CS-251

LOGIC DESIGN & SWITCHING THEORY (LDST)

CSIT SPRING 2022, BATCH: 2021

LOGIC DESIGN & SWITCHING THEORY

Introduction

Logic Design: Logic gates are the basic building blocks of digital systems and the art of interconnecting such gates to achieve prescribed outcomes is called logic design.

Switching Theory: Switching theory is the abstract mathematical formalization used in the logic design of digital networks. Switching theory studies properties of switching circuits and attempts to discover design procedures which optimize some parameters of the design i.e it is the discipline which makes use of mathematical models and technique to handle problems associated with the design of digital circuits.

Types of Number System

There are various types of number systems in mathematics. The four most common number system types are:

BASE: r
SYMBOLS: 0 – (r-1)
TOTAL # OF SYMBOLS : r

1. Decimal number system **(Base- 10) , r=10**
2. Binary number system **(Base- 2) , r=2**
3. Octal number system **(Base-8) , r=8**
4. Hexadecimal number system **(Base- 16) , r=16**

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

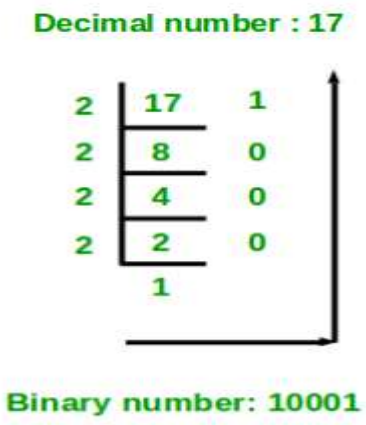
Number System Conversion Methods

Decimal to Other Bases

Converting a decimal number to other base numbers is easy. We have to divide the decimal number by the converted value of the new base.

1) Decimal to Binary

Decimal numbers can be converted to binary by repeated division of the number by 2 while recording the remainder.

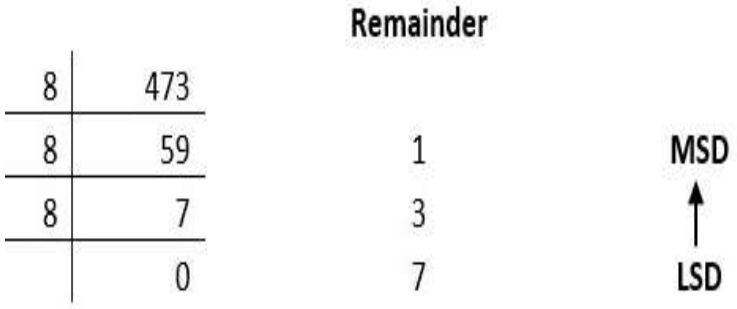


b) $43_{10} = 101011_2$

c) $27_{10} = 11011_2$

2) Decimal to Octal

Decimal numbers can be converted to octal by repeated division of the number by 8 while recording the remainder



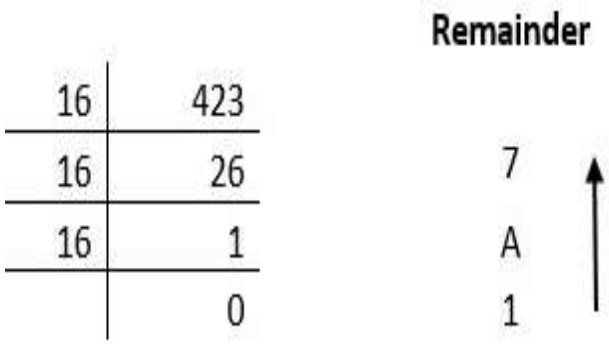
$473_{10} = 731_8$

b) 33(Dec) = 41(Oct)

c)88(Dec) = 130(Oct)

3) Decimal to Hexadecimal

Decimal numbers can be converted to octal by repeated division of the number by 16 while recording the remainder.



$423_{10} = 1A7_{16}$

b) 910(Dec) = 38E(Hex)

c)450(Dec) = 1C2(Hex)

Binary to Other Bases

1) Binary to Decimal:

In this conversion, binary number to a decimal number, we use multiplication method, in such a way that, if a number with base n has to be converted into a number with base 10, then each digit of the given number is multiplied from MSB to LSB with reducing the power of the base.

Example a). Convert $(1101)_2$ into a decimal number.
Solution: Given a binary number $(1101)_2$.

Now multiplying each digit from MSB to LSB with reducing the power of the base number 2.

$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$= 8 + 4 + 0 + 1$

$= 13$

Therefore, $(1101)_2 = (13)_{10}$

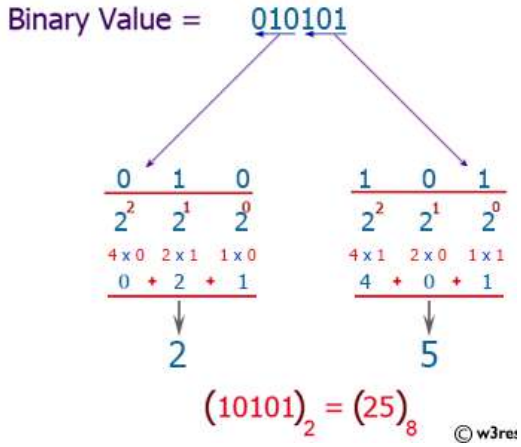
b) 1010(Bin) = 10(Dec)

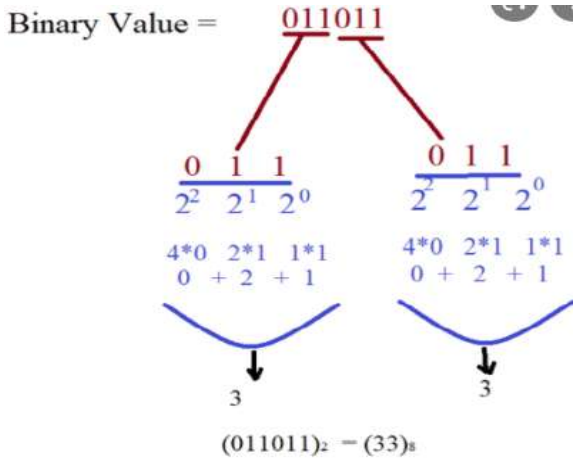
c)10101(Bin) = 21(Dec)

2) Binary to Octal

Using Grouping:

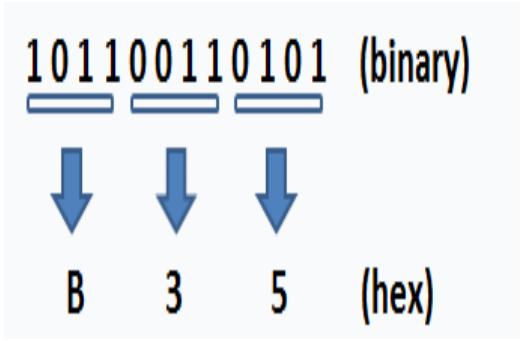
Since, there are only 8 digits (from 0 to 7) in octal number system, so we can represent any digit of octal number system using only 3 bit

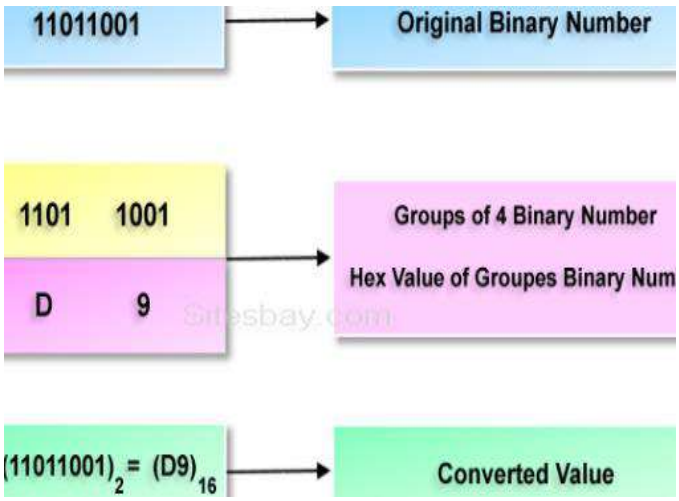




3) Binary to Hexadecimal

1. Start at the rightmost digit and break the binary number up into groups of four digits. These are known as nibbles.
2. Next, convert each group of four digits into hex equivalent.
3. Put the hex digits together.





Octal to Other Bases

1) Octal to Binary

There is a simple direct method to convert an octal number to binary number. Since there are only 8 symbols (i.e., 0, 1, 2, 3, 4, 5, 6, and 7) in octal representation system and its base (i.e., 8) So, you can represent each digit of octal in group of 3 bits in binary number.

Octal Symbol	Binary equivalent
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Example-a Convert octal number 540 into binary number.

$(540)_8$

$= (101\ 100\ 000)_2$

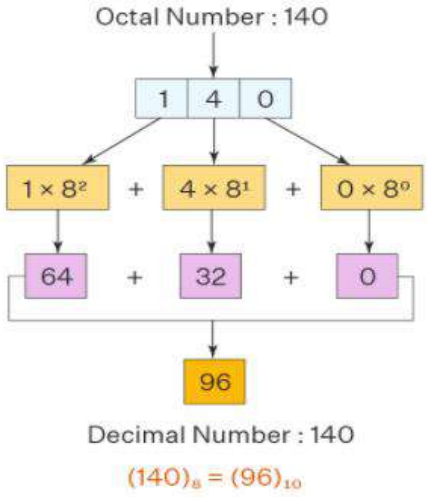
$= (101100000)_2$

b) $12_8 = (001\ 010)_2$

c) $56_8 = (101110)_2$

2) Octal to Decimal

1. Arrange the octal number with the power of 8.
2. Find the value of each of the numbers with exponents i.e. 8^0 is 1, 8^1 is 8, etc.
3. Multiply each number.
4. Add the product of all numbers to obtain the decimal number.



b) Convert Octal number 2671 to a Decimal number.

Solution:

$(2671)8 = 2 \times 8^3 + 6 \times 8^2 + 7 \times 8^1 + 1 \times 8^0$

$(2671)8 = 2 \times 512 + 6 \times 64 + 7 \times 8 + 1 \times 1$

$(2671)8 = 1024 + 384 + 56 + 1$

$(2671)8 = 1465$

Therefore, $(2671)8 = (1465)10$

c) convert octal number (121)8 to its decimal form.

Solution:

$(121)8 = 1 \times 8^2 + 2 \times 8^1 + 1 \times 8^0$

$(121)8 = 1 \times 64 + 2 \times 8 + 1 \times 1$

$(121)8 = 64 + 16 + 1$

Therefore, $(121)8 = (81)10$.

3) Octal to hexadecimal

Using the below two methods, we can convert the octal number system into the hexadecimal number system**.**

1. Convert the octal number into **binary** and then convert the binary into hexadecimal.
2. Convert the octal number into **decimal** and then convert the decimal into hexadecimal.

Octal → Binary → Hexadecimal

Let's convert $(56)_8$ into hexadecimal

Step 1: Convert $(56)_8$ into Binary

In order to convert the octal number into binary, we need to express every octal value using 3 binary bits.

Binary equivalent of **5** is **$(101)_2$**.

Binary equivalent of **6** is **$(110)_2$**.

$= (56)_8$

$= (101)(110)$

$= (101110)_2$

Step 2: Convert $(101110)_2$ into Hexadecimal

In order to convert the binary number into hexadecimal, we need to group every 4 binary bits and calculate the value[From left to right].

$(101110)_2$ in hexadecimal

$= (101110)_2$

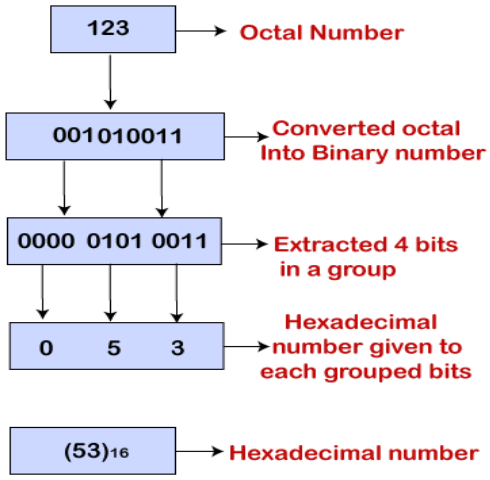
$= (10)(1110)$

$= (2)(14)$

$= (2e)_{16}$

14 equivalent hexadecimal is **e**.

This method is relatively easy compared to the below method.



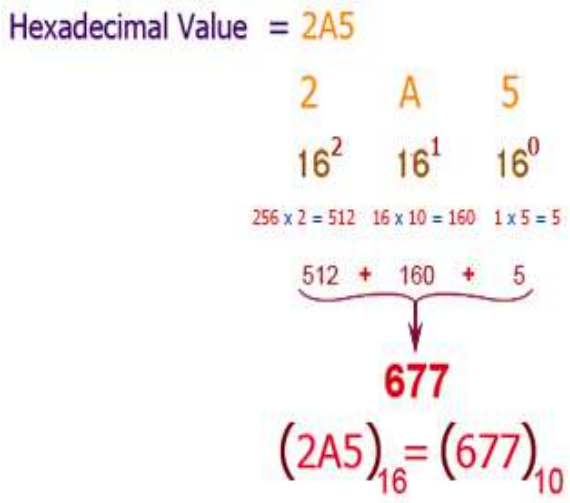
Hexadecimal to Other Bases

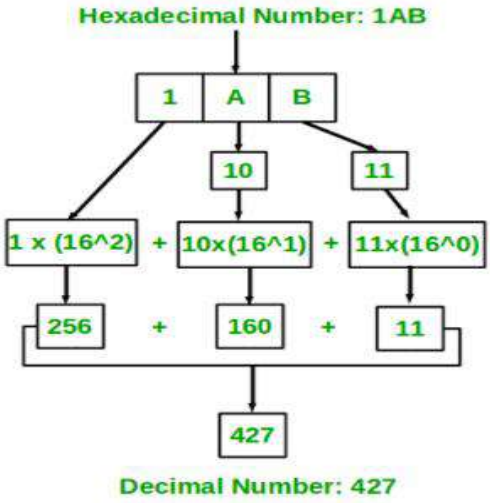
1) Hexadecimal to Decimal

Here are the steps to convert hex to decimal:

- Get the decimal equivalent of hex from table.
- Multiply every digit with 16 power of digit location.
- Sum all the multipliers.

7DE is a hex number
7DE = $(7 * 16^2) + (13 * 16^1) + (14 * 16^0)$
7DE = (7 * 256) + (13 * 16) + (14 * 1)
7DE = 1792 + 208 + 14
7DE = 2014 (in decimal number)



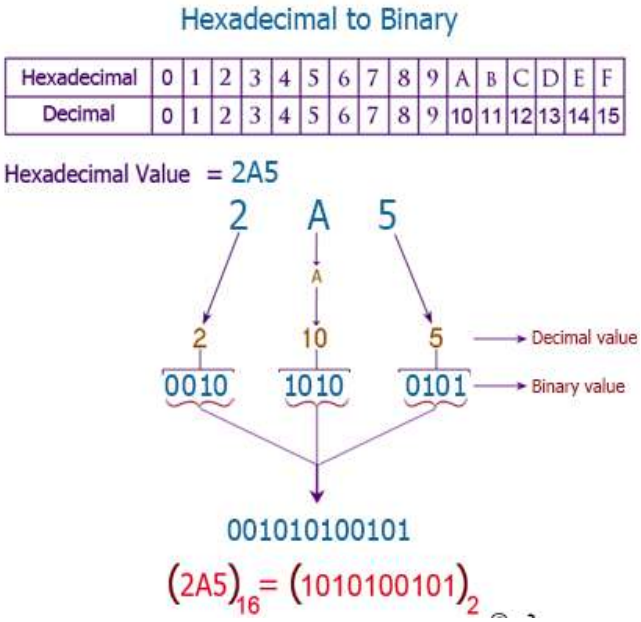


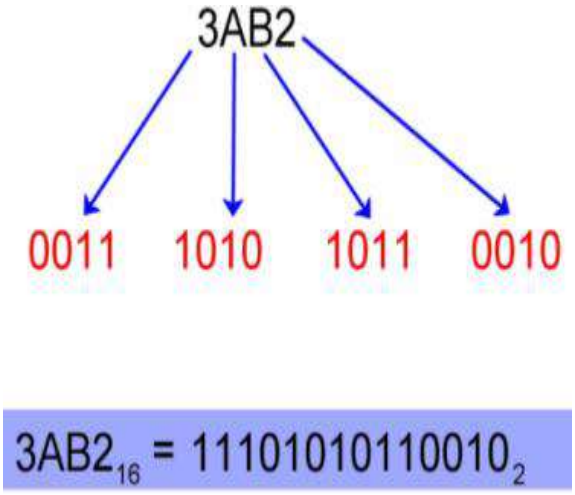
2) Hexadecimal to Binary

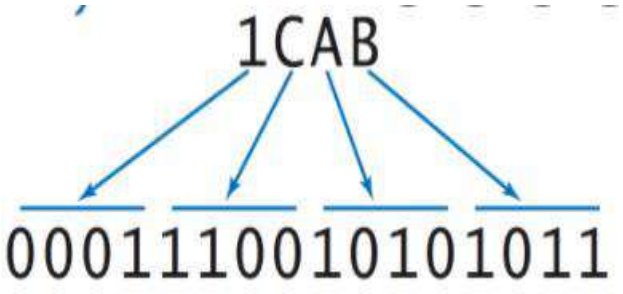
Step 1: Write down the hex number. If there are any, change the hex values represented by letters to their decimal equivalents.

Step 2: Each hex digit represents four binary digits and therefore is equal to a power of 2.

Step 3: Read the 1's and 0's from left to right to get the binary equivalent of the given hex number.







3) Hexadecimal to Octal

There is another method to convert any hexadecimal to its equivalent octal. As we know, hexadecimal numbers include binary digits; therefore, we can club these binary numbers into a pair so that we can relate it with the octal numbers. Let us check the method with steps and example:

i. For each given hexadecimal number digit, write the equivalent binary number. If any of the binary equivalents are less than 4 digits, add 0's to the left side.
ii. Combine and make the groups of binary digits from right to left, each containing 3 digits. Add 0's to the left if there are less than 3 digits in the last group.
iii. Find the octal equivalent of each binary group.

Example: Convert $1BC_{16}$ into an octal number.
Solution: Given, $1BC_{16}$ is a hexadecimal number.

$1 \rightarrow 0001$, $B \rightarrow 1011$, $C \rightarrow 1100$

Now group them from right to left, each having 3 digits.

000, 110, 111, 100

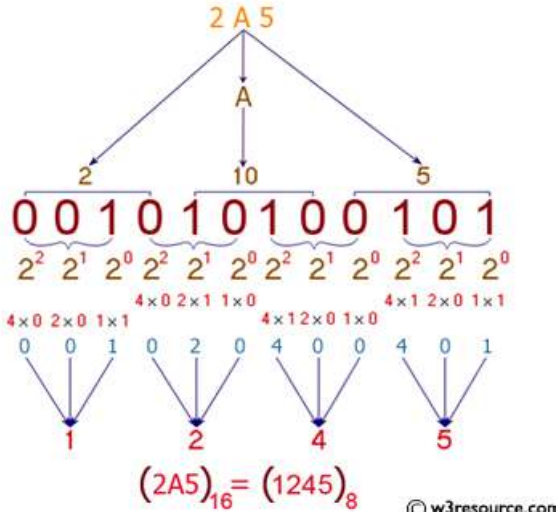
$000 \rightarrow 0$, $110 \rightarrow 6$, $111 \rightarrow 7$, $100 \rightarrow 4$

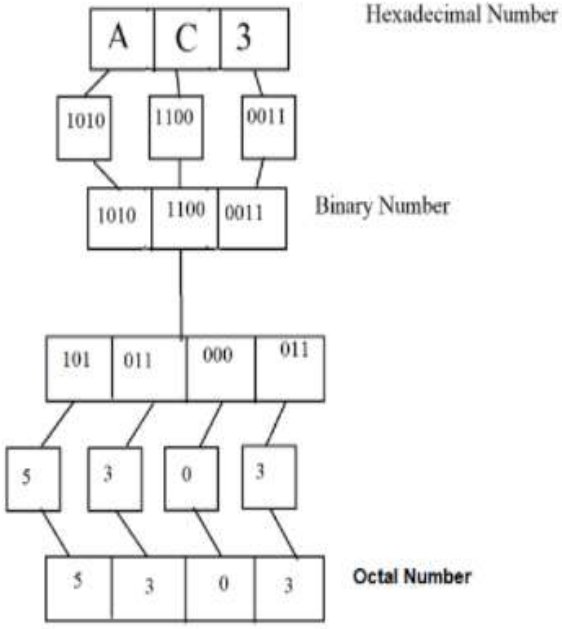
Hence, $1BC_{16} = 674_8$

Hexadecimal to Octal

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

Hexadecimal Value = 2A5





CS-251

LOGIC DESIGN & SWITCHING THEORY (LDST)

CSIT SPRING 2024, BATCH: 2023

Logic Gates

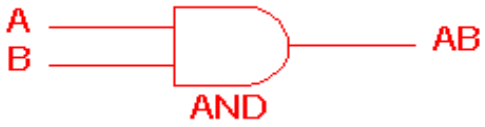
Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on certain logic. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc.

There are **3** basic logic gates:

1) AND GATE:

The AND gate is an electronic circuit that gives a **high** output (**1**) only if all its inputs are **high**. A dot **(.)** is used to show the AND operation i.e. **A.B**.
Bear in mind that this dot is sometimes omitted i.e. **AB**

LOGIC SYMBOL



TRUTH TABLE

2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

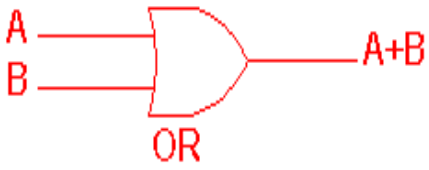
LOGIC EXPRESSION

Output=A.B

2) OR GATE:

The OR gate is an electronic circuit that gives a **high** output (**1**) if one or more of its inputs are **high**. A plus **(+)** is used to show the OR operation.
i.e. **A+B**

LOGIC SYMBOL



TRUTH TABLE

2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

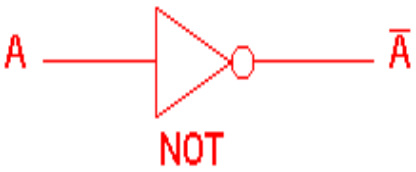
LOGIC EXPRESSION:

Output=A+B

3) NOT GATE

The NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an ***inverter***. If the input variable is **A** the inverted output is known as **NOT A**. This is also shown as **A'**, or A with a bar over the top.

LOGIC SYMBOL



TRUTH TABLE

NOT gate	
A	$\bar{A}$
0	1
1	0

LOGIC EXPRRSSION:

A=NOT A or A'

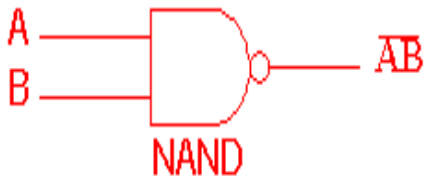
Derived Gates

1) NAND GATE

This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate. The outputs of all NAND gates are **high** if **any** of the inputs are **low**.

The symbol is an AND gate with a small circle on the output. The small circle represents inversion.

LOGIC SYMBOL



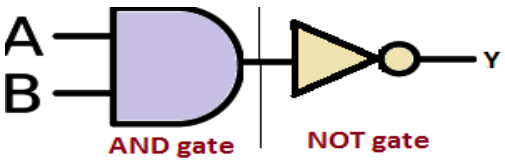
TRUTH TABLE

2 Input NAND gate		
A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

LOGIC EXPRESSION:

$$\text{Output}=\overline{AB}$$

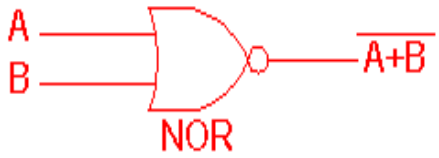
EQUIVALENT CIRCUIT:



2) NOR GATE

This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate. The outputs of all NOR gates are **low** if **any** of the inputs are **high**.
The symbol is an OR gate with a small circle on the output. The small circle represents inversion.

LOGIC SYMBOL **TRUTH TABLE**

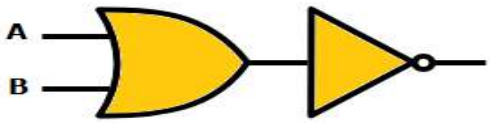


2 Input NOR gate		
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

LOGIC EXPRESSION:

Output: $A+\overline{B}$

EQUIVALENT CIRCUIT:

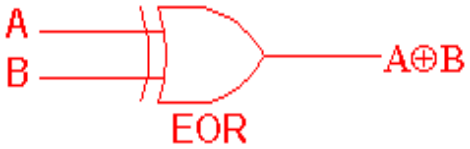


$X = \overline{A + B}$

1) 'Exclusive-OR' Gate (XOR Gate):

The 'Exclusive-OR' gate is a circuit which will give a **high** output if either, but not both, of its two inputs are **high**. An encircled plus sign **(⊕)** is used to show the EOR operation.

LOGIC SYMBOL



TRUTH TABLE

2 Input EXOR gate		
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

LOGIC EXPRESSION:

Output=A⊕B

$$A \oplus B = A \cdot \overline{B} + \overline{A} \cdot B$$

In general

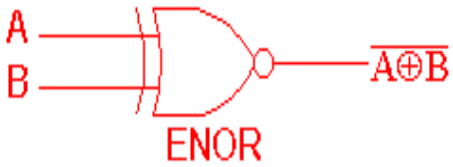
XOR gate is an **"odd"** function.

Output is **"1"** if the input variables have an **odd** number of **1's**.

2) 'Exclusive-NOR' Gate (XNOR):

The 'Exclusive-NOR' gate circuit does the opposite to the EOR gate. It will give a **low** output if either, but not both, of its two inputs are **high**. The symbol is an EXOR gate with a small circle on the output. The small circle represents inversion.

LOGIC SYMBOL



TRUTH TABLE

2 Input EXNOR gate		
A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

LOGIC EXPRESSION:

$$\text{Output} = A \overline{\oplus} B$$

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

In general

XNOR gate is an **"even"** function.

Output is **"1"** if the input variables have **even** number of **0's**.

*For "odd" # of input variables:

XOR=XNOR

inputs			output
A	B	C	$Y = A \oplus B \oplus C$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

For "even" # of input variables:

XOR=$\overline{\text{XNOR}}$

XNOR=$\overline{\text{XOR}}$

CS-251

LOGIC DESIGN & SWITCHING THEORY (LDST)

CSIT SPRING 2022, BATCH: 2021

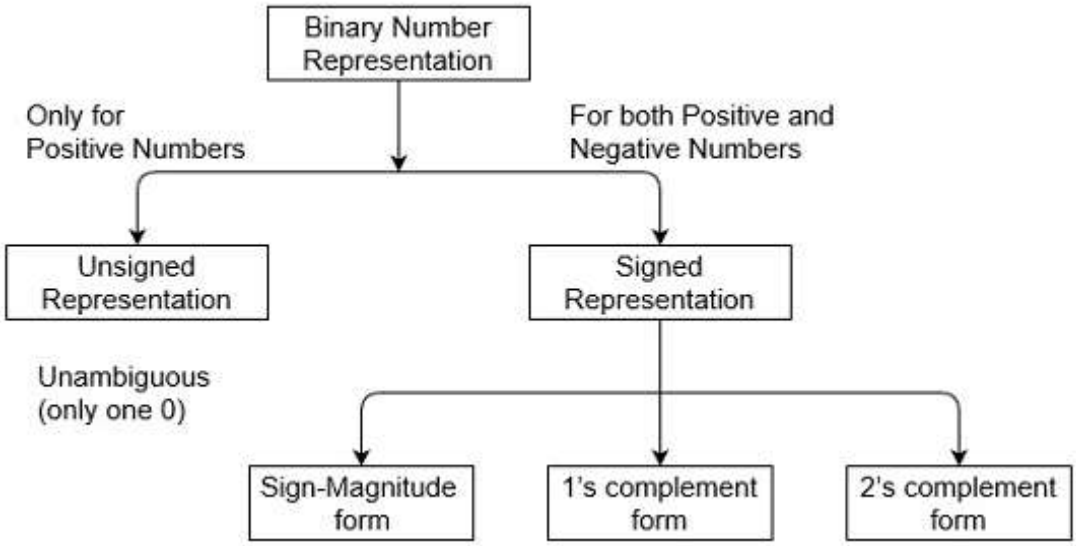
Unsigned and Signed Binary Numbers

Variables such as integers can be represent in two ways, i.e., signed and unsigned. Signed numbers use sign flag or can be distinguish between negative values and positive values. Whereas unsigned numbers stored only positive numbers but not negative numbers.

Number representation techniques like: Binary, Octal, Decimal and Hexadecimal number representation techniques can represent numbers in both signed and unsigned ways.

Representation of Binary Numbers:

Binary numbers can be represented in signed and unsigned way. Unsigned binary numbers do not have sign bit, whereas signed binary numbers uses signed bit as well or these can be distinguishable between positive and negative numbers.



1. Unsigned Numbers:

Unsigned numbers don't have any sign, these can contain only magnitude of the number. So, representation of unsigned binary numbers are all positive numbers only. For example, representation of positive decimal numbers are positive by default. We always assume that there is a positive sign symbol in front of every number.

Representation of Unsigned Binary Numbers:

Since there is no sign bit in this unsigned binary number, so N bit binary number represent its magnitude only. Zero (0) is also unsigned number. This representation has only one zero (0), which is always positive. Every number in unsigned number representation has only one unique binary equivalent form, so this is unambiguous representation technique.

The range of unsigned binary number is from **0 to (2^n-1)**.

Example-1: Represent decimal number 92 in unsigned binary number.

Simply convert it into Binary number, it contains only magnitude of the given number.
$= (92)_{10}$

$= (1011100)_2$

It's 7 bit binary magnitude of the decimal number 92.

Example-2: Find range of 5 bit unsigned binary numbers. Also, find minimum and maximum value in this range.

Since, range of unsigned binary number is from **0 to (2^n-1)**.

Therefore, range of 5 bit unsigned binary number is *from* 0 to (2^5-1) which is equal from minimum value **0** (i.e., 00000) to maximum value **31** (i.e., 11111).

2. Signed Numbers:

Signed numbers contain sign flag, this representation distinguish positive and negative numbers. This technique contains both sign bit and magnitude of a number. For example, in representation of negative decimal numbers, we need to put negative symbol in front of given decimal number.

Representation of Signed Binary Numbers:

There are three types of representations for signed binary numbers. Because of extra signed bit, binary number zero has two representation, either positive (0) or negative (1), so ambiguous representation.

These are: Sign-Magnitude form, 1's complement form, and 2's complement form which are explained as following below.

2.(a) Sign-Magnitude form:

For n bit binary number, 1 bit is reserved for sign symbol. If the value of sign bit is 0, then the given number will be positive, else if the value of sign bit is 1, then the given number will be negative. Remaining (n-1) bits represent magnitude of the number. Since magnitude of number zero (0) is always 0, so there can be two representation of number zero (0), positive (+0) and negative (-0), which depends on value of sign bit. Hence these representations are ambiguous generally because of two representation of number zero (0). Generally sign bit is a most significant bit (MSB) of representation.

The range of Sign-Magnitude form is from $(2^{(n-1)}-1)$ to $(2^{(n-1)}-1)$.

For Example:

Range of 6 bit Sign-Magnitude form binary number is from (2^5-1) to (2^5-1) which is equal from minimum value -31 (i.e., 1 11111) to maximum value +31 (i.e., 0 11111). And zero (0) has two representation, -0 (i.e., 1 00000) and +0 (i.e., 0 00000).

2.(b) 1's complement form:

Since, 1's complement of a number is obtained by inverting each bit of given number. So, we represent positive numbers in binary form and negative numbers in 1's complement form. There is extra bit for sign representation. If value of sign bit is 0, then number is positive and you can directly represent it in simple binary form, but if value of sign bit 1, then number is negative and you have to take 1's complement of given binary number. You can get negative number by 1's complement of a positive number and positive number by using 1's complement of a negative number. Therefore, in this representation, zero (0) can have two representation, that's why 1's complement form is also ambiguous form.

The range of 1's complement form is *from* $(2^{(n-1)}-1)$ to $(2^{(n-1)}-1)$.

For Example:

Range of 6 bit 1's complement form binary number is from (2^5-1) to (2^5-1) which is equal from minimum value -31 (i.e., 1 00000) to maximum value +31 (i.e., 0 11111). And zero (0) has two representation, -0 (i.e., 1 11111) and +0 (i.e., 0 00000).

2.(c) 2's complement form:

Since, 2's complement of a number is obtained by inverting each bit of given number plus 1 to least significant bit (LSB). So, we represent positive numbers in binary form and negative numbers in 2's complement form. There is extra bit for sign representation. If value of sign bit is 0, then number is positive and you can directly represent it in simple binary form, but if value of sign bit 1, then number is negative and you have to take 2's complement of given binary number. You can get negative number by 2's complement of a positive number and positive number by directly using simple binary representation. If value of most significant bit (MSB) is 1, then take 2's complement from, else not. Therefore, in this representation, zero (0) has only one (unique) representation which is always positive.

The range of 2's complement form is *from* **$(2^{(n-1)})$** to **$(2^{(n-1)}-1)$.**

For Example:

Range of 6 bit 2's complement form binary number is from (2^5) to (2^5-1) which is equal from minimum value -32 (i.e., 1 00000) to maximum value +31 (i.e., 0 11111). And zero (0) has two representation, -0 (i.e., 1 11111) and +0 (i.e., 0 00000).

Addition and Subtraction using 1's complement

Arithmetic operations such as addition and subtraction using 1's complement. We can perform addition and subtraction using 1's, 2's, 9's, and 10's complement.

Addition using 1's complement

There are three different cases possible when we add two binary numbers which are as follows:

Case 1: Addition of the positive number with a negative number when the positive number has a greater magnitude.

Initially, calculate the 1's complement of the given negative number. Sum up with the given positive number. If we get the end-around carry 1, it gets added to the LSB.

Example: 1101 and -1001

1. First, find the 1's complement of the negative number 1001. So, for finding 1's complement, change all 0 to 1 and all 1 to 0. The 1's complement of the number 1001 is 0110.
2. Now, add both the numbers, i.e., 1101 and 0110;
   1101+0110=1 0011
3. By adding both numbers, we get the end-around carry 1. We add this end around carry to the LSB of 0011.
   0011+1=0100

   Example: Add 1110 and -1101.

   1. So, take 1's complement of 1101, which will be 0010, then add with given number.
   2. 1110+0010=1 0000 , then add this carry bit to the LSB,
   3. 0000+1=0001 , which is the answer.

Case 2: Adding a positive value with a negative value in case the negative number has a higher magnitude.

Initially, calculate the 1's complement of the negative value. Sum it with a positive number. In this case, we did not get the end-around carry. So, take the 1's complement of the result to get the final result.

Example: Add 1010 and -1100 in five-bit registers.

Note that there are five-bit registers, so these new numbers will be 01010 and -01100.

1. Now take 1's complement of 01100 which will be 10011

2. Add 01010+10011=11101 .

3. Then take 1's complement of this result, which will be 00010 and this will be negative number, i.e., -00010, which is the answer.

Example: 1101 and -1110

1. First find the 1's complement of the negative number 1110. So, for finding 1's complement, we change all 0 to 1, and all 1 to 0. 1's complement of the number 1110 is 0001.
2. Now, add both the numbers, i.e., 1101 and 0001;
   1101+0001= 1110
3. Now, find the 1's complement of the result 1110 that is the final result. So, the 1's complement of the result 1110 is 0001, and we add a negative sign before the number so that we can identify that it is a negative number.

Case 3: Addition of two negative numbers

In this case, first find the 1's complement of both the negative numbers, and then we add both these complement numbers. In this case, we always get the end-around carry, which get added to the LSB, and for getting the final result, we take the 1's complement of the result.

Example: -1101 and -1110 in five-bit register

1. Firstly find the 1's complement of the negative numbers 01101 and 01110. So, for finding 1's complement, we change all 0 to 1, and all 1 to 0. 1's complement of the number 01110 is 10001, and 01101 is 10010.
2. Now, we add both the complement numbers, i.e., 10001 and 10010;
   10001+10010= 1 00011
3. By adding both numbers, we get the end-around carry 1. We add this end-around carry to the LSB of 00011.
   00011+1=00100
4. Now, find the 1's complement of the result 00100 that is the final answer. So, the 1's complement of the result 00100 is 110111, and add a negative sign before the number so that we can identify that it is a negative number.

   Example: Add -1010 and -0101 in five bit-register.

   These five bit numbers are -01010 and -00101.

   1. Add complements of these numbers, 10101+11010 =1 01111 .

2. Since, there is a carry bit 1, so add this to the LSB of result, i.e., 01111+1=10000 .
3. Now take the 1's complement of this result, which will be 01111 and this number is negative, i.e, -01111, which is answer.

Note that end-around-carry-bit addition occurs only in 1's complement arithmetic operations but not in 2's complement arithmetic operations

Subtractions by 1's Complement:

The algorithm to subtract two binary number using 1's complement is explained as following below:

- Take 1's complement of the subtrahend
- Add with minuend
- If the result of above addition has carry bit 1, then add it to the least significant bit (LSB) of given result
- If there is no carry bit 1, then take 1's complement of the result which will be negative

Note that subtrahend is number that to be subtracted from the another number, i.e., minuend.

Example (Case-1: When Carry bit 1):

Evaluate 10101 - 00101

1. According to above algorithm, take 1's complement of subtrahend 00101, which will be 11010
2. Then add both of these. So, 10101 + 11010 =1 01111 .
3. Since, there is carry bit 1, so add this to the LSB of given result, i.e., 01111+1=10000 which is the answer.

Example 1: 10101 - 00111

1. We take 1's complement of subtrahend 00111, which comes out 11000. Now, sum them. So,

2. 10101+11000 =1 01101.
3. In the above result, we get the carry bit 1, so add this to the LSB of a given result, i.e., 01101+1=01110, which is the answer.

Example (Case-2: When no Carry bit):

Example: 10101 – 10111

1. We take 1's complement of subtrahend 10111, which comes out 01000. Now, add both of the numbers. So,
2. 10101+01000 =11101.
3. In the above result, we didn't get the carry bit. So calculate the 1's complement of the result, i.e., 00010, which is the negative number and the final answer.

Addition and Subtraction using 2's complement

Addition using 2's complement

Case 1: Addition of the positive number with a negative number when the positive number has a greater magnitude.

Initially find the 2's complement of the given negative number. Sum up with the given positive number. If we get the end-around carry 1 then the number will be a positive number and the carry bit will be discarded and remaining bits are the final result

Example: 1101 and -1001

1. First, find the 2's complement of the negative number 1001. So, for finding 2's complement, change all 0 to 1 and all 1 to 0 or find the 1's complement of the number 1001. The 1's complement of the number 1001 is 0110, and add 1 to the LSB of the result 0110. So the 2's complement of number 1001 is 0110+1=0111
2. Add both the numbers, i.e., 1101 and 0111;
   1101+0111=1 0100

3. By adding both numbers, we get the end-around carry 1. We discard the end-around carry. So, the addition of both numbers is 0100.

Case 2: Adding of the positive value with a negative value when the negative number has a higher magnitude.

Initially, add a positive value with the 2's complement value of the negative number. Here, no end-around carry is found. So, we take the 2's complement of the result to get the final result.

Example: 1101 and -1110

1. First, find the 2's complement of the negative number 1110. So, for finding 2's complement, add 1 to the LSB of its 1's complement value 0001.
   0001+1=0010
2. Add both the numbers, i.e., 1101 and 0010;
   1101+0010= 1111
3. So, the 2's complement of the result 1111 is 0001, and add a negative sign before the number so that we can identify that it is a negative number.

Case 3: Addition of two negative numbers

In this case, first, find the 2's complement of both the negative numbers, and then we will add both these complement numbers. In this case, we will always get the end-around carry, which will be added to the LSB, and forgetting the final result, we will take the2's complement of the result.

Example: -1101 and -1110 in five-bit register

1. Firstly find the 2's complement of the negative numbers 01101 and 01110. So, for finding 2's complement, we add 1 to the LSB of the 1's complement of these numbers. 2's complement of the number 01110 is 10010, and 01101 is 10011.
2. We add both the complement numbers, i.e., 10001 and 10010;
   10010+10011= 1 00101
3. By adding both numbers, we get the end-around carry 1. This carry is discarded and the final result is the 2.s complement of the result 00101. So,

the 2's complement of the result 00101 is 11011, and we add a negative sign before the number so that we can identify that it is a negative number.

Subtraction using 2's complement

These are the following steps to subtract two binary numbers using 2's complement

- In the first step, find the 2's complement of the subtrahend.
- Add the complement number with the minuend.
- If we get the carry by adding both the numbers, then we discard this carry and the result is positive else take 2's complement of the result which will be negative.

Example 1: 10101 - 00111

1. We take 2's complement of subtrahend 00111, which is 11001. Now, sum them. So,
2. 10101+11001 =1 01110.
3. In the above result, we get the carry bit 1. So we discard this carry bit and remaining is the final result and a positive number.

Example 2: 10101 - 10111

1. We take 2's complement of subtrahend 10111, which comes out 01001. Now, we add both of the numbers. So,
2. 10101+01001 =11110.
3. In the above result, we didn't get the carry bit. So calculate the 2's complement of the result, i.e., 00010. It is the negative number and the final answer.

Date:

"k-Map"

K-Map can produce Sum of product (SOP) or Product of Sum (POS) expression considering which of the two (0, 1) outputs are being grouped in it.

The grouping of 0's result in POS expression and the grouping of 1's result in SOP expression.

Rules

- Rule -01:-

→ We can either group 0's with 0's or 1's with 1's but we cannot group 0's and 1's together.

- Rule -02:-

→ Groups may overlap each other

- Rule -03:-

→ We can only create a group whose num of cells can be represented in the power of 2, i.e 1, 2, 4, 8, 16 and so on.

Date:_____

- **Rule -04 :-**

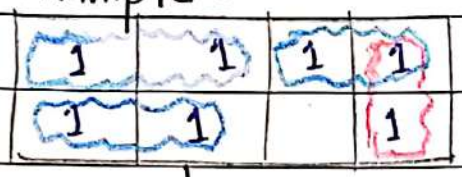
→ Groups can be only either horizontal or vertical

→ We can not create groups of diagonal or any other shape.

- **Rule -05 :-**

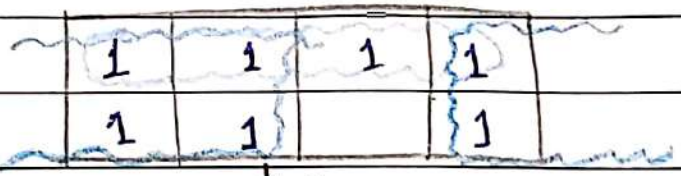
→ Each group should be as large as possible.

Example :—



Incorrect

X



Correct

✓

- **Rule -06 :-**

→ Opposite grouping and corner grouping are allowed.

CONSTRUCTION OF MAP

"2 Variable K-Map"

Cells = $2^2 = 4$

A \ B	0	1
0	$\bar{A}\bar{B}^0$	$\bar{A}B^1$
1	$A\bar{B}^2$	AB^3

$A = 1$, $B = 1$

$\bar{A} = 0$, $\bar{B} = 0$

Date:

"3 Variable k-Map"

Cells = $2^3 = 8$

AB \ C	0	1
00	0	1
01	2	3
11	6	7
10	4	5

Key = $A = 1 = B = C$

$\bar{A} = 0 = \bar{B} = \bar{C}$

"OR"

A \ BC	00	01	11	10
0	$\bar{A}\bar{B}\bar{C}^0$	$\bar{A}\bar{B}C^1$	$\bar{A}BC^3$	$\bar{A}B\bar{C}^2$
1	$A\bar{B}\bar{C}^4$	$A\bar{B}C^5$	ABC^7	$AB\bar{C}^6$

"4 Variable k-Map"

AB \ CD	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

Date:

* Convenient way of express Fonchons :-

- SOP

$F(\text{input variables}) = \sum(\text{list of minterms})$

E.g $F(A, B, C) = \sum(1, 2, 4, 7)$

- POS

$F(\text{input variables}) = \pi(\text{list of maxterms})$

E.g $F(A, B, C) = \pi(0, 3, 5, 6)$

EXAMPLES

Minterms = m

Q1 //

A	B	F	
0	0	1	$\rightarrow m_0$
0	1	1	$\rightarrow m_1$
1	0	1	$\rightarrow m_2$
1	1	0	$\rightarrow m_3$

$F = \sum(m_0, m_1, m_2)$

Solve using k - Map

Date:

Ans

F	B=0	B=1
A=0	1 (0)	1 (1)
A=1	1 (2)	0 (3)

$F = \bar{A} + \bar{B}$ Ans.

Q2 $F(A,B,C) = \sum(m_0, m_1, m_2, m_4, m_5, m_6)$

AB \ C	0	1
00	1 (0)	1 (1)
01	1 (2)	3
11	1 (6)	7
10	1 (4)	1 (5)

$F = \bar{C} + \bar{B}$

Q3 $F_F(A,B,C,D) = \sum(0,2,5,7,8,10,13,15)$

AB \ CD	00	01	11	10
00	1 (0)	0 (1)	0 (3)	1 (2)
01	0 (4)	1 (5)	1 (7)	0 (6)
11	0 (12)	1 (13)	1 (15)	0 (14)
10	1 (8)	0 (9)	0 (11)	1 (10)

$F = BD + \bar{B}\bar{D}$

* For 3 variable you can follow any structure.

Date:

Q// $F(A,B,C) = \pi(0,3,6,7)$ → Maxterms = π = 0

A \ BC	00	01	11	10
0	0^0	1^1	0^3	1^2
1	1^4	1^5	0^7	0^6

Total 3 groups

$F = \bar{A}\bar{B}\bar{C} + BC + AB$

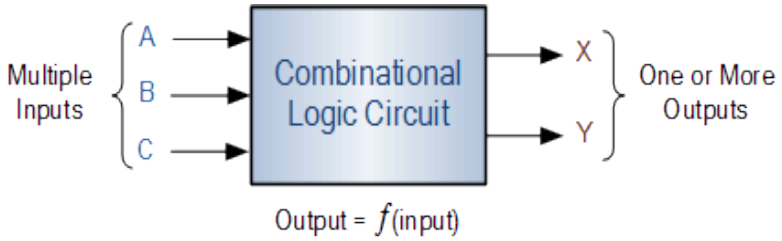
LOGIC DESIGN AND SWITCHING THEORY
CS-251

COMPUTER SCIENCE & INFORMATION TECHNOLOGY

SPRING 2023, Batch 2022

COMBINATIONAL LOGIC CIRCUITS

(ADDERS & SUBTRACTORS)

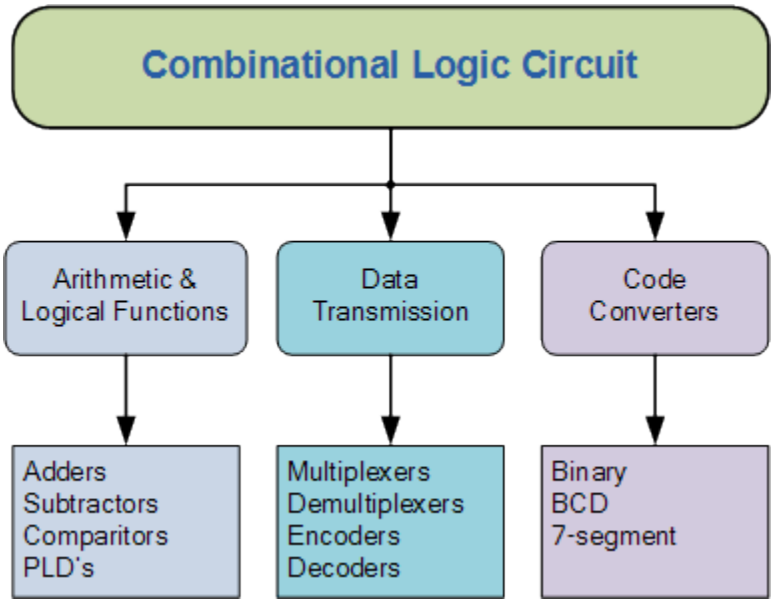


Combinational Logic Circuits are memoryless digital logic circuits whose output at any instant in time depends only on the combination of its inputs.

Unlike Sequential Logic Circuits whose outputs are dependent on both their present inputs and their previous output state giving them some form of Memory. The outputs of Combinational Logic Circuits are only determined by the logical function of their current input state, logic "0" or logic "1", at any given instant in time.

The result is that combinational logic circuits have no feedback, and any changes to the signals being applied to their inputs will immediately have an effect at the output. In other words, in a Combinational Logic Circuit, the output is dependant at all times on the combination of its inputs. Thus a combinational circuit is memoryless.

Classification of Combinational Logic



Half Adder

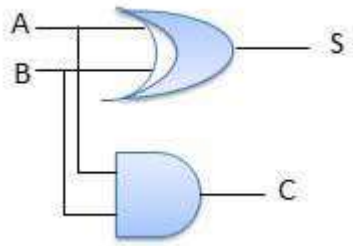
Half adder is a combinational logic circuit with two inputs and two outputs. The half adder circuit is designed to add two single bit binary number A and B. It is the basic building block for addition of two **single** bit numbers. This circuit has two outputs **carry** and **sum**.

Block diagram

Truth Table

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

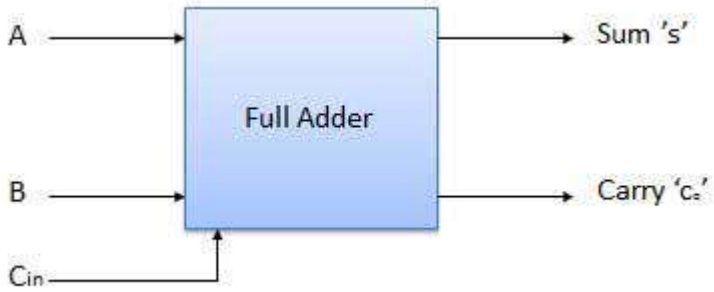
Circuit Diagram



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 combinational circuit.

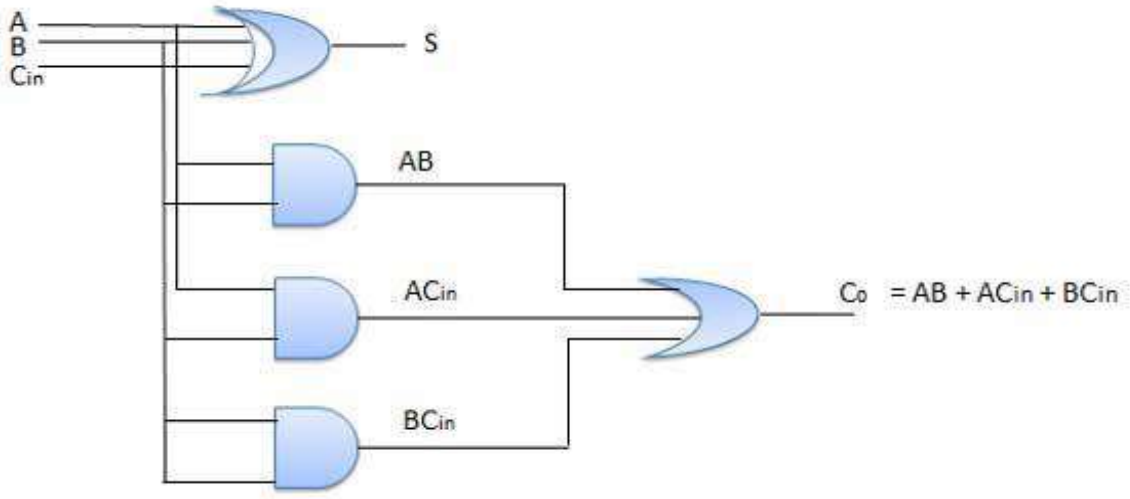
Block diagram



Truth Table

Inputs			Output	
A	B	Cin	S	Co
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

Circuit Diagram



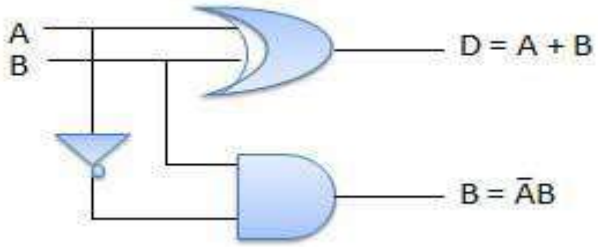
Half Subtractors

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.

Truth Table

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

Circuit Diagram



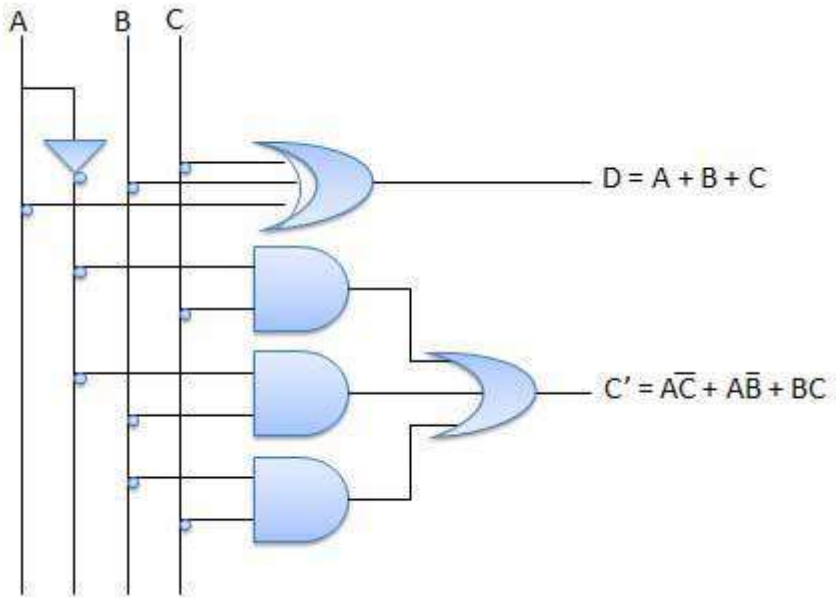
Full Subtractors

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.

Truth Table

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

Circuit Diagram



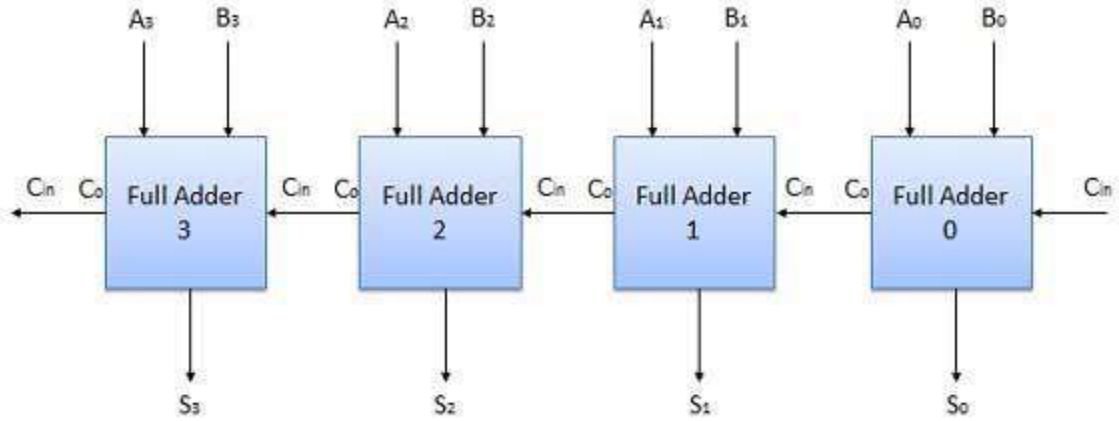
N-Bit Parallel Adder

The Full Adder is capable of adding only two single digit binary number along with a carry input. But in practical we need to add binary numbers which are much longer than just one bit. To add two n-bit binary numbers we need to use the n-bit parallel adder. It uses a number of full adders in cascade. The carry output of the previous full adder is connected to carry input of the next full adder.

4 Bit Parallel Adder/ Ripple Parallel Adder

In the block diagram, A0 and B0 represent the LSB of the four bit words A and B. Hence Full Adder-0 is the lowest stage. Hence its Cin has been permanently made 0. The rest of the connections are exactly same as those of n-bit parallel adder is shown in fig. The four bit parallel adder is a very common logic circuit.

Block diagram



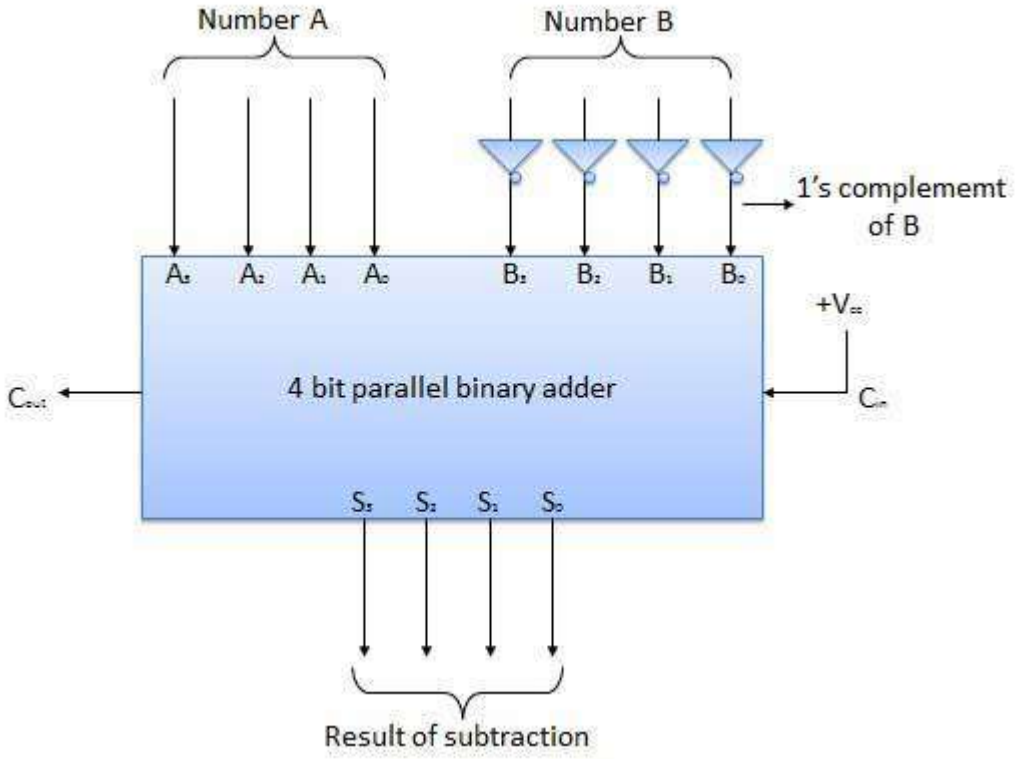
N-Bit Parallel Subtractor

The subtraction can be carried out by taking the 1's or 2's complement of the number to be subtracted. For example we can perform the subtraction (A-B) by adding either 1's or 2's complement of B to A. That means we can use a binary adder to perform the binary subtraction.

4 Bit Parallel Subtractor

The number to be subtracted (B) is first passed through inverters to obtain its 1's complement. The 4-bit adder then adds A and 2's complement of B to produce the subtraction. S3 S2 S1 S0 represents the result of binary subtraction (A-B) and carry output Cout represents the polarity of the result. If A > B then Cout = 0 and the result of binary form (A-B) then Cout = 1 and the result is in the 2's complement form.

Block diagram



4-bit binary Adder-Subtractor

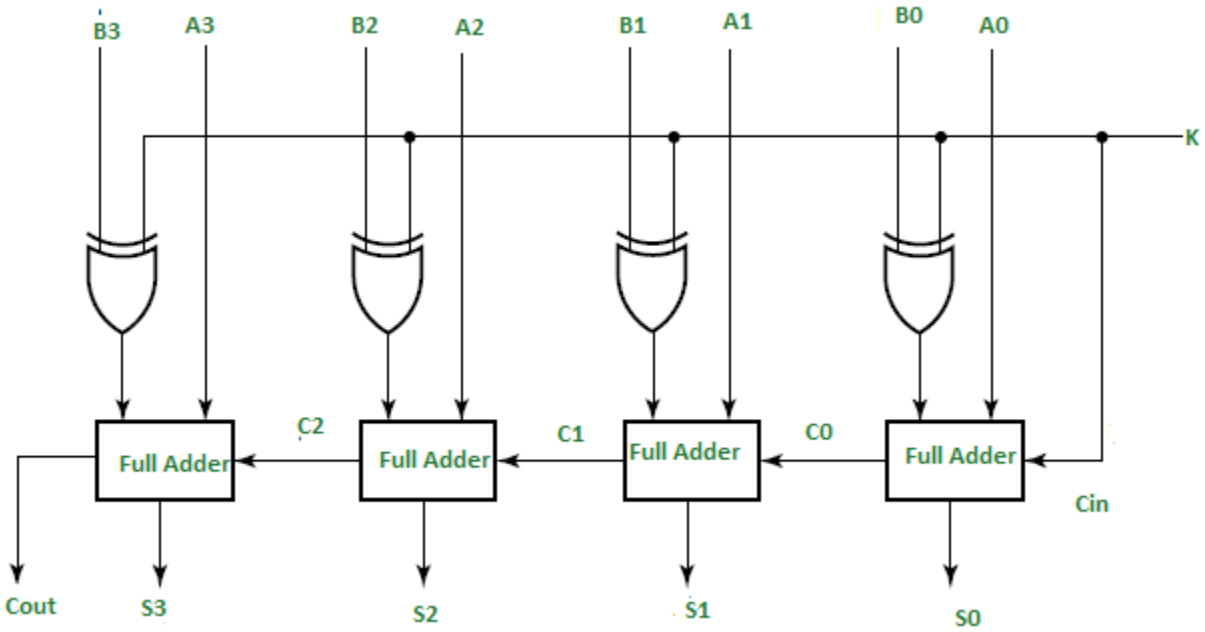
In Digital Circuits, A **Binary Adder-Subtractor** is capable of both the addition and subtraction of binary numbers in one circuit itself. The operation is performed depending on the binary value the control signal holds. It is one of the components of the ALU (Arithmetic Logic Unit).
This Circuit Requires prerequisite knowledge of Exor Gate, Binary Addition and Subtraction, and Full Adder.

Let's consider two 4-bit binary numbers A and B as inputs to the Digital Circuit for the operation with digits

A0 A1 A2 A3 for A

B0 B1 B2 B3 for B

The circuit consists of 4 full adders since we are performing operations on 4-bit numbers. There is a control line K that holds a binary value of either 0 or 1which determines that the operation is carried out is addition or subtraction.



As shown in the figure, the first full adder has a control line directly as its input(input carry Cin), The input A0 (The least significant bit of A) is directly input in the full adder. The third input is the exor of B0 and K. The two outputs produced are Sum/Difference (S0) and Carry (C0).

If the value of K (Control line) is 1, the output of B0(exor)K=B0′(Complement B0). Thus the operation would be A+(B0′). Now 2's complement subtraction for two numbers A and B is given by A+B'+Cin. This suggests that when K=1, the operation being performed on the four-bit numbers is subtraction.

Similarly If the Value of K=0, B0 (exor) K=B0. The operation is A+B which is s

simple binary addition. This suggests that When K=0, the operation is performed on the four-bit numbers in addition.

Then C0 is serially passed to the second full adder as one of it's outputs. The sum/difference S0 is recorded as the least significant bit of the sum/difference. A1, A2, A3 are direct inputs to the second, third and fourth full adders. Then the third input is the B1, B2, B3 EXORed with K to the second, third and fourth full adder respectively. The carry C1, C2 are serially passed to the successive full adder as one of the inputs. C3 becomes the total carry to the sum/difference. S1, S2, S3 are recorded to form the result with S0.

For an n-bit binary adder-subtractor, we use n number of full adders.

Carry Look Ahead Header

The drawback of ‘Ripple carry adder‘ is that it has a carry propagation delay that introduces slow computation. Since adders are used in designs like multipliers and divisions, it causes slowness in their computation. To tackle this issue, a carry look-ahead adder (CLA) can be used that reduces propagation delay with additional hardware complexity.

CLA has introduced some functions like ‘carry generate (G)’ and ‘carry propagate (P)’ to boost the speed.

Carry Generate (G): This function denotes how the carry is generated for single-bit two inputs regardless of any input carry.

As we have seen in the full adder, carry is generated using the equation as A.B.

Hence, G = A·B (similar to how carry is generated by full adder)

Carry Propagate (P): This function denotes when the carry is propagated to the next stage with an addition whenever there is an input carry.

Let’s consider single bit two inputs A and B.

A	B	Carry In	Description
0	0	1	Carry is not propagated (0)
0	1	1	Carry is propagated (1)
1	0	1	Carry is propagated (1)
1	1	1	Carry is propagated (1)

Thus, P = A+B

Carry computation function for next stage

$C_{j+1} = G_j + (P_j \cdot C_j)$

$= A_j \cdot B_j + (A_j + B_j) \cdot C_j$

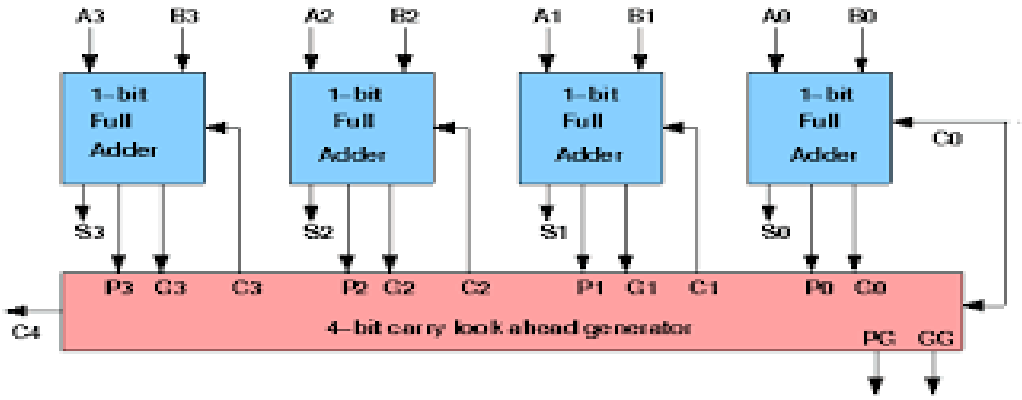
However, if you notice clearly whenever Aj=1 and Bj=1

Cj+1 = 1 (always) as Aj·Bj component nullifies effect of [(Aj + Bj)·Cj] part. Thus, it does not matter even if you use the equation propagate function as P = A (+) B as mentioned in other literature.

G = A·B

P = A + B or A (+) B

Block Diagram



Computation of all carry for next stage addition

Inputs –

B3 B2 B1 B0

And Cin

Output -

Sum = S3 S2 S1 S0

And Cout

Sum Sj = Pj ^ Cj = Aj ^ Bj ^ Cj

C0 = Cin

C1 = G0 + (P0·C0)

= A0·B0 + (A0^B0)·C0

C2 = G1 + (P1·C1)

= A1·B1 + (A1^B1)·(A0·B0 + (A0^B0)·C0)

C3 = G2 + (P2·C2)

= A2·B2 + (A2^B2)·C2

= A2·B2 + (A2^B2)·(A1.B1 + (A1^B1)·(A0B0 + (A0^B0)·C0))

C4 = G3 + (P3·C3)

= A3·B3 + (A3^B3)·C3

= A3·B3 + (A3^B3)·(A2.B2 + (A2^B2)·(A1.B1 + (A1^B1)·(A0B0 + (A0^B0)·C0)))

Cout = C4;

Notice that in each stage we are ultimately dependent on inputs A, B, and Cin. This actually makes it faster as compared to the ripple carry adder which is highly dependent on the carry generated from the previous stage.

The final implementation is brought down to XOR, AND, and OR gate level.

Disadvantage:

Requires complex hardware to generate the carry based on input signals and the complexity grows with an increasing number of bits.

Advantage:

It does not need to wait till the previous stage generates the carry. It can generate carry for each full adder simultaneously, hence it is faster as compared to the ripple carry adder.

LOGIC DESIGN AND SWITCHING THEORY
CS-251

COMPUTER SCIENCE & INFORMATION TECHNOLOGY

SPRING 2023, Batch 2022

COMBINATIONAL LOGIC CIRCUITS

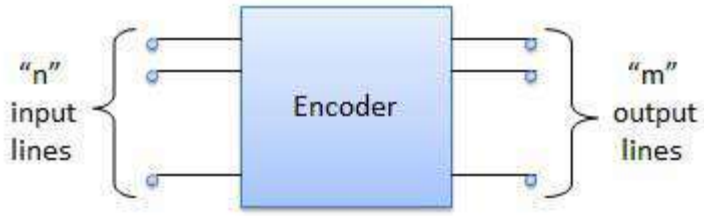
(ENCODERS & DECODERS)

Miss Maha Zaka

Encoder

Encoder is a combinational circuit which is designed to perform the inverse operation of the decoder. An encoder has n number of input lines and m number of output lines. An encoder produces an m bit binary code corresponding to the digital input number. The encoder accepts an n input digital word and converts it into an m bit another digital word.

Block diagram



Priority Encoder

Priority Encoders take all of their data inputs one at a time and converts them into an equivalent binary code at its output

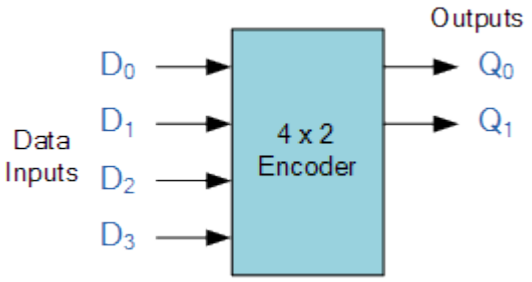
Unlike a multiplexer that selects one individual data input line and then sends that data to a single output line or switch. The job of a priority encoder is to produce a binary output address for the input with the highest priority.

The **Digital Encoder** more commonly called a **Binary Encoder** takes ALL its data inputs one at a time and then converts them into a single encoded output. So we can say that a binary encoder, is a multi-input combinational logic circuit that converts the logic level "1" data at its inputs into an equivalent binary code at its output.

Generally, digital encoders produce outputs of 2-bit, 3-bit or 4-bit codes depending upon the number of data input lines. An "n-bit" binary encoder has 2^n input lines and n-bit output lines with common types that include 4-to-2, 8-to-3 and 16-to-4 line configurations.

The output lines of a digital encoder generate the binary equivalent of the input line whose value is equal to "1" and are available to encode either a decimal or hexadecimal input pattern to typically a binary or "B.C.D" (binary coded decimal) output code.

4-to-2 Bit Binary Encoder



Inputs				Outputs	
D_3	D_2	D_1	D_0	Q_1	Q_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
0	0	0	0	x	x

One of the main disadvantages of standard digital encoders is that they can generate the wrong output code when there is more than one input present at logic level "1". For example, if we make inputs D_1 and D_2 HIGH at logic "1" both at the same time, the resulting output is neither at "01" or at "10" but will be at "11" which is an output binary number that is different to the actual input present. Also, an output code of all logic "0"s can be generated when all of its inputs are at "0" OR when input D_0 is equal to one.

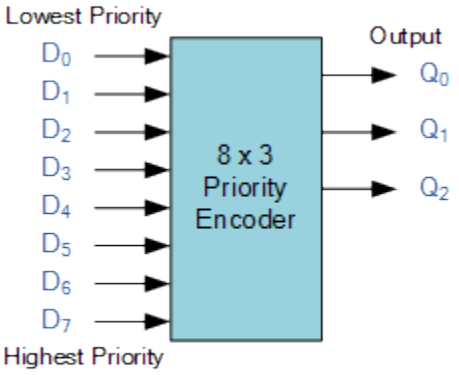
One simple way to overcome this problem is to "Prioritise" the level of each input pin. So if there is more than one input at logic level "1" at the same time, the actual output code would only correspond to the input with the highest designated priority. Then this type of digital encoder is known commonly as a **Priority Encoder** or **P-encoder** for short.

Priority Encoder

The **Priority Encoder** solves the problems mentioned above by allocating a priority level to each input. The *priority encoders* output corresponds to the currently active input which has the highest priority. So when an input with a higher priority is present, all other inputs with a lower priority will be ignored.

The priority encoder comes in many different forms with an example of an 8-input priority encoder along with its truth table shown below.

8-to-3 Bit Priority Encoder



Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

X = dont care

Priority encoders are available in standard IC form and the TTL 74LS148 is an 8-to-3 bit priority encoder which has eight active LOW (logic "0") inputs and provides a 3-bit code of the highest ranked input at its output.

Priority encoders output the highest order input first for example, if input lines "D2", "D3" and "D5" are applied simultaneously the output code would be for input "D5" ("101") as this has the highest order out of the 3 inputs. Once input "D5" had been removed the next highest output code would be for input "D3" ("011"), and so on.

The truth table for a 8-to-3 bit priority encoder is given as:

Digital Inputs								Binary Output		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	**1**	0	0	0
0	0	0	0	0	0	**1**	X	0	0	1

0	0	0	0	0	**1**	X	X	0	1	0
0	0	0	0	**1**	X	X	X	0	1	1
0	0	0	**1**	X	X	X	X	1	0	0
0	0	**1**	X	X	X	X	X	1	0	1
0	**1**	X	X	X	X	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1

Where X equals “dont care”, that is it can be at a logic “0” level or at a logic “1” level.

From this truth table, the Boolean expression for the encoder above with data inputs D_0 to D_7 and outputs Q_0, Q_1, Q_2 is given as:

Output Q_0

$$Q_0 = \sum(1, 3, 5, 7)$$

$$Q_0 = \sum\left(\overline{D}_7\overline{D}_6\overline{D}_5\overline{D}_4\overline{D}_3\overline{D}_2D_1 + \overline{D}_7\overline{D}_6\overline{D}_5\overline{D}_4D_3 + \overline{D}_7\overline{D}_6D_5 + D_7\right)$$

$$Q_0 = \sum\left(\overline{D}_6\overline{D}_4\overline{D}_2D_1 + \overline{D}_6\overline{D}_4D_3 + \overline{D}_6D_5 + D_7\right)$$

$$Q_0 = \sum\left(\overline{D}_6\left(\overline{D}_4\overline{D}_2D_1 + \overline{D}_4D_3 + D_5\right) + D_7\right)$$

Output Q_1

$$Q_1 = \sum(2, 3, 6, 7)$$

$$Q_1 = \sum(\bar{D}_7\bar{D}_6\bar{D}_5\bar{D}_4\bar{D}_3D_2 + \bar{D}_7\bar{D}_6\bar{D}_5\bar{D}_4D_3 + \bar{D}_7D_6 + D_7)$$

$$Q_1 = \sum(\bar{D}_5\bar{D}_4D_2 + \bar{D}_5\bar{D}_4D_3 + D_6 + D_7)$$

$$Q_1 = \sum(\bar{D}_5\bar{D}_4(D_2 + D_3) + D_6 + D_7)$$

Output Q_2

$$Q_2 = \sum(4, 5, 6, 7)$$

$$Q_2 = \sum(\bar{D}_7\bar{D}_6\bar{D}_5D_4 + \bar{D}_7\bar{D}_6D_5 + \bar{D}_7D_6 + D_7)$$

$$Q_2 = \sum(D_4 + D_5 + D_6 + D_7)$$

Then the final Boolean expression for the priority encoder including the zero inputs is defined as:

Priority Encoder Output Expression

$$Q_0 = \sum(\bar{D}_6(\bar{D}_4\bar{D}_2D_1 + \bar{D}_4D_3 + D_5) + D_7)$$

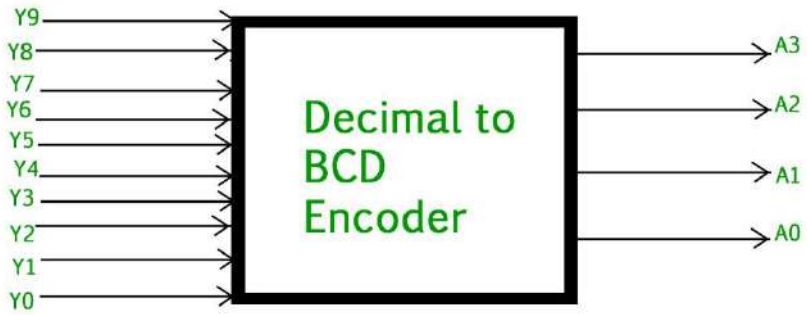
$$Q_1 = \sum(\bar{D}_5\bar{D}_4(D_2 + D_3) + D_6 + D_7)$$

$$Q_2 = \sum(D_4 + D_5 + D_6 + D_7)$$

In practice these zero inputs would be ignored allowing the implementation of the final Boolean expression for the outputs of the 8-to-3 **priority encoder**. We can constructed a simple encoder from the expression above using individual OR gates as follows

Decimal to BCD Encoder

The decimal-to-binary encoder usually consists of **10 input lines** and **4 output lines**. Each input line corresponds to each decimal digit and 4 outputs correspond to the BCD code. This encoder accepts the decoded decimal data as an input and encodes it to the BCD output which is available on the output lines. The figure below shows the logic symbol of the decimal to BCD encoder :



Decimal to BCD Encoder

The truth table for decimal to BCD encoder is as follows.

INPUTS										**OUTPUTS**			
Y9	**Y8**	**Y7**	**Y6**	**Y5**	**Y4**	**Y3**	**Y2**	**Y1**	**Y0**	**A3**	**A2**	**A1**	**A0**
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0

INPUTS										OUTPUTS			
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1

Logical expression for A3, A2, A1, and A0.

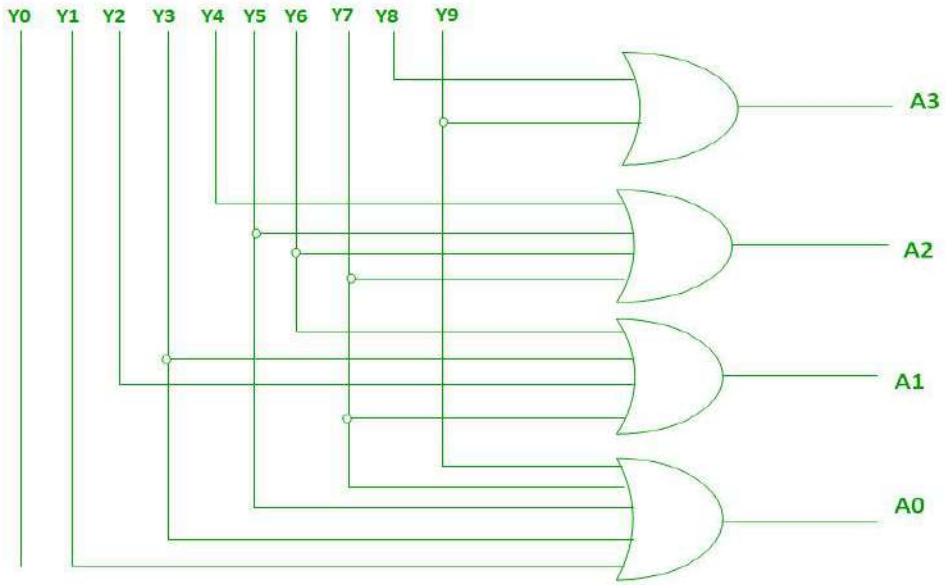
A3 = Y9 + Y8

A2 = Y7 + Y6 + Y5 +Y4

A1 = Y7 + Y6 + Y3 +Y2

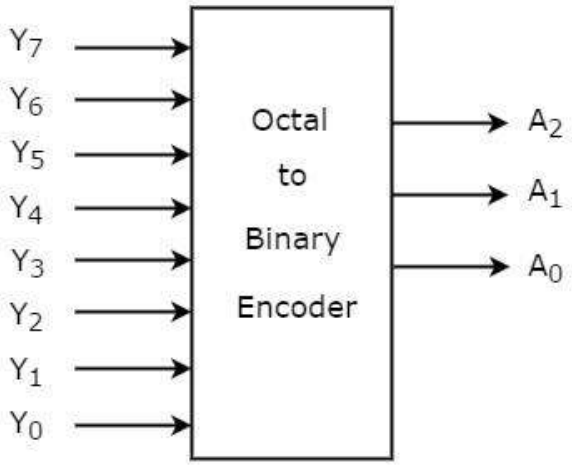
A0 = Y9 + Y7 +Y5 +Y3 + Y1

The above two Boolean functions can be implemented using OR gates.



Octal to Binary Encoder

Octal to binary Encoder has eight inputs, Y_7 to Y_0 and three outputs A_2, A_1 & A_0. Octal to binary encoder is nothing but 8 to 3 encoder. The **block diagram** of octal to binary Encoder is shown in the following figure.



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

Inputs								**Outputs**		
Y_7	**Y_6**	**Y_5**	**Y_4**	**Y_3**	**Y_2**	**Y_1**	**Y_0**	**A_2**	**A_1**	**A_0**
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1

0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

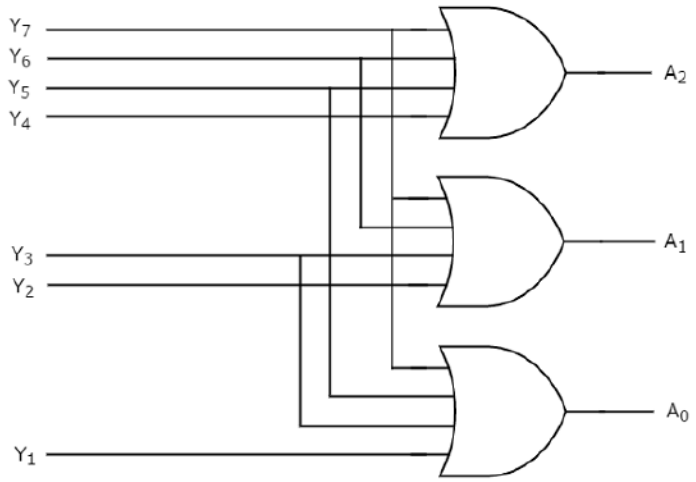
From Truth table, we can write the **Boolean functions** for each output as

$$A2=Y7+Y6+Y5+Y4$$

$$A1=Y7+Y6+Y3+Y2$$

$$A0=Y7+Y5+Y3+Y1$$

We can implement the above Boolean functions by using four input OR gates. The **circuit diagram** of octal to binary encoder is shown in the following figure.



The above circuit diagram contains three 4-input OR gates. These OR gates encode the eight inputs with three bits.

Magnitude Comparator

A combinational circuit that compares two digital or binary numbers and compare them in the form of A < B, A = B & A > B is known as magnitude comparator. A Magnitude comparator is also known as digital comparator.

1 Bit Magnitude Comparator

A magnitude comparator that compares two bits is known as 1 bit comparator. It consist of two inputs (A & sB) each of 1 bit and has three outputs. The three outputs are A is less than B (A < B) , A is equal to B (A = B) and A is greater than B (A > B).

1 Bit Comparator Truth Table

Inputs		Outputs		
B	A	A > B	A = B	A < B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

The output expression is written in SOP form (A = 1 & A’ = 0), similarly (B = 1 & B’ = 0).The output is written only when logic 1 is obtained in the output.

Expression for A < B

Y = A’B

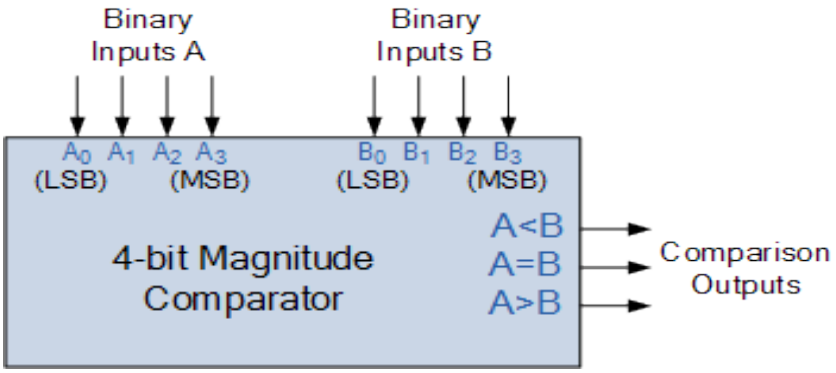
Expression for A = B

Y = A’B’ + AB

Expression for A > B

Y = AB’

4-bit Magnitude Comparator

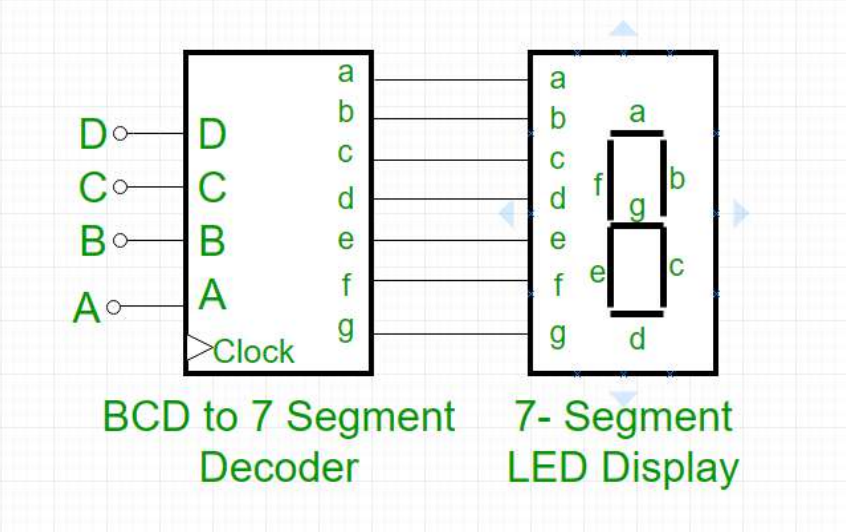


Some commercially available digital comparators such as the TTL 74LS85 or CMOS 4063 4-bit magnitude comparator have additional input terminals that allow more individual comparators to be "cascaded" together to compare words larger than 4-bits with magnitude comparators of "n"-bits being produced. These cascading inputs are connected directly to the corresponding outputs of the previous comparator as shown to compare 8, 16 or even 32-bit words.

BCD to 7 Segment Decoder

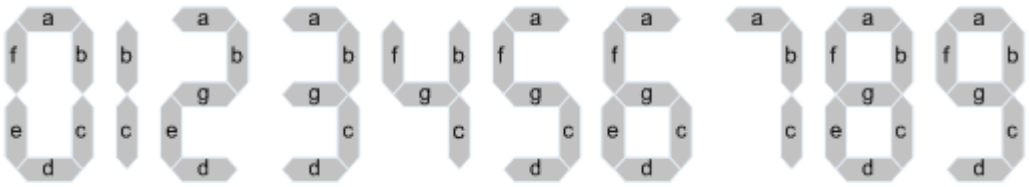
In **Binary Coded Decimal (BCD)** encoding scheme each of the decimal numbers(0-9) is represented by its equivalent binary pattern(which is generally of 4-bits).
Whereas, **Seven segment** display is an electronic device which consists of seven Light Emitting Diodes (LEDs) arranged in a some definite pattern (common cathode or common anode type), which is used to display Hexadecimal numerals(in this case decimal numbers, as input is BCD i.e., 0-9).
Two types of seven segment LED display:

1. **Common Cathode Type:** In this type of display all cathodes of the seven LEDs are connected together to the ground or -Vcc(hence, common cathode) and LED displays digits when some 'HIGH' signal is supplied to the individual anodes.
2. **Common Anode Type:** In this type of display all the anodes of the seven LEDs are connected to battery or +Vcc and LED displays digits when some 'LOW' signal is supplied to the individual cathodes.



Truth Table – For common cathode type BCD to seven segment decoder:

A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1



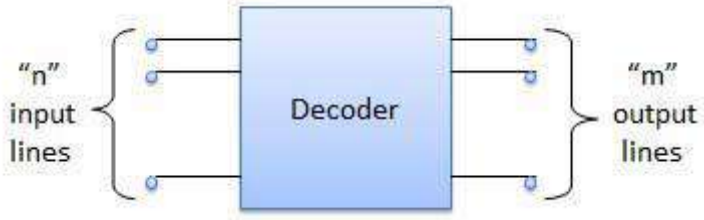
7-Segment Display Elements for all Numbers.

;

Decoder:

A decoder is a combinational circuit. It has n input and to a maximum m = 2n outputs. Decoder is identical to a demultiplexer without any data input. It performs operations which are exactly opposite to those of an encoder.

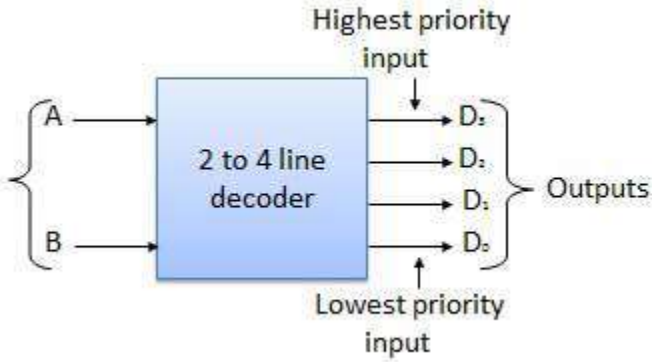
Block diagram



2 to 4 Line Decoder

The block diagram of 2 to 4 line decoder is shown in the fig. A and B are the two inputs where D through D are the four outputs. Truth table explains the operations of a decoder. It shows that each output is 1 for only a specific combination of inputs.

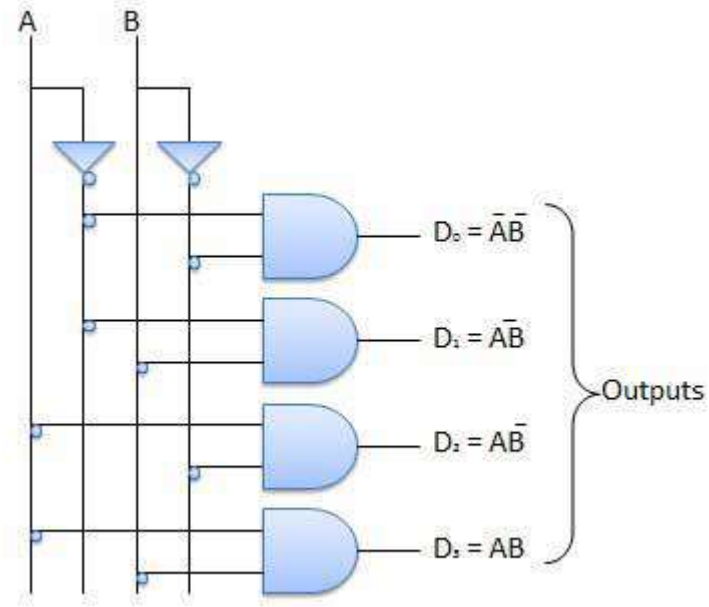
Block diagram



Truth Table

Inputs		Output			
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
0	1	0	0	1	0
1	1	0	0	0	1

Logic Circuit



Implementation of Higher-order Decoders

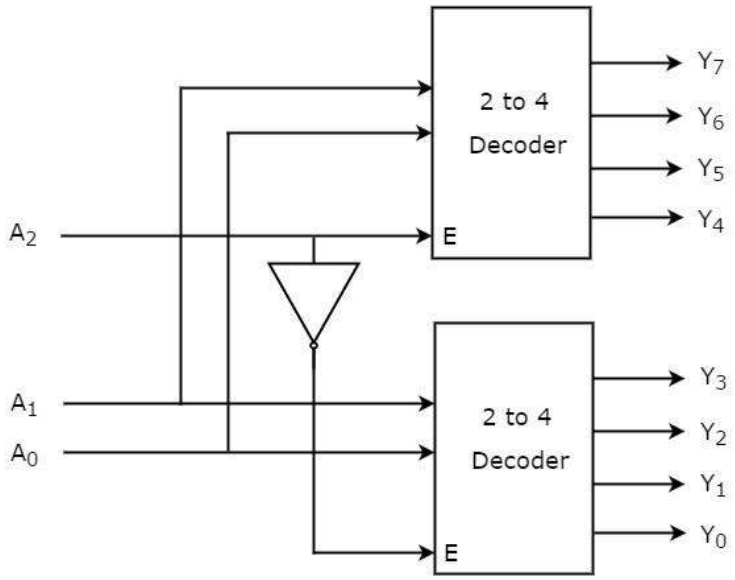
Now, let us implement the following two higher-order decoders using lower-order decoders.

- 3 to 8 decoder
- 4 to 16 decoder

➢ 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.

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.

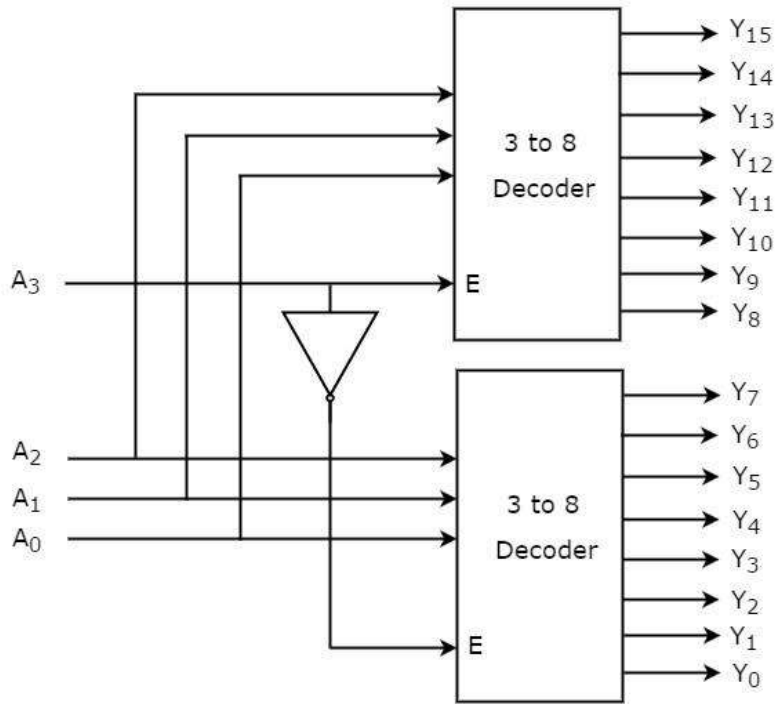


The parallel inputs A_1 & A_0 are applied to each 2 to 4 decoder. The complement of input A_2 is connected to Enable, E of lower 2 to 4 decoder in order to get the outputs, Y_3 to Y_0. These are the **lower four min terms**. The input, A_2 is directly connected to Enable, E of upper 2 to 4 decoder in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

- **4 to 16 Decoder**

In this section, let us implement **4 to 16 decoder using 3 to 8 decoders**. We know that 3 to 8 Decoder has three inputs A_2, A_1 & A_0 and eight outputs, Y_7 to Y_0. Whereas, 4 to 16 Decoder has four inputs A_3, A_2, A_1 & A_0 and sixteen outputs, Y_{15} to Y_0.

Therefore, we require two 3 to 8 decoders for implementing one 4 to 16 decoder. The **block diagram** of 4 to 16 decoder using 3 to 8 decoders is shown in the following figure.



The parallel inputs A_2, A_1 & A_0 are applied to each 3 to 8 decoder. The complement of input, A3 is connected to Enable, E of lower 3 to 8 decoder in order to get the outputs, Y_7 to Y_0. These are the **lower eight min terms**. The input, A_3 is directly connected to Enable, E of upper 3 to 8 decoder in order to get the outputs, Y_{15} to Y_8. These are the **higher eight min terms**.

LOGIC DESIGN AND SWITCHING THEORY
CS-251

COMPUTER SCIENCE & INFORMATION TECHNOLOGY

SPRING 2023, Batch 2022

COMBINATIONAL LOGIC CIRCUITS

(Multiplexers And DEmultiplexers)

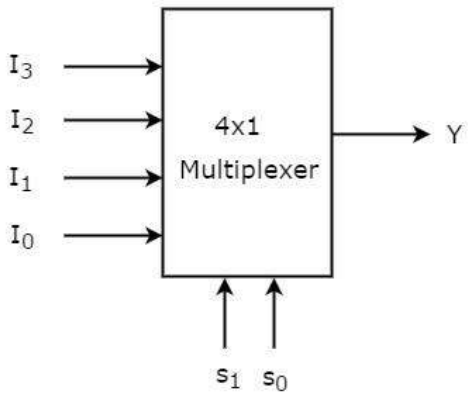
Miss Maha Zaka

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.



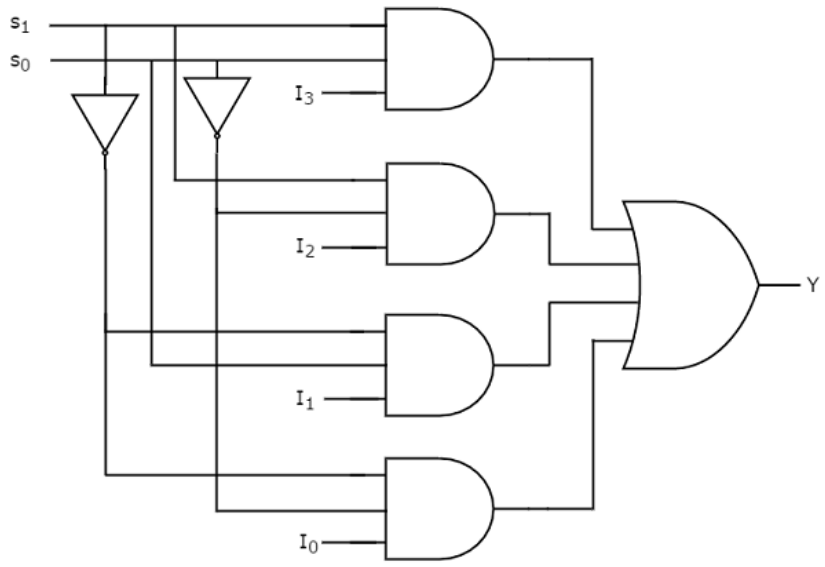
One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. **Truth table** of 4x1 Multiplexer is shown below.

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

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

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

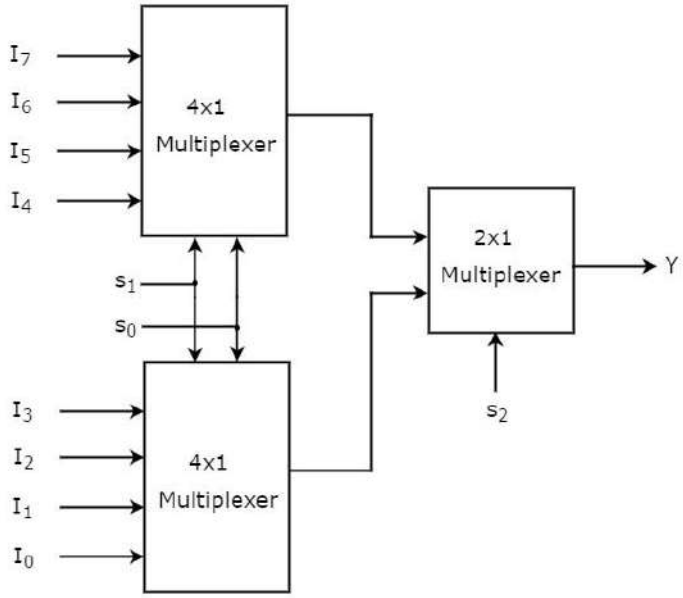
In this section, let us implement 8x1 Multiplexer using 4x1 Multiplexers and 2x1 Multiplexer. We know that 4x1 Multiplexer has 4 data inputs, 2 selection lines and one output. Whereas, 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output.

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

We can implement 8x1 Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 8x1 Multiplexer is shown in the following figure.



The same **selection lines, s_1 & s_0** are applied to both 4x1 Multiplexers. The data inputs of upper 4x1 Multiplexer are I_7 to I_4 and the data inputs of lower 4x1 Multiplexer are I_3 to I_0. Therefore, each 4x1 Multiplexer produces an output based on the values of selection lines, s_1 & s_0.

The outputs of first stage 4x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, s_2** is applied to 2x1 Multiplexer.

- If s_2 is zero, then the output of 2x1 Multiplexer will be one of the 4 inputs I_3 to I_0 based on the values of selection lines s_1 & s_0.
- If s_2 is one, then the output of 2x1 Multiplexer will be one of the 4 inputs I_7 to I_4 based on the values of selection lines s_1 & s_0.

Therefore, the overall combination of two 4x1 Multiplexers and one 2x1 Multiplexer performs as one 8x1 Multiplexer.

➢ 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.

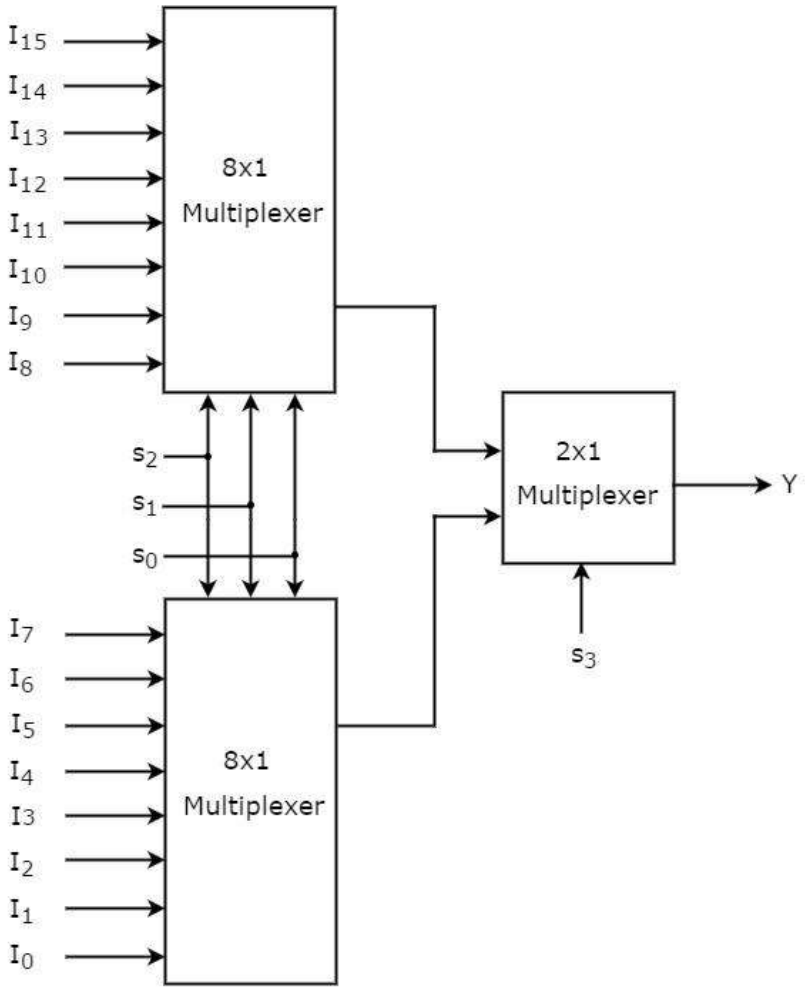
So, we require two **8x1 Multiplexers** in first stage in order to get the 16 data inputs. Since, each 8x1 Multiplexer produces one output, we require a 2x1 Multiplexer in second stage by considering the outputs of first stage as inputs and to produce the final output.

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.

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}

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.



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.

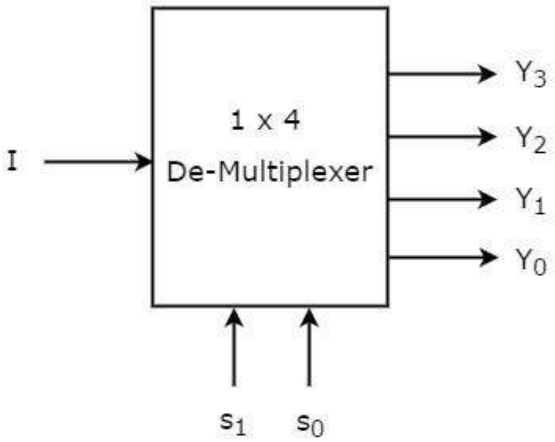
Demultiplexer

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

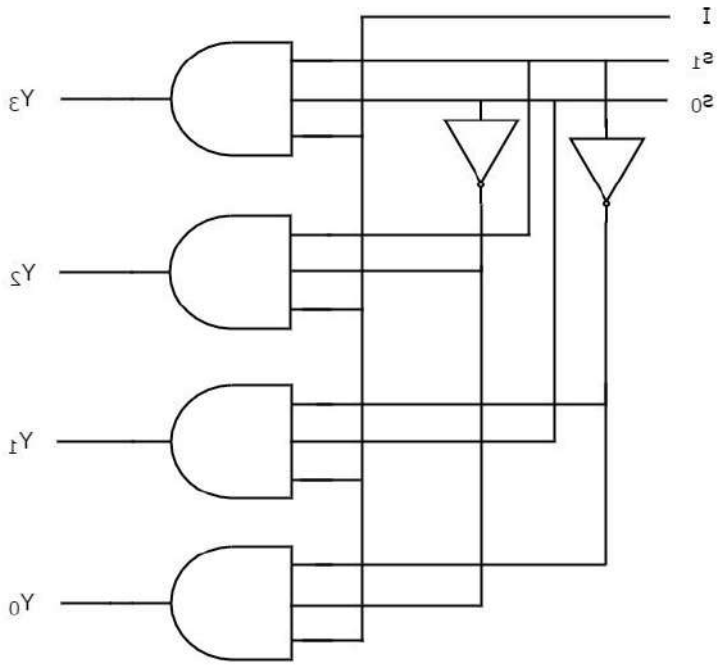
Y3=s1s0I

Y2=s1s0′I

Y1=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

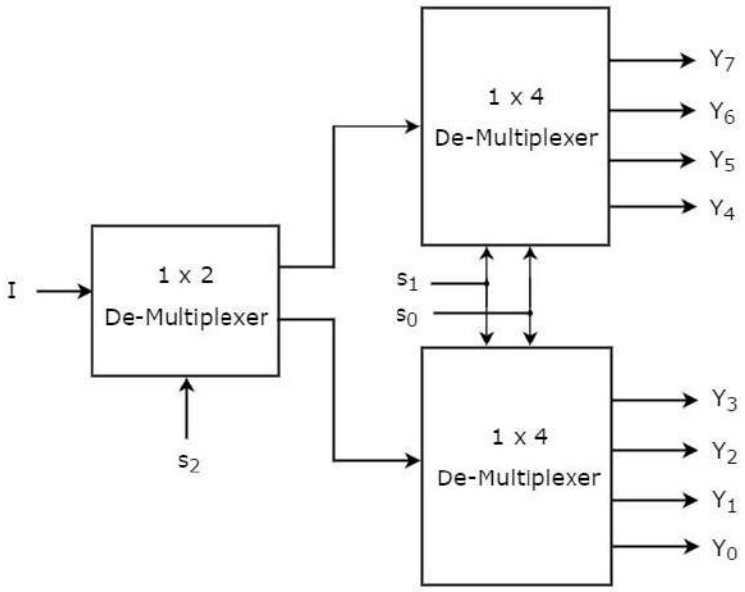
In this section, let us implement 1x8 De-Multiplexer using 1x4 De-Multiplexers and 1x2 De-Multiplexer. We know that 1x4 De-Multiplexer has single input, two selection lines and four outputs. Whereas, 1x8 De-Multiplexer has single input, three selection lines and eight outputs.

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.

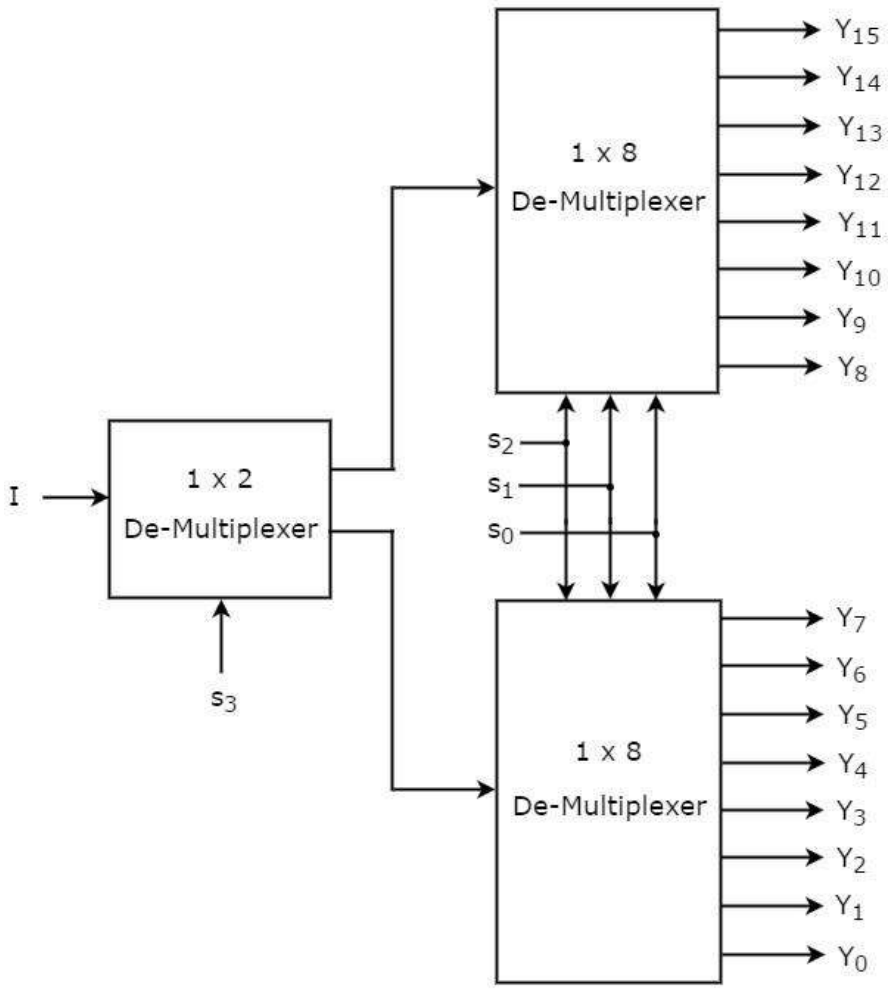
The other **selection line, s_2** is applied to 1x2 De-Multiplexer. If s_2 is zero, then one of the four outputs of lower 1x4 De-Multiplexer will be equal to input, I based on the values of selection lines s_1 & s_0. Similarly, if s_2 is one, then one of the four outputs of upper 1x4 DeMultiplexer will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

LOGIC DESIGN AND SWITCHING THEORY
CS-251

COMPUTER SCIENCE & INFORMATION TECHNOLOGY

SPRING 2023, Batch 2022

SEQUENTIAL LOGIC CIRCUITS

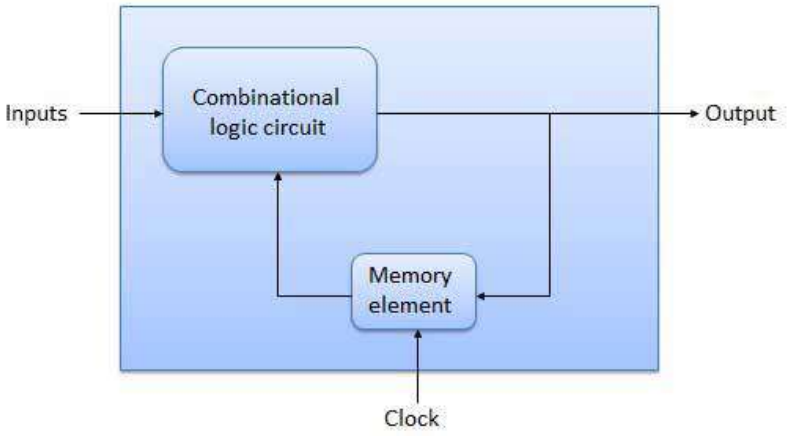
(LATCHES & FLIPFLOPS)

Miss Maha Zaka

SEQUENTIAL LOGIC CIRCUITS

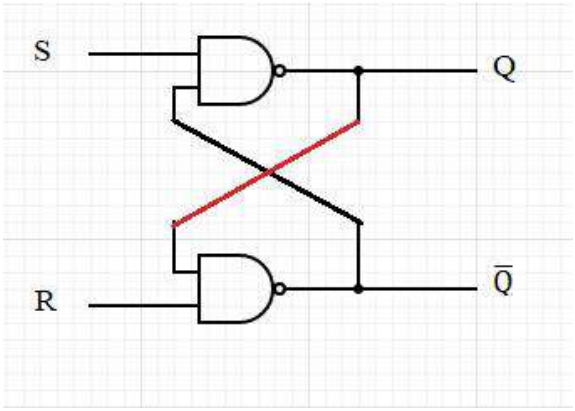
The combinational circuit does not use any memory. Hence the previous state of input does not have any effect on the present state of the circuit. But sequential circuit has memory so output can vary based on input. This type of circuits uses previous input, output, clock and a memory element.

Block diagram



➢ SR latch using NAND gate:

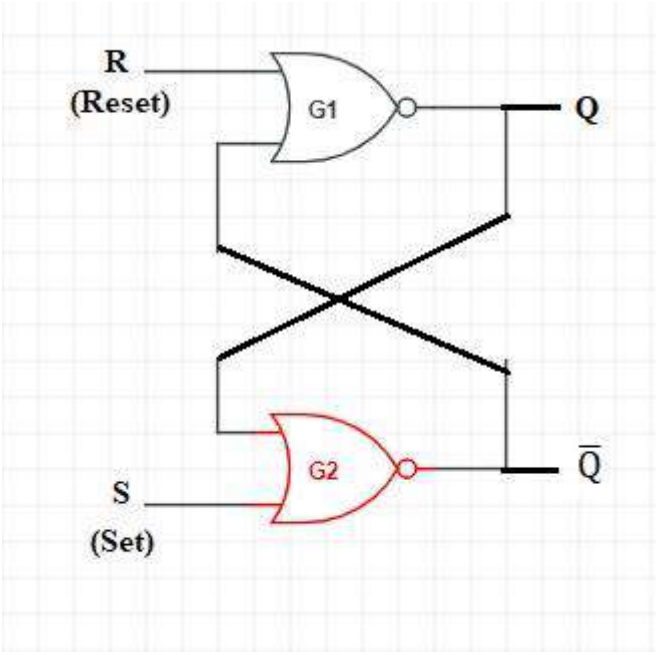
In SR latch using NAND gate we will replace NOR gate with the NAND gate. The inputs are interchange in SR NOR latch we have reset in the upward gate and set in the lower gate. While in this circuit we are applying set to the upper NAND gate and reset to the lower NAND gate and Q and $\bar{Q}$ represents the output of the latch. The circuit will be set when the Q = 1 and the circuit will be reset when the Q = 0.



S	R	Q	Q^-
0	0	Not used	
0	1	1	0
1	0	0	1
1	1	Previous state	

➢ SR Latch using NOR gates:

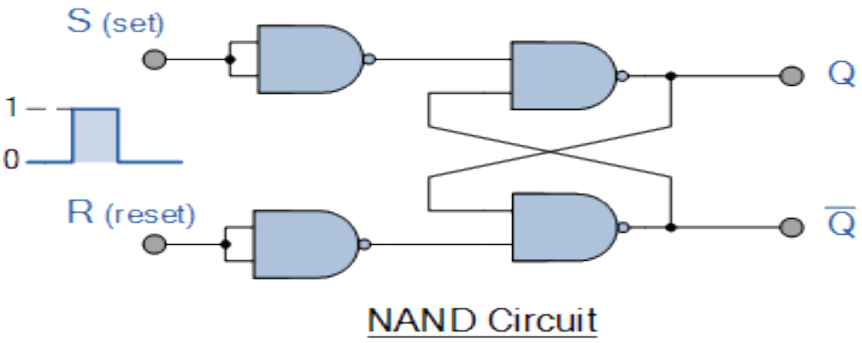
sr flip flop:- Latch is basic storage element in which we store 0 or 1. Latch as name suggest it holds 0 or 1. In the circuit "R" stands for reset and "S" stand for set. Q and are the output of the latch. When the circuit will be reset Q value will be equal to 0 and when the circuit will be set the Q value will be equal to 1.



S

S	R	Q	$\overline{Q}$
0	0	Previous state	
0	1	0	1
1	0	1	0
1	1	Not used	

➢ Positive NAND Gate SR Flip-flop

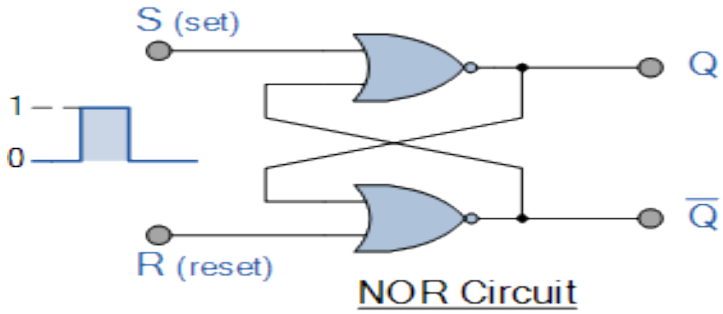


NAND Circuit

S	R	Q	$\overline{Q}$
0	0	No change	
0	1	0	1
1	0	1	0
1	1	X (Invalid)	X

As well as using NAND gates, it is also possible to construct simple one-bit **SR Flip-flops** using two cross-coupled NOR gates connected in the same configuration. The circuit will work in a similar way to the NAND gate circuit above, except that the inputs are active HIGH and the invalid condition exists when both its inputs are at logic level "1", and this is shown below.

➢ The NOR Gate SR Flip-flop



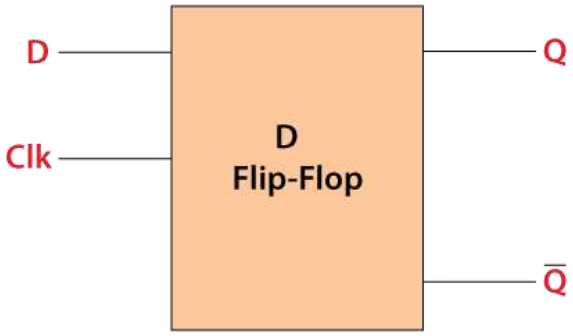
NOR Circuit

S	R	Q	$\overline{Q}$
0	0	No change	
0	1	1	0
1	0	0	1
1	1	X (Invalid)	X

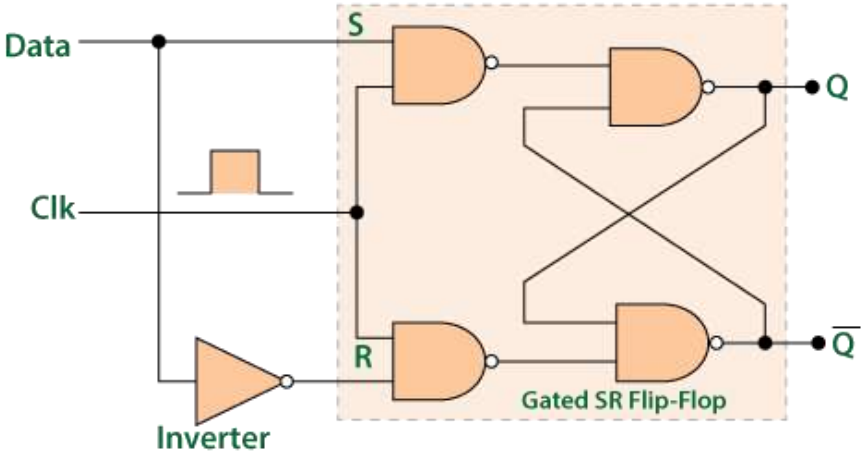
➢ D flip flop

The D flip flop is the most important flip flop from other clocked types. It ensures that at the same time, both the inputs, i.e., S and R, are never equal to 1. The Delay flip-flop is designed using a gated SR flip-flop with an inverter connected between the inputs allowing for a single input D(Data).

Block Diagram



Circuit Diagram



When the clock input is set to 1, the "set" and "reset" inputs of the flip-flop are both set to 1. So it will not change the state and store the data present on its output before the clock transition occurred. In simple words, the output is "latched" at either 0 or 1.

Truth Table for the D-type Flip Flop

Clock	D	Q	Q'	Description
↓ » 0	X	Q	Q'	Memory no change
↑ » 1	0	0	1	Reset Q » 0
↑ » 1	1	1	0	Set Q » 1

➢ JK Flip Flop

The SR Flip Flop or Set-Reset flip flop has lots of advantages. But, it has the following switching problems:

- When Set 'S' and Reset 'R' inputs are set to 0, this condition is always avoided.
- When the Set or Reset input changes their state while the enable input is 1, the incorrect latching action occurs.

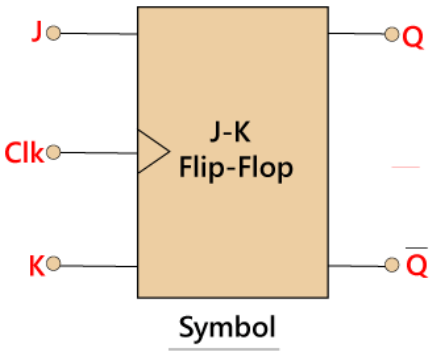
The JK Flip Flop removes these two drawbacks of SR Flip Flop.

The JK flip flop is one of the most used flip flops in digital circuits. The JK flip flop is a universal flip flop having two inputs 'J' and 'K'. In SR flip flop, the 'S' and 'R' are the shortened abbreviated letters for Set and Reset, but J and K are not. The J and K are themselves autonomous letters which are chosen to distinguish the flip flop design from other types.

The JK flip flop work in the same way as the SR flip flop work. The JK flip flop has 'J' and 'K' flip flop instead of 'S' and 'R'. The only difference between JK flip flop and SR flip flop is that when both inputs of SR flip flop is set to 1, the circuit produces the invalid states as outputs, but in case of JK flip flop, there are no invalid states even if both 'J' and 'K' flip flops are set to 1.

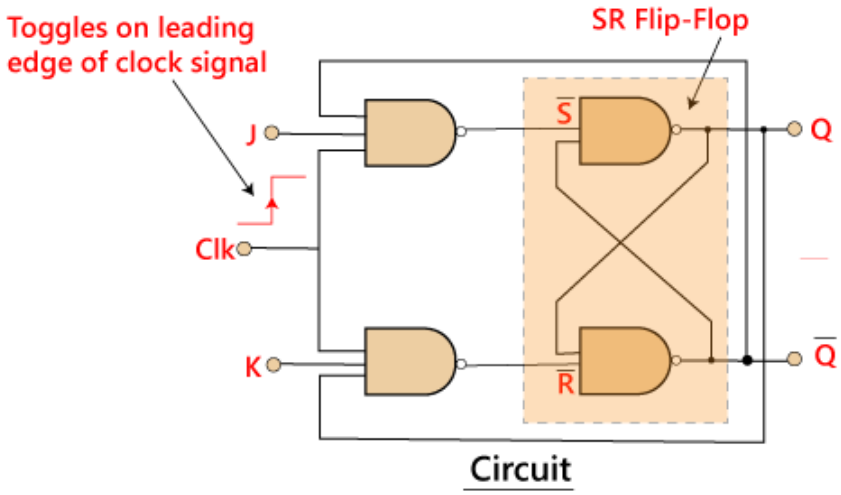
The JK Flip Flop is a gated SR flip-flop having the addition of a clock input circuitry. The invalid or illegal output condition occurs when both of the inputs are set to 1 and are prevented by the addition of a clock input circuit. So, the JK flip-flop has four possible input combinations, i.e., 1, 0, "no change" and "toggle". The symbol of JK flip flop is the same as **SR Bistable Latch** except for the addition of a clock input.

Block Diagram:

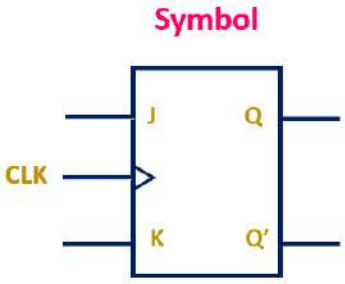


S

Circuit Diagram:



Truth Table for JK FlipFlop



Truth Table

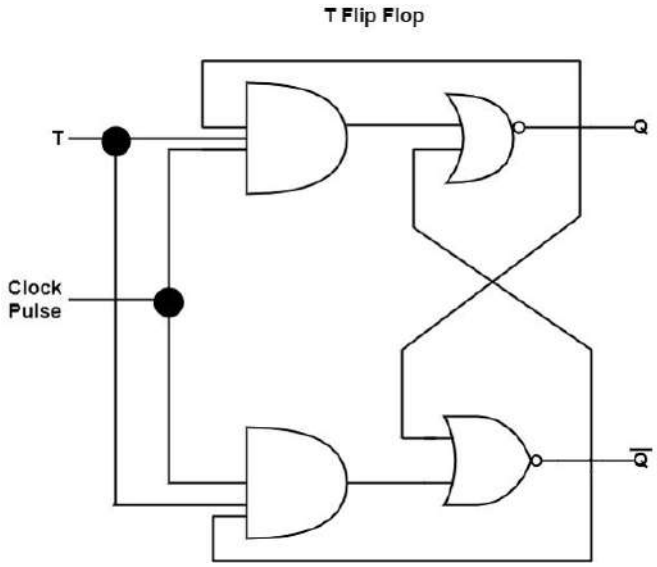
CLK	J	K	Q_{n+1}
↑	0	0	Q_n
↑	0	1	0
↑	1	0	1
↑	1	1	Q_n'

.

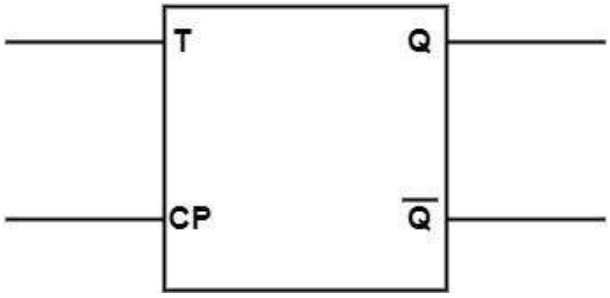
➢ T flip-flop

The T flip-flop is also called toggle flip-flop. It is a change of the JK flip-flop. The T flip flop is received by relating both inputs of a JK flip-flop. The T flip-flop is received by relating the inputs 'J' and 'K'. When T = 0, both AND gates are disabled. Therefore, there is no change in the output. When T= 1, the output toggles.

The diagram demonstrates the circuit diagram of a T flip-flop.



The logic symbol of the T flip-flop is shown in the figure.



Truth Table for T FlipFlop

Q_N	T	Q_{N+1}
0	0	0
0	1	1
1	0	1
1	1	0

LOGIC DESIGN AND SWITCHING THEORY
CS-251

COMPUTER SCIENCE & INFORMATION TECHNOLOGY

SPRING 2023, Batch 2022

SEQUENTIAL LOGIC CIRCUITS

COUNTERS & REGISTERS

Miss Maha Zaka

A **Counter** is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal. Counters are used in digital electronics for counting purpose, they can count specific event happening in the circuit. For example, in UP counter a counter increases count for every rising edge of clock. Not only counting, a counter can follow the certain sequence based on our design like any random sequence 0,1,3,2… .They can also be designed with the help of flip flops. They are used as frequency dividers where the frequency of given pulse waveform is divided. Counters are sequential circuit that count the number of pulses can be either in binary code or BCD form. The main properties of a counter are timing, sequencing and counting. Counter works in two modes.

Up counter

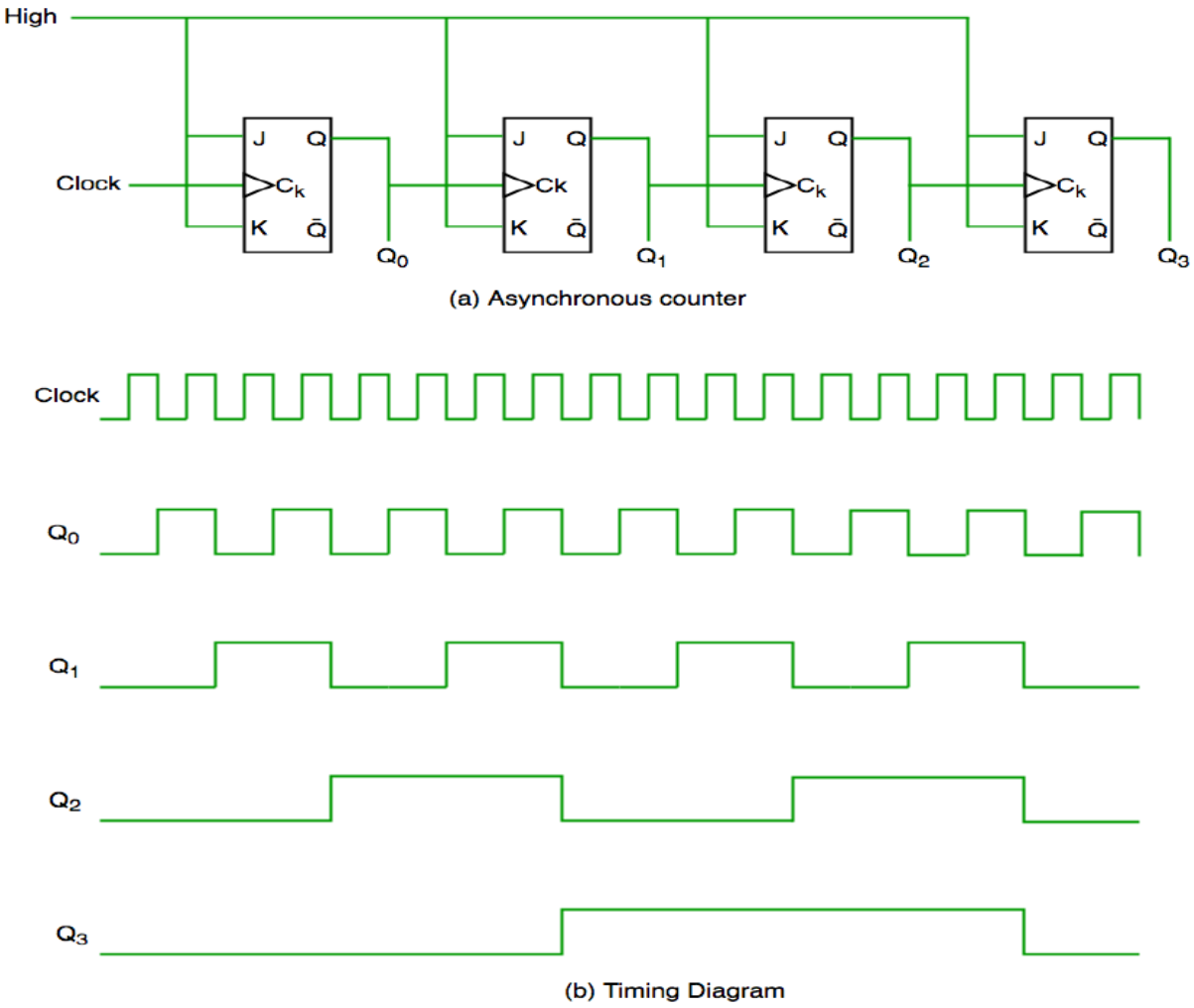
Down counter

Counters are broadly divided into two categories

1. Asynchronous counter
2. Synchronous counter

1. Asynchronous Counter

In asynchronous counter we don't use universal clock, only first flip flop is driven by main clock and the clock input of rest of the following flip flop is driven by output of previous flip flops. We can understand it by following diagram-

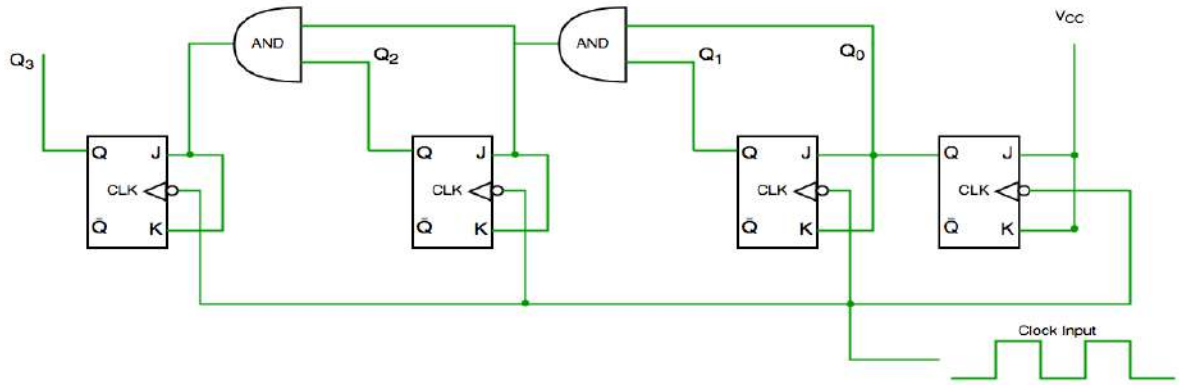


It is evident from timing diagram that Q0 is changing as soon as the rising edge of clock pulse is encountered, Q1 is changing when rising edge of Q0 is encountered(because Q0 is like clock pulse for second flip flop) and so on. In this way ripples are generated through Q0,Q1,Q2,Q3 hence it is also called **RIPPLE counter and serial counter.** A ripple counter is a cascaded arrangement of flip flops where the output of one flip flop drives the clock input of the following flip flop

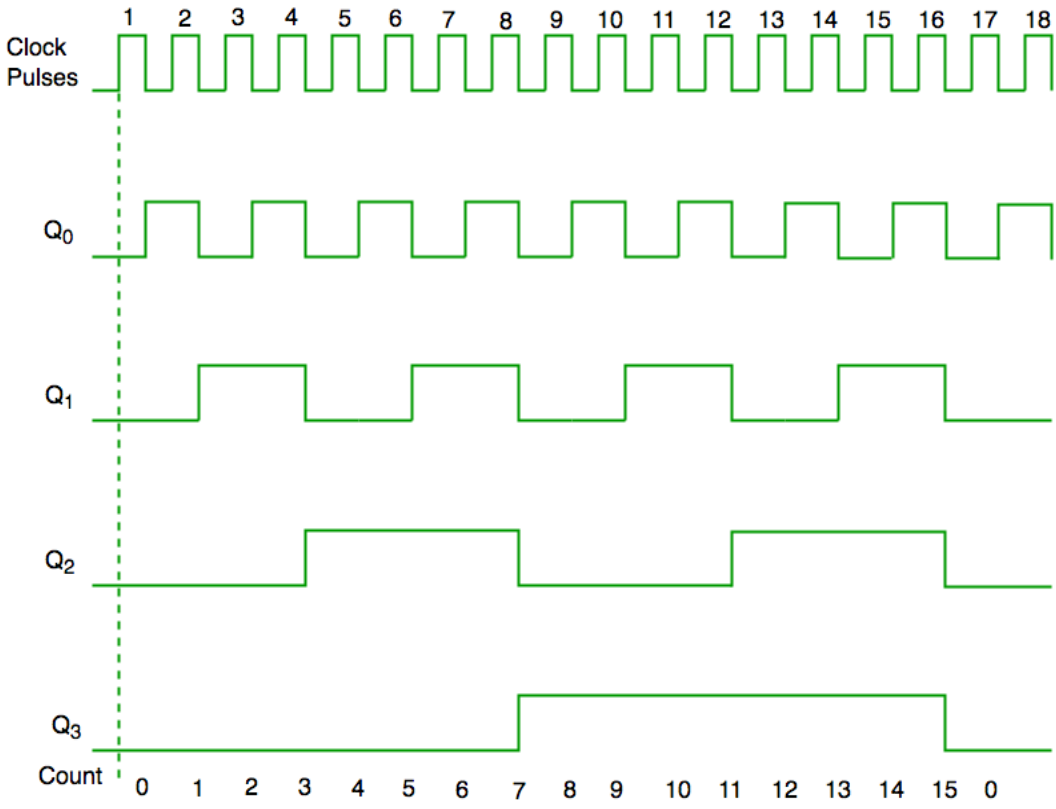
2. Synchronous Counter

Unlike the asynchronous counter, synchronous counter has one global clock which drives each flip flop so output changes in parallel. The one advantage of synchronous counter over asynchronous counter is, it can operate on higher frequency than asynchronous counter as it does not have cumulative delay

because of same clock is given to each flip flop. It is also called as parallel counter.



Synchronous counter circuit



Timing diagram synchronous counter

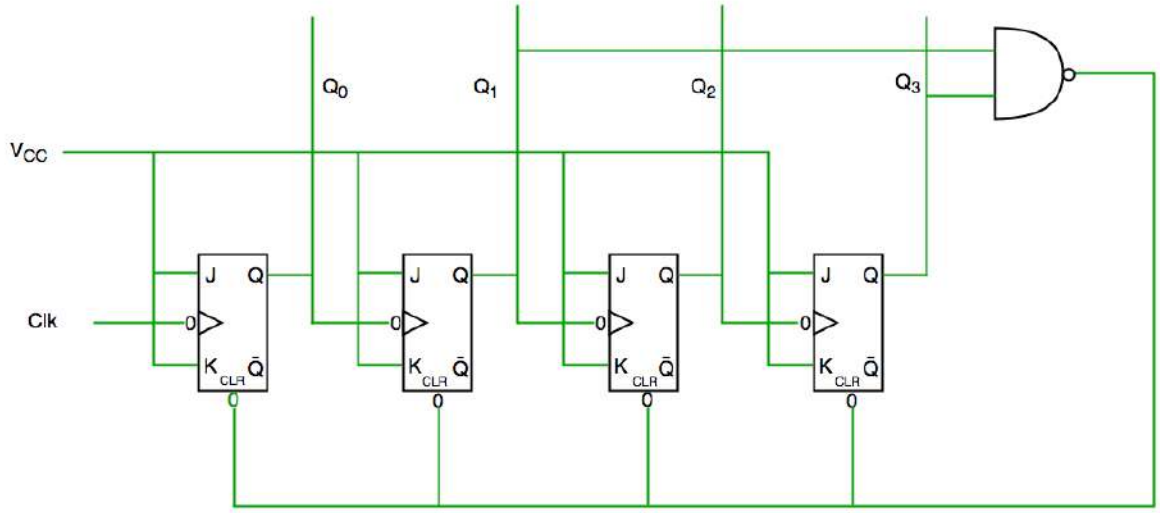
From circuit diagram we see that Q0 bit gives response to each falling edge of clock while Q1 is dependent on Q0, Q2 is dependent on Q1 and Q0 , Q3 is dependent on Q2,Q1 and Q0.

Decade Counter

A decade counter counts ten different states and then reset to its initial states. A simple decade counter will count from 0 to 9 but we can also make the decade counters which can go through any ten states between 0 to 15(for 4 bit counter).

Clock pulse	Q3	Q2	Q1	Q0
s0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	0	0	0	0

Truth table for simple decade counter



We see from circuit diagram that we have used nand gate for Q3 and Q1 and feeding this to clear input line because binary representation of 10 is—

1010

And we see Q3 and Q1 are 1 here, if we give NAND of these two bits to clear input then counter will be clear at 10 and again start from beginning.

Important point: Number of flip flops used in counter are always greater than equal to ($\mathbf{\log_2 n}$) where n=number of states in counter.

Modulus Counter (MOD-N Counter)

The 2-bit ripple counter is called as MOD-4 counter and 3-bit ripple counter is called as MOD-8 counter. So in general, an n-bit ripple counter is called as modulo-N counter. Where, MOD number = 2^n.

Type of modulus

- 2-bit up or down (MOD-4)
- 3-bit up or down (MOD-8)
- 4-bit up or down (MOD-16)

Application of counters

- Frequency counters
- Digital clock
- Time measurement
- A to D converter
- Frequency divider circuits
- Digital triangular wave generator.

Register

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

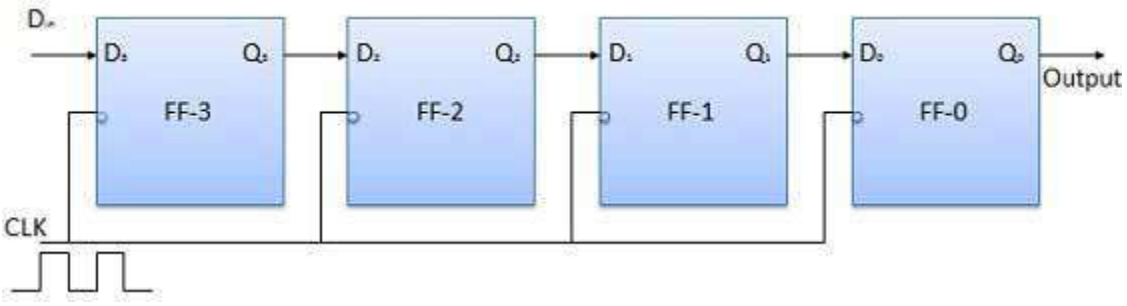
The binary data in a register can be moved within the register from one flip-flop to another. The registers that allow such data transfers are called as **shift registers**. There are four mode of operations of a shift register.

- Serial Input Serial Output
- Serial Input Parallel Output
- Parallel Input Serial Output
- Parallel Input Parallel Output

➢ Serial Input Serial Output

Let all the flip-flop be initially in the reset condition i.e. $Q_3 = Q_2 = Q_1 = Q_0 = 0$. If an entry of a four bit binary number 1 1 1 1 is made into the register, this number should be applied to $\mathbf{D_{in}}$ bit with the LSB bit applied first. The D input of FF-3 i.e. D_3 is connected to serial data input $\mathbf{D_{in}}$. Output of FF-3 i.e. Q_3 is connected to the input of the next flip-flop i.e. D_2 and so on.

Block Diagram



Truth Table

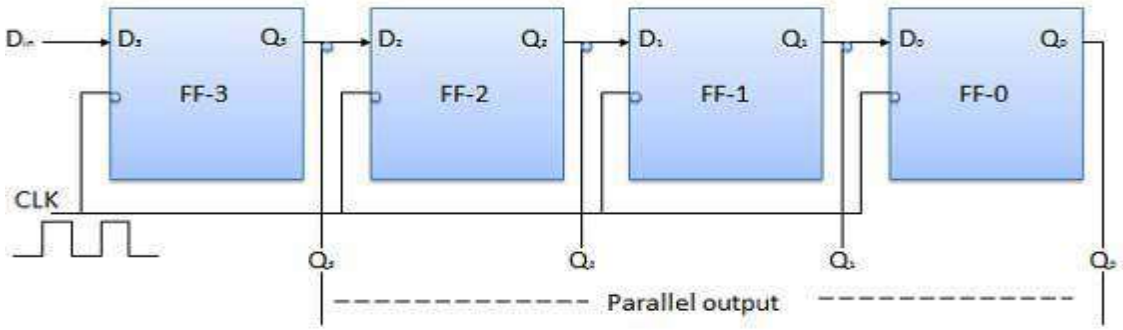
	CLK	$D_{in} = Q_3$	$Q_3 = D_2$	$Q_2 = D_1$	$Q_1 = D_0$	Q_0
Initially			0	0	0	0
(i)	↓	1	1	0	0	0
(ii)	↓	1	1	1	0	0
(iii)	↓	1	1	1	1	0
(iv)	↓	1	1	1	1	1

→ Direction of data travel

➢ Serial Input Parallel Output

- In such types of operations, the data is entered serially and taken out in parallel fashion.
- Data is loaded bit by bit. The outputs are disabled as long as the data is loading.
- As soon as the data loading gets completed, all the flip-flops contain their required data, the outputs are enabled so that all the loaded data is made available over all the output lines at the same time.
- 4 clock cycles are required to load a four bit word. Hence the speed of operation of SIPO mode is same as that of SISO mode.

Block Diagram



➢ Parallel Input Serial Output (PISO)

- Data bits are entered in parallel fashion.
- The circuit shown below is a four bit parallel input serial output register.
- Output of previous Flip Flop is connected to the input of the next one via a combinational circuit.
- The binary input word B_0, B_1, B_2, B_3 is applied though the same combinational circuit.
- There are two modes in which this circuit can work namely - shift mode or load mode.

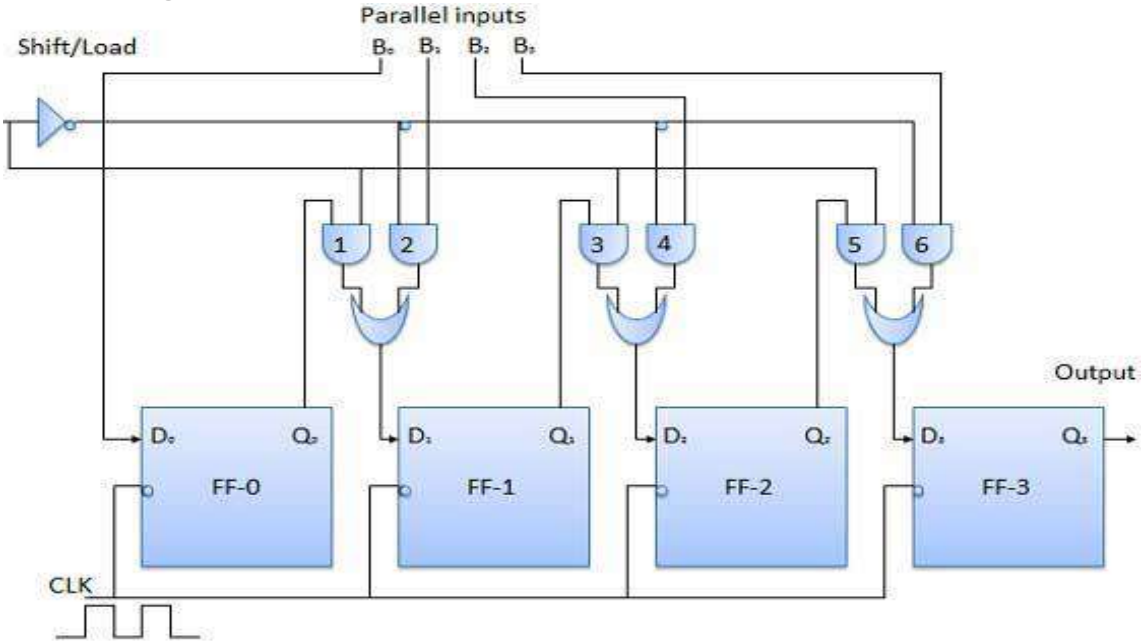
Load mode

When the shift/load bar line is low (0), the AND gate 2, 4 and 6 become active they will pass B_1, B_2, B_3 bits to the corresponding flip-flops. On the low going edge of clock, the binary input B_0, B_1, B_2, B_3 will get loaded into the corresponding flip-flops. Thus parallel loading takes place.

Shift mode

When the shift/load bar line is low (1), the AND gate 2, 4 and 6 become inactive. Hence the parallel loading of the data becomes impossible. But the AND gate 1,3 and 5 become active. Therefore the shifting of data from left to right bit by bit on application of clock pulses. Thus the parallel in serial out operation takes place.

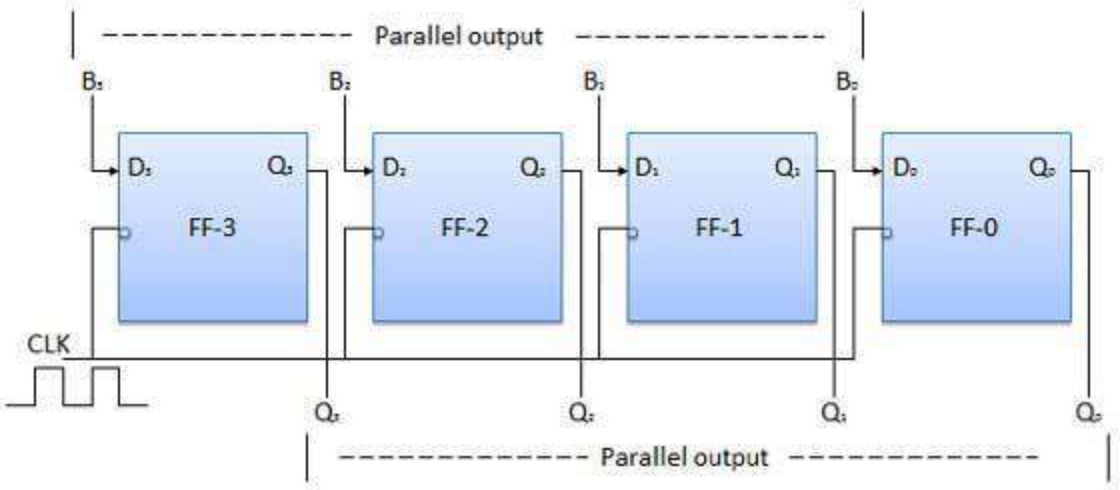
Block Diagram



- **Parallel Input Parallel Output (PIPO)**

In this mode, the 4 bit binary input B_0, B_1, B_2, B_3 is applied to the data inputs D_0, D_1, D_2, D_3 respectively of the four flip-flops. As soon as a negative clock edge is applied, the input binary bits will be loaded into the flip-flops simultaneously. The loaded bits will appear simultaneously to the output side. Only clock pulse is essential to load all the bits.

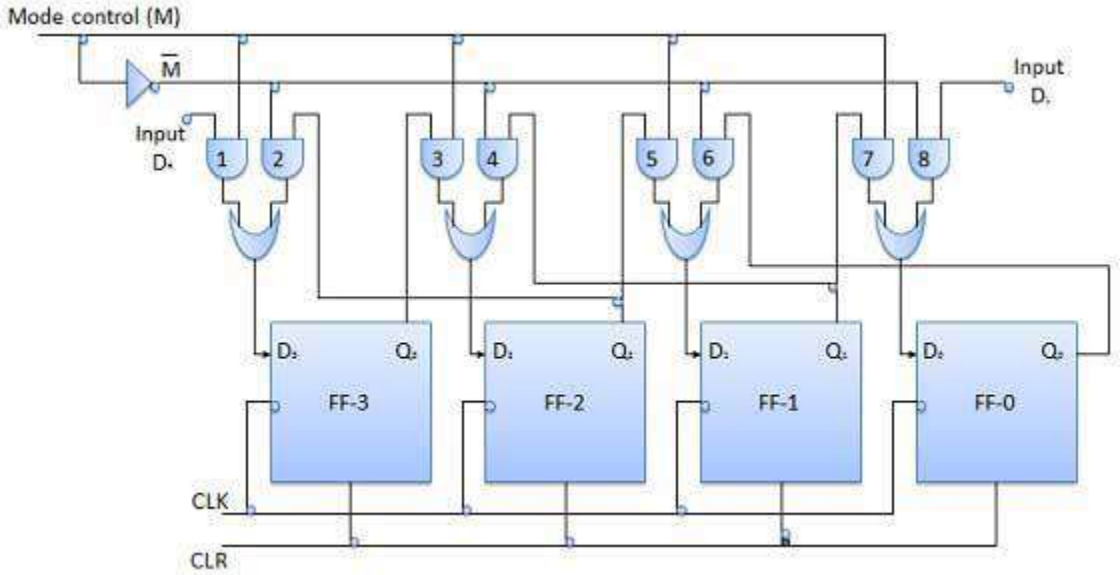
Block Diagram



- **Bidirectional Shift Register**

- If a binary number is shifted left by one position then it is equivalent to multiplying the original number by 2. Similarly if a binary number is shifted right by one position then it is equivalent to dividing the original number by 2.
- Hence if we want to use the shift register to multiply and divide the given binary number, then we should be able to move the data in either left or right direction.
- Such a register is called bi-directional register. A four bit bi-directional shift register is shown in fig.
- There are two serial inputs namely the serial right shift data input DR, and the serial left shift data input DL along with a mode select input (M).

Block Diagram



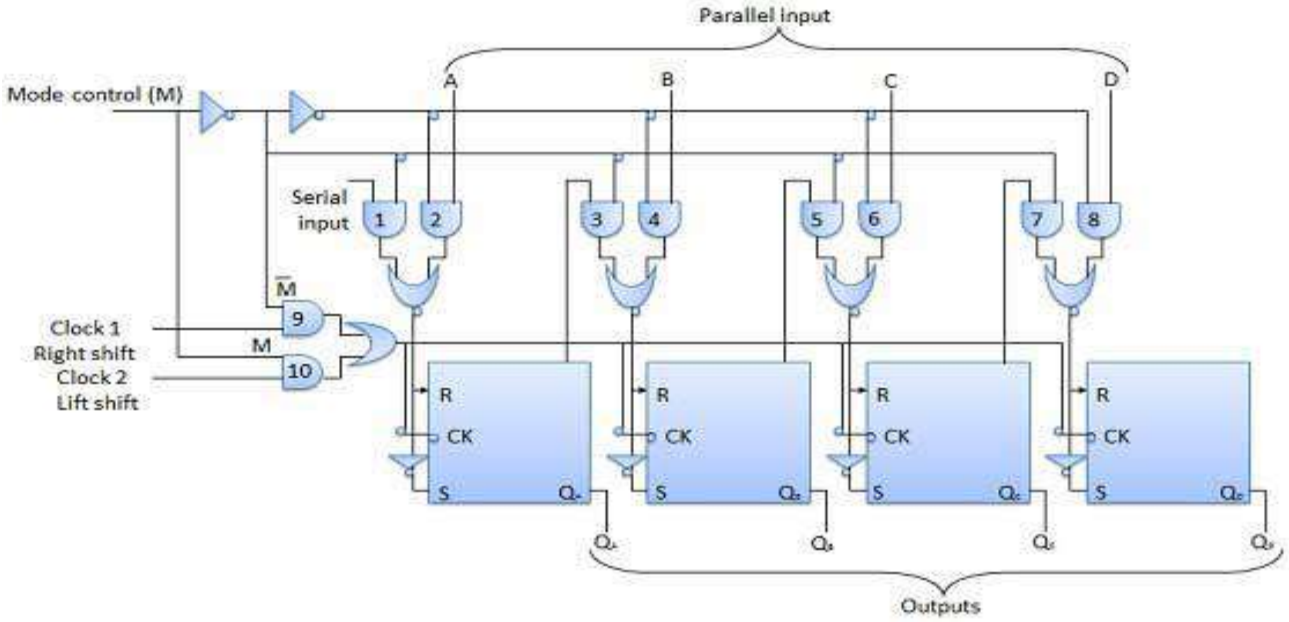
➢ Universal Shift Register

A shift register which can shift the data in only one direction is called a uni-directional shift register. A shift register which can shift the data in both directions is called a bi-directional shift register. Applying the same logic, a shift register which can shift the data in both directions as well as load it parallely, is known as a universal shift register. The shift register is capable of performing the following operation −

- Parallel loading
- Left Shifting
- Right shifting

The mode control input is connected to logic 1 for parallel loading operation whereas it is connected to 0 for serial shifting. With mode control pin connected to ground, the universal shift register acts as a bi-directional register. For serial left operation, the input is applied to the serial input which goes to AND gate-1 shown in figure. Whereas for the shift right operation, the serial input is applied to D input.

Block Diagram



Cascade 2x4Decoder to form 4x16