1x2 De-Multiplexer. If s_3 is zero, then one of the eight outputs of lower 1x8 De-Multiplexer will be equal to input, I based on the values of selection lines s_2, s_1 & s_0. Similarly, if s3 is one, then one of the 8 outputs of upper 1x8 De-Multiplexer will be equal to input, I based on the values of selection lines s_2, s_1 & s_0.

4.7 ROM

ROM stands for Read Only Memory. The memory from which we can only read but cannot write on it. This type of memory is non-volatile. The information is stored permanently in such memories during manufacture. A ROM stores such instructions that are required to start a computer. This operation is referred to as bootstrap. ROM chips are not only used in the computer but also in other electronic items like washing machine and microwave oven.

MROM (Masked ROM)

The very first ROMs were hard-wired devices that contained a pre-programmed set of data or instructions. These kind of ROMs are known as masked ROMs, which are inexpensive.

PROM (Programmable Read Only Memory)

PROM is read-only memory that can be modified only once by a user. The user buys a blank PROM and enters the desired contents using a PROM program. Inside the PROM chip, there are small fuses which are burnt open during programming. It can be programmed only once and is not erasable.

EPROM (Erasable and Programmable Read Only Memory)

EPROM can be erased by exposing it to ultra-violet light for a duration of up to 40 minutes. Usually, an EPROM eraser achieves this function. During programming, an electrical charge is trapped in an insulated gate region. The charge is retained for more than 10 years because the charge has no leakage path. For erasing this charge, ultra-violet light is passed through a quartz crystal window (lid). This exposure to ultra-violet light dissipates the charge. During normal use, the quartz lid is sealed with a sticker.

EEPROM (Electrically Erasable and Programmable Read Only Memory)

EEPROM is programmed and erased electrically. It can be erased and reprogrammed about ten thousand times. Both erasing and programming take about 4 to 10 ms (millisecond). In EEPROM, any location can be selectively erased and programmed. EEPROMs can be erased one byte at a time, rather than erasing the entire chip. Hence, the process of reprogramming is flexible but slow.

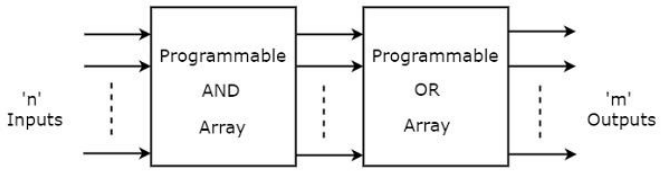
Advantages of ROM

The advantages of ROM are as follows −

- Non-volatile in nature
- Cannot be accidentally changed
- Cheaper than RAMs
- Easy to test
- More reliable than RAMs
- Static and do not require refreshing
- Contents are always known and can be verified

4.8 PLA

PLA is a programmable logic device that has both Programmable AND array & Programmable OR array. Hence, it is the most flexible PLD. The block diagram of PLA is shown in the following figure.



Here, the inputs of AND gates are programmable. That means each AND gate has both normal and complemented inputs of variables. So, based on the requirement, we can program any of those inputs. So, we can generate only the required product terms by using these AND gates.

Here, the inputs of OR gates are also programmable. So, we can program any number of required product terms, since all the outputs of AND gates are applied as inputs to each OR gate. Therefore, the outputs of PAL will be in the form of sum of products form.

Example

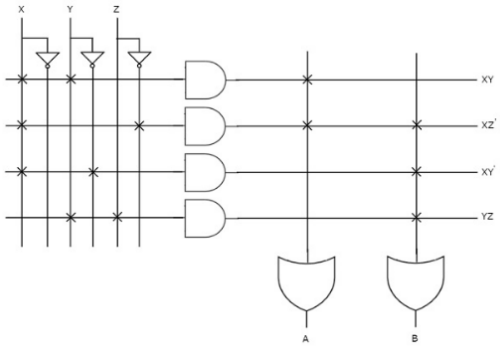
Let us implement the following Boolean functions using PLA.

$A=XY+XZ'$

$B=XY'+YZ+XZ'$

The given two functions are in sum of products form. The number of product terms present in the given Boolean functions A & B are two and three respectively. One product term, $Z'X$ is common in each function.

So, we require four programmable AND gates & two programmable OR gates for producing those two functions. The corresponding PLA is shown in the following figure.

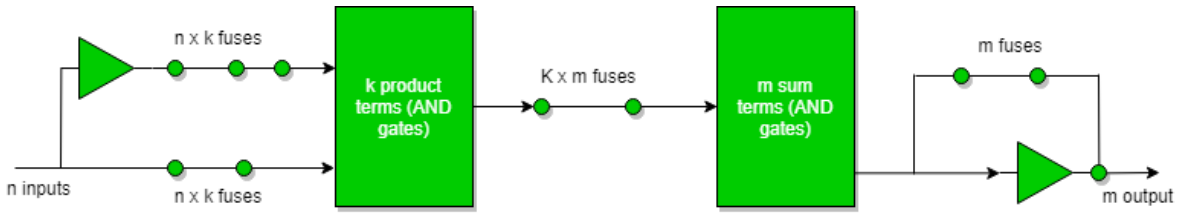


The programmable AND gates have the access of both normal and complemented inputs of variables. In the above figure, the inputs X, X′, Y, Y′, Z & Z′, are available at the inputs of each AND gate. So, program only the required literals in order to generate one product term by each AND gate.

All these product terms are available at the inputs of each programmable OR gate. But, only program the required product terms in order to produce the respective Boolean functions by each OR gate. The symbol 'X' is used for programmable connections.

4.9 DESIGNING OF CIRCUITS USING PLA

Basic block diagram for PLA:



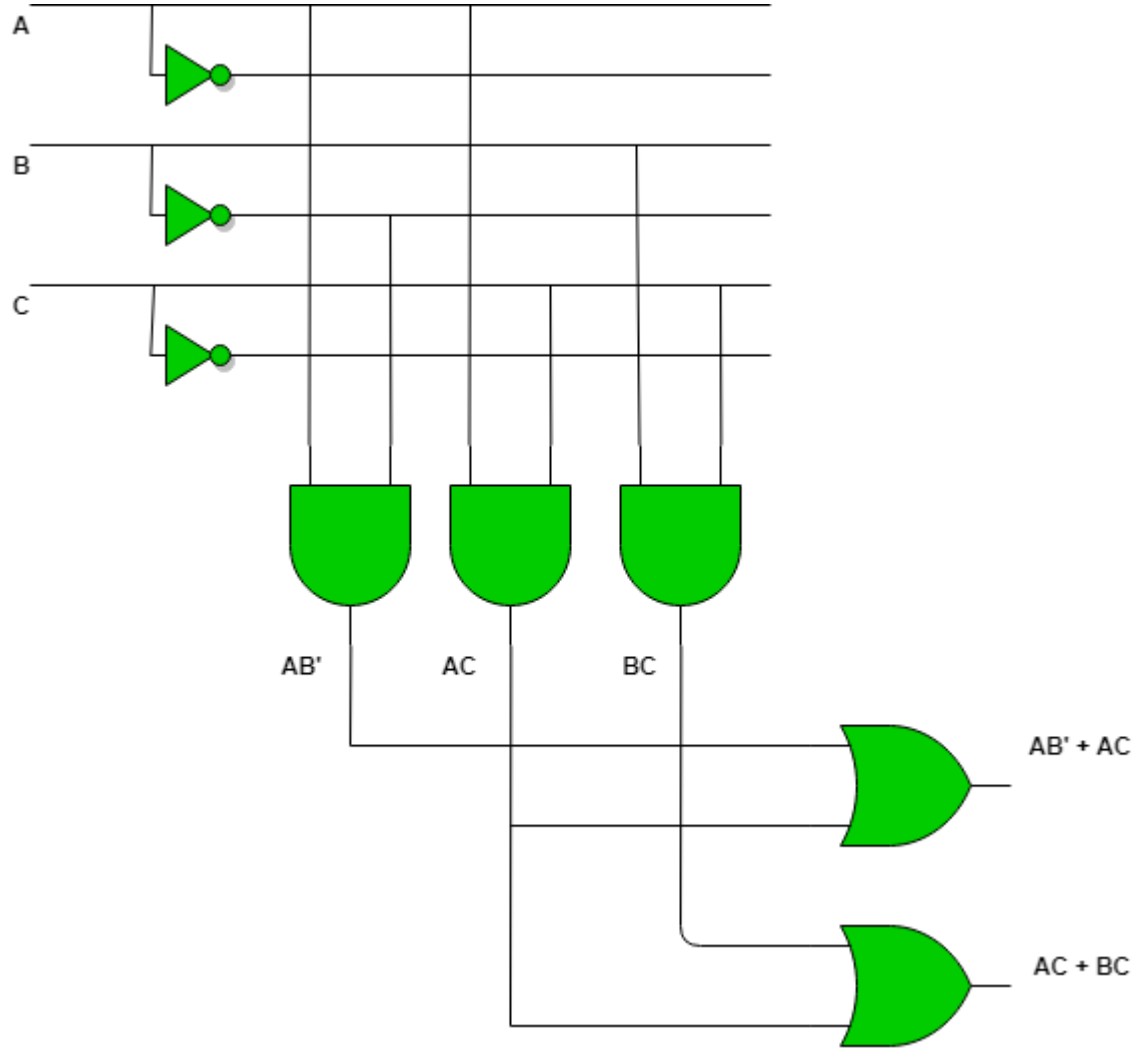
Following Truth table will be helpful in understanding function on no of inputs-

A	B	C	F1	F2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

F1 = AB'C' + AB'C + ABC' + ABC
on simplifying we get : F1 = AB + AC'

F2 = A’BC + AB’C + ABC
on simplifying we get: F2 = BC + AC

For realization of above function following circuit diagram will be used.



PLA is used for implementation of various combinational circuits using buffer, AND gate and OR gate. In PLA, all the minterms are not realized but only required minterms are implemented. As PLA has programmable AND gate array and programmable OR gate array, it provides more flexibility but disadvantage is, it is not easy to use.

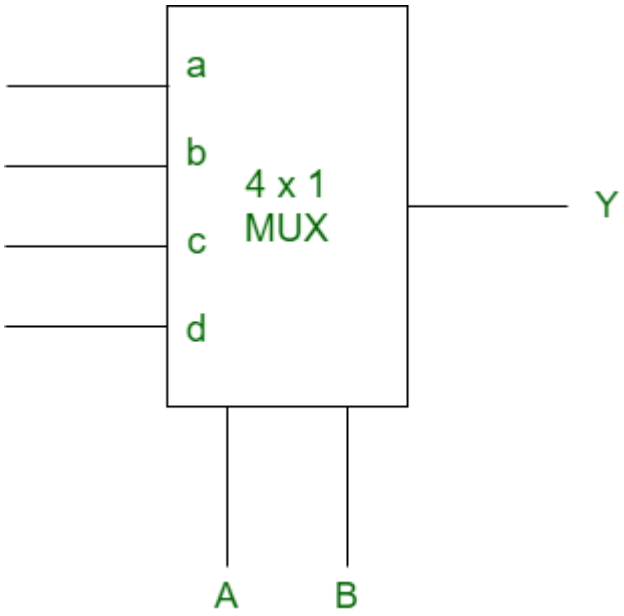
Applications:

- PLA is used to provide control over datapath.
- PLA is used as a counter.
- PLA is used as a decoders.
- PLA is used as a BUS interface in programmed I/O.

4.10 DESIGNING OF CIRCUITS USING MUX

Given a SOP function and a multiplexer is also given. We will need to implement the given SOP function using the given MUX.
There are certain steps involved in it:
Step 1: Draw the truth table for the given number of variable function.
Step 2: Consider one variable as input and remaining variables as select lines.
Step 3: Form a matrix where input lines of MUX are columns and input variable and its compliment are rows.
Step 4: Find AND between both rows on the basis of the truth table.
Step 5: Hence whatever is found is considered as input of MUX.
We will illustrate it with an example:

Example:
Given SOP function f(A, B, C) = m(0, 1, 4, 6, 7) and MUX is



For 3 variable function, the truth table is

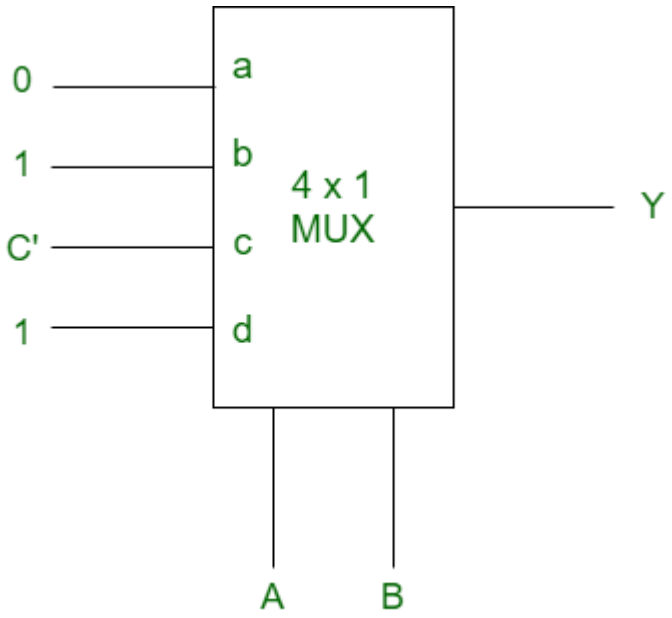
Truth Table

	A	B	C	Y
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

Let A and B are the select lines and C be the input,

	a	b	c	d
C'	0	2	4	6
C	1	3	5	7
	1	0	C'	1

Thus, for the implementation of given logical function, required is one 4×1 MUX and and inverter.



References :

1. Moris Mano, “Digital Computer Fundamentals” TMH 3rd Edition
2. http://www.tutorialspoint.com/computer_logical_organization/number_system_conversion.htm
3. http://www.electronics-tutorials.ws/binary/signed-binary-numbers.html
4. HAMMING, R. →. “Error Detecting and Error Correcting Codes.” Bell System Tech. Jour., 29 (1950): 147–160.
5. A.P GODSE,D.A.GODSE .”Digital Systems”.Technical Publications.Pune.
6. http://www.tutorialspoint.com/computer_logical_organization/binary_codes.htm
7. http://nptel.ac.in/courses/Webcourse-contents/IIScBANG/Digital%20Systems/Digital%20Systems.pdf
8. Digital Logic Circuits by D.A.Godse A.P.Godse

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

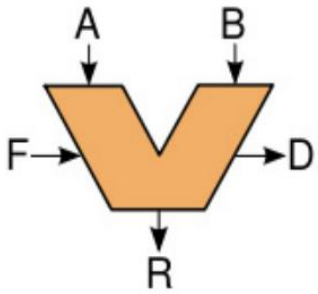
UNIT - V

Digital Computer Fundamentals-SBSA1101

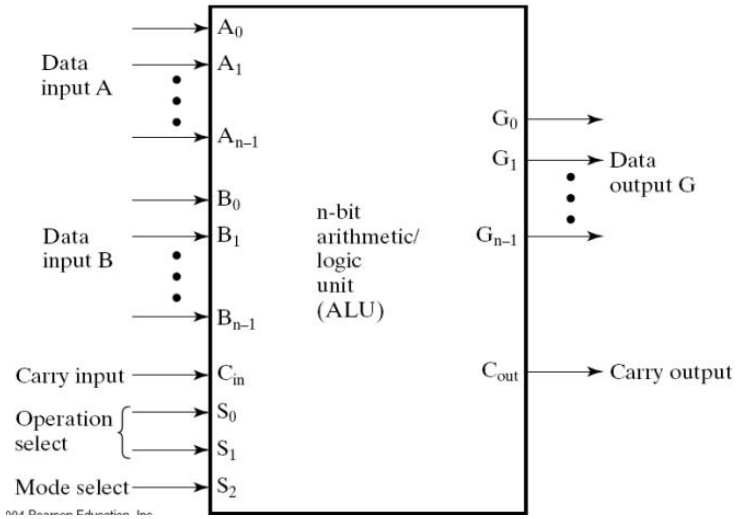
5.1 DESIGN OF ALU

Microprocessors tend to have a single module that performs arithmetic operations on integer values. This is because many of the different arithmetic and logical operations can be performed using similar (if not identical) hardware. The component that performs the arithmetic and logical operations is known as the Arithmetic Logic Unit, or ALU.

The ALU is one of the most important components in a microprocessor, and is typically the part of the processor that is designed first. Once the ALU is designed, the rest of the microprocessor is implemented to feed operands and control codes to the ALU.An arithmetic logic unit (ALU) performs arithmetic and logic operations.



A and B are the inputs to the ALU (aka operands), R is the Output or Result, F is the Code or Instruction from the Control Unit (aka as op-code), D is the Output status; it indicates cases such as: carry-in, carry-out, overflow, division-by-zero, etc.



ALU has 3 Main Parts:

1. An arithmetic circuit (add, subtract)
2. A logic circuit (bitwise operation)
3. A selector to choose between the two circuits

To Design a ALU, the arithmetic section, logic section and the selector Section hast obe designed and all the three sections has to be Combined.

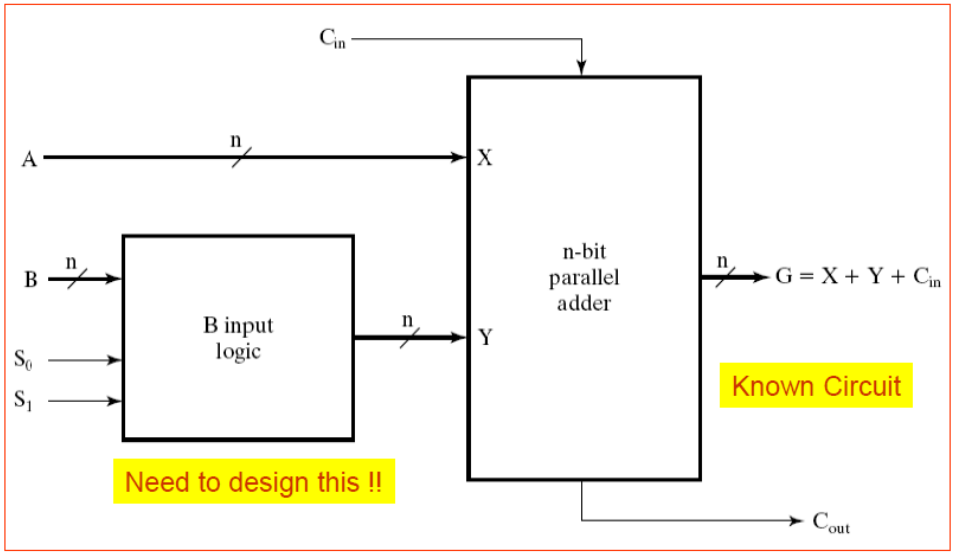
Arithmetic Section

Function Table for Arithmetic Circuit

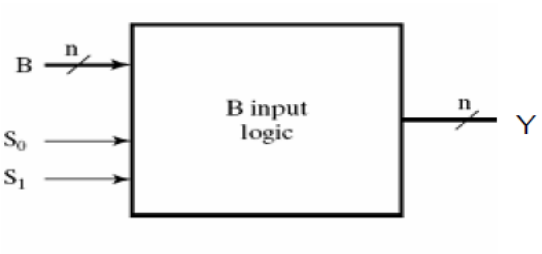
Select		Input	$G = A + Y + C_{in}$	
S_1	S_0	Y	$C_{in} = 0$	$C_{in} = 1$
0	0	all 0's	$G = A$ (transfer)	$G = A + 1$ (increment)
0	1	B	$G = A + B$ (add)	$G = A + B + 1$
1	0	$\overline{B}$	$G = A + \overline{B}$	$G = A + \overline{B} + 1$ (subtract)
1	1	all 1's	$G = A - 1$ (decrement)	$G = A$ (transfer)

Arithmetic Circuit Block Diagram

G = A + Y + Cin



Building the B input Logic



Inputs			Output	
S_1	S_0	B_i	Y_i	
0	0	0	0	$Y_i = 0$
0	0	1	0	
0	1	0	0	$Y_i = B_i$
0	1	1	1	
1	0	0	1	$Y_i = \overline{B_i}$
1	0	1	0	
1	1	0	1	$Y_i = 1$
1	1	1	1	

Select		Input
S_1	S_0	Y
0	0	all 0's
0	1	B
1	0	$\overline{B}$
1	1	all 1's

Input = S_1, S_0 and B
Output = Y

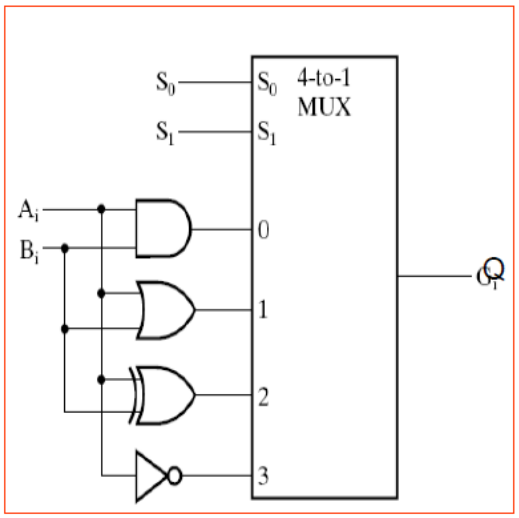
- Obtain the K-Map
- Get the Boolean Expression

$$Y = BS_0 + \overline{B}S_1$$

Logic Section

Building the Logic Circuit

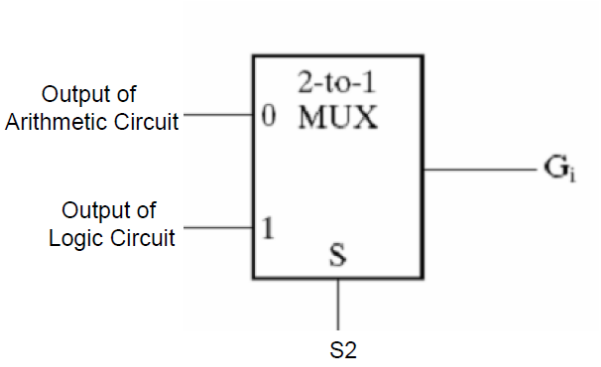
S_1	S_0	Output	Operation
0	0	$G = A \wedge B$	AND
0	1	$G = A \vee B$	OR
1	0	$G = A \oplus B$	XOR
1	1	$G = \overline{A}$	NOT



Note : if 4 bit is wanted, then we have to arrange it in array

One Stage of Logic Circuit

Selector Section



5.2 DESIGN OF STATUS REGISTER

A status register, flag register, or condition code register (CCR) is a collection of status flag bits for a processor. Examples of such registers include FLAGS register in the x86 architecture. The status register is a hardware register that contains information about the state of the processor. Individual bits are implicitly or explicitly read and/or written by the machine code instructions executing on the processor. The status register lets an instruction take action contingent on the outcome of a previous instruction.

Typically, flags in the status register are modified as effects of arithmetic and bit manipulation operations. For example, a Z bit may be set if the result of the operation is zero and cleared if it is nonzero. Other classes of instructions may also modify the flags to indicate status. For example, a string instruction may do so to indicate whether the instruction terminated because it found a match/mismatch or because it found the end of the string. The flags are read by a subsequent conditional instruction so that the specified action (depending on the processor, a jump, call, return, or so on) occurs only if the flags indicate a specified result of the earlier instruction. A status register may often have other fields as well, such as more specialized flags, interrupt enable bits, and similar types of information. During an interrupt, the status of the thread currently executing can be preserved (and later recalled) by storing the current value of the status register along with the program counter and other active registers into the machine stack or some other reserved area of memory.

Common flags

This is a list of the most common CPU status register flags, implemented in almost all modern processors.

Flag	Name	Description
Z	Zero flag	Indicates that the result of an arithmetic or logical operation (or, sometimes, a load) was zero.

C	Carry flag	Enables numbers larger than a single word to be added/subtracted by carrying a binary digit from a less significant word to the least significant bit of a more significant word as needed. It is also used to extend bit shifts and rotates in a similar manner on many processors (sometimes done via a dedicated **X** flag).
S / **N**	Sign flag Negative flag	Indicates that the result of a mathematical operation is negative. In some processors,[2] the N and S flags are distinct with different meanings and usage: One indicates whether the last result was negative whereas the other indicates whether a subtraction or addition has taken place.
V / **O** / **W**	Overflow flag	Indicates that the signed result of an operation is too large to fit in the register width using two's complement representation.

Other flags

On some processors, the status register also contains flags such as these:

Flag	Name	Description
H / **A** / **DC**	Half-carry flag Auxiliary flag Digit Carry Decimal adjust flag	Indicates that a bit carry was produced between the nibbles (typically between the 4-bit halves of a byte operand) as a result of the last arithmetic operation. Such a flag is generally useful for implementing BCD arithmetic operations on binary hardware.
P	Parity flag	Indicates whether the number of set bits of the last result is odd or even.
I	Interrupt flag	On some processors, this bit indicates whether interrupts are enabled or masked.[3] If the processor has multiple interrupt priority levels, such as the PDP-11, several bits may be used to indicate the priority of the current thread, allowing it to be interrupted only by hardware set to a higher priority. On other architectures, a bit may indicate that an interrupt is currently active, and that the current thread is part

		of an interrupt handler.
S	Supervisor flag	On processors that provide two or more protection rings, one or more bits in the status register indicate the ring of the current thread (how trusted it is, or whether it must use the operating system for requests that could hinder other threads). On a processor with only two rings, a single bit may distinguish Supervisor from User mode.

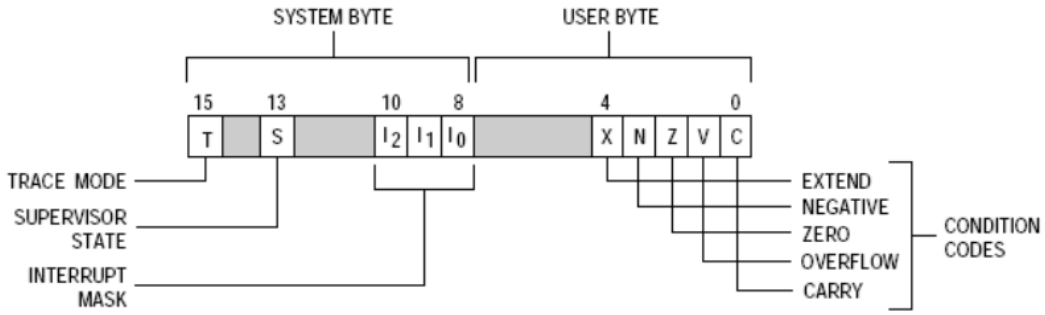
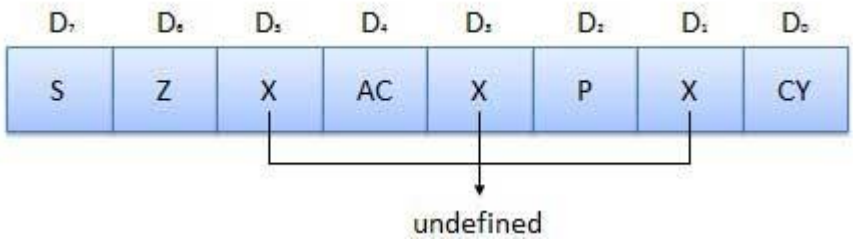


Fig 2-4 from M68000 8-/16-/32-Bit Microprocessors User's Manual [pdf, 184p; Motorola]

5.3 DESIGN OF ACCUMULATOR

It is an 8-bit register that is part of ALU. This register is used to store 8-bit data & in performing arithmetic & logic operation. The result of operation is stored in accumulator.



Register A is an 8-bit register used in 8085 to perform arithmetic, logical, I/O & LOAD/STORE operations. Register A is quite often called as an Accumulator. An accumulator is a register for short-term, intermediate storage of arithmetic and logic data in a computer's CPU (Central Processing Unit).In an arithmetic operation involving two operands, one operand has tobe in this register. And the result of the arithmetic operation will be stored

or accumulated in this register. Similarly, in a logical operation involving two operands, one operand has to be in the accumulator. Also, some other operations, like complementing and decimal adjustment, can be performed only on the accumulator.

Let us now consider a program segment which involves the content of Accumulate only. In 8085 Instruction set, **STA** is a mnemonic that stands for STore Accumulator contents in memory. In this instruction, Accumulator 8-bit content will be stored in a memory location whose 16-bit address is indicated in the instruction as a16. This instruction uses absolute addressing for specifying the destination. This instruction occupies 3-Bytes of memory. First Byte is required for the opcode, and next successive 2-Bytes provide the 16-bit address divided into 8-bits each consecutively.

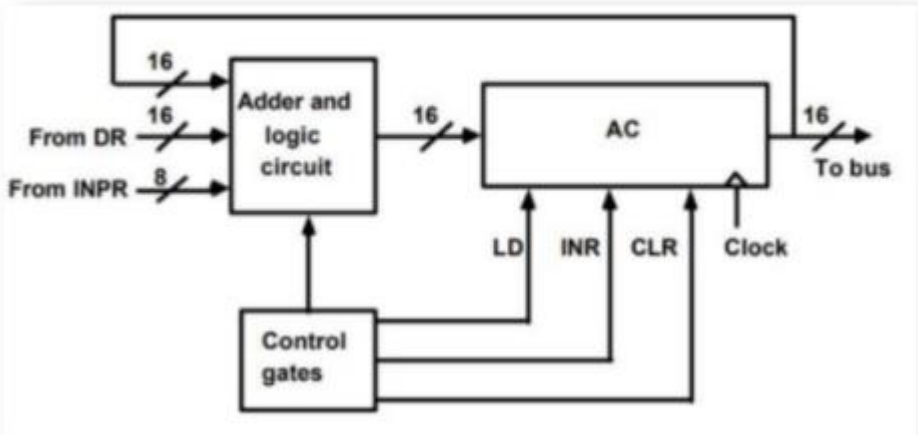
Mnemonics, Operand	**Opcode (in HEX)**	**Bytes**
STA Address	32	3

Let us consider **STA 4050** Has an example instruction of this type. It is a 3-Byte instruction. The first Byte will contain the opcode hex value 32H. As in 8085 assembly language coding supports low order Byte of the address should be mentioned at first then the high order Byte of the address should be mentioned next. So next Byte in memory will hold 50H and after that 40H will be kept in the last third Byte. Let us suppose the initial content of Accumulator is ABH and initial content of memory location 4050H is CDH. So after execution, Accumulator content will remain as ABH and 4050H location's content will become ABH replacing its previous content CDH. The content tracing of this instruction has been shown below –

	Before	**After**
(A)	ABH	ABH
(4050H)	CDH	ABH

Address	**Hex Codes**	**Mnemonic**	**Comment**
2008	2A	STA 4050H	The content of the memory location 4050H ← A
2009	50		Low-order Byte of the address
200A	40		High order Byte of the address

Accumulator Unit



5.4 INTRODUCTION TO COMPUTER DESIGN.

What a computer is used for, what tasks it must perform, and how it interacts with humans and other systems determine the functionality of the machine and, therefore, its architecture, memory, and I/O. An arbitrary desktop computer (not necessarily a PC) is shown in Figure 1-11. It has a large main memory to hold the operating system, applications, and data, and an interface to mass storage devices (disks and DVD/CD-ROMs). It has a variety of I/O devices for user input (keyboard, mouse, and audio), user output (display interface and audio), and connectivity (networking and peripherals). The fast processor requires a system manager to monitor its core temperature and supply voltages, and to generate a system reset.

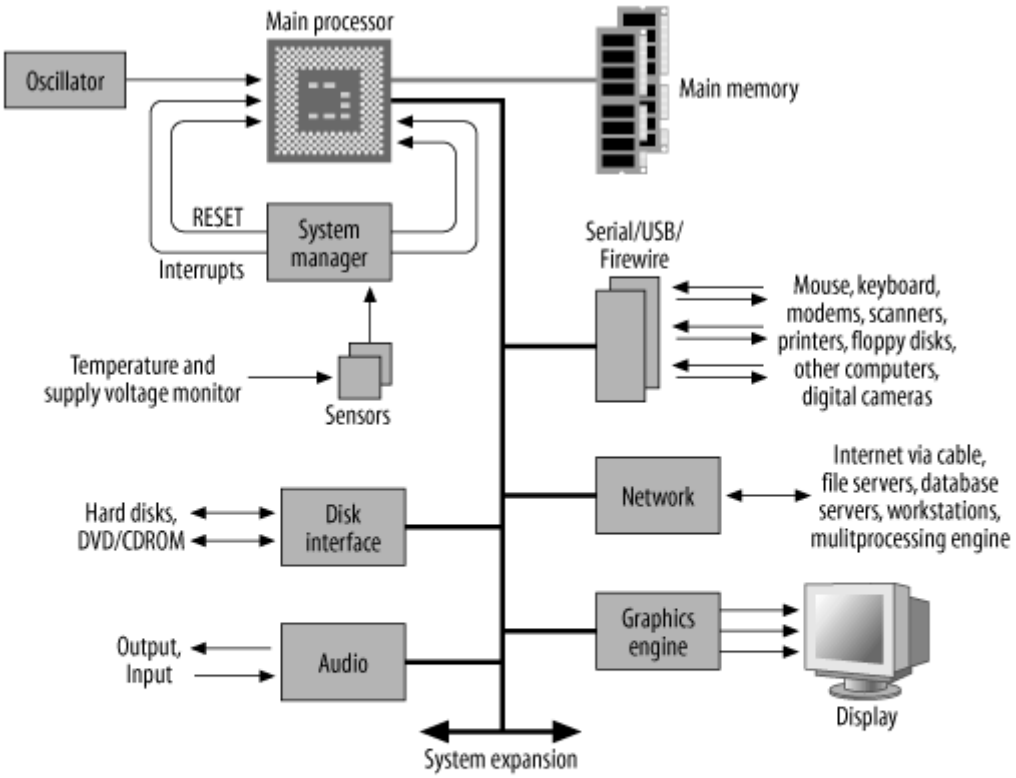


Figure 1-11. Block diagram of a generic computer

Large-scale embedded computers may also take the same form. For example, they may act as a network router or gateway, and so will require one or more network interfaces, large memory, and fast operation. They may also require some form of user interface as part of their embedded application and, in many ways, may simply be a conventional computer dedicated to a specific task. Thus, in terms of hardware, many high-performance embedded systems are not that much different from a conventional desktop machine.

Smaller embedded systems use microcontrollers as their processor, with the advantage that this processor will incorporate much of the computer's functionality on a single chip. An arbitrary embedded system, based on a generic microcontroller, is shown in Figure 1-12.

The microcontroller has, at a minimum, a CPU, a small amount of internal memory (ROM and/or RAM), and some form of I/O, which is implemented within a microcontroller as subsystem blocks. These subsystems provide the additional functionality for the processor and are common across many processors. The subsystems that you will typically find in microcontrollers will be discussed in the coming chapters.

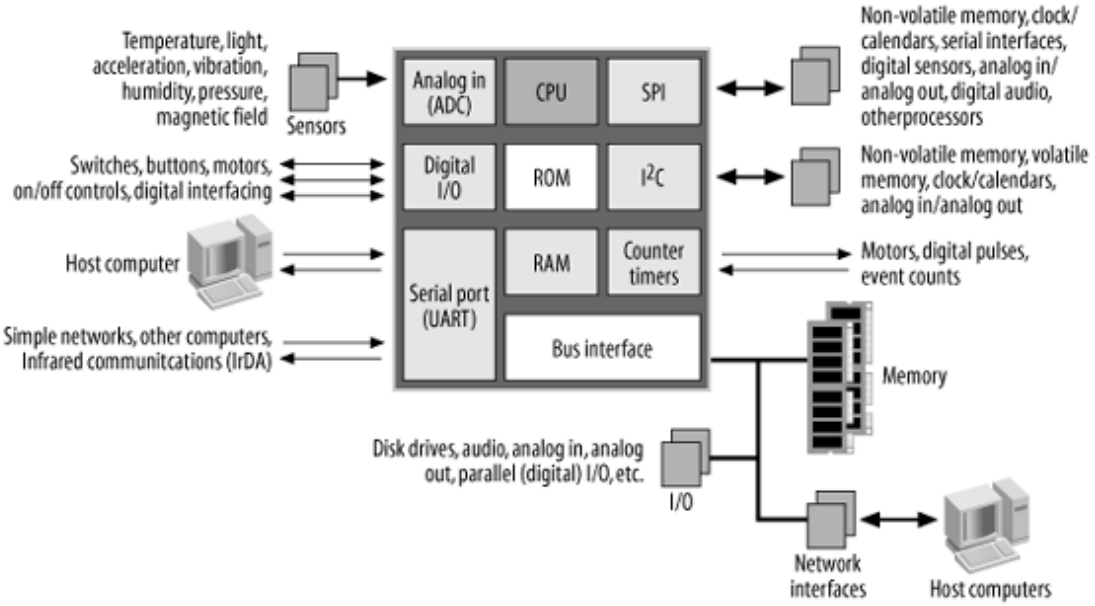


Figure 1-12. Block diagram of an embedded computer

References :

1. Moris Mano, “Digital Computer Fundamentals” TMH 3rd Edition
2. http://www.tutorialspoint.com/computer_logical_organization/number_system_conversion.htm
3. http://www.electronics-tutorials.ws/binary/signed-binary-numbers.html
4. HAMMING, R. →. “Error Detecting and Error Correcting Codes.” Bell System Tech. Jour., 29 (1950): 147–160.
5. A.P GODSE,D.A.GODSE .”Digital Systems”.Technical Publications.Pune.
6. http://www.tutorialspoint.com/computer_logical_organization/binary_codes.htm
7. http://nptel.ac.in/courses/Webcourse-contents/IIScBANG/Digital%20Systems/Digital%20Systems.pdf
8. Digital Logic Circuits by D.A.Godse A.P.Godse