

COMPUTER ARITHMETIC AND DATA REPRESENTATION

DATA REPRESENTATION

- A digital computer uses the binary system, which has two digits: 0 and 1 and binary digit is called bit
- Information is represented in digital computers by groups of bits.
- A programmer uses decimal digits A,B...Z, a...z, special symbols,+, -,x etc for convenience.
- The decimal digits, letters, symbols etc. are converted to binary digits 0 and 1 within the computer.
- To understand operation of digital computers, it is necessary to have know number systems

DATA REPRESENTATION

- **The base or radix of a number system is defined as the number of different digits, which can occur in each position in the number system.**
- **Decimal Number System**
 - Decimal Number System has base or radix of 10 thus it uses power of 10 and includes the digits from 0 to 9.
- **Binary Number System**
 - The base of Binary number system is 2 as it has only 2 numbers 0 and 1.
- **Octal and Hexadecimal Number System**
 - The octal number system has base 8; the digits are 0,1,2,3,4,5,6, and 7.
 - Hexadecimal number system has base 16 0 to 9 and A, B, C, D, E and F.

BINARY ADDITION

- Adding Binary numbers is very similar to the longhand addition of decimal number
- Starts with adding the bits one column, or a place weight at a time, from right to left.
- The rules for binary addition are shown in table below:

	Sum	Carry
0+0	0	0
1+0	1	0
0+1	1	0
1+1	0	1
1+1+1	1	1

EXAMPLE

$$\begin{array}{r} 1 \\ 1 \ 0 \ 1 \ 0 \ 1 \\ + \underline{0 \ 0 \ 1 \ 1 \ 0} \\ \hline 1 \ 1 \ 0 \ 1 \ 1 \end{array}$$

Decimal

(21)

(6)

(27)

EXPLANATION

$$1+0=1$$

$$0+1=1$$

$$1+1=0 \text{ carry } 1$$

$$1+0+0=1$$

$$1+0=1$$

EXAMPLE

$$\begin{array}{r} (85)_{10} \\ + (181)_{10} \\ \hline (266)_{10} \end{array} \quad + \quad \begin{array}{r} 11111 \\ 11111 \\ 100001010 \end{array}$$

EXAMPLE

11 1 ← Carry bits → 11

1001101	1001001	1000111
+ 0010010	+ 0011001	+ 0010110
<hr/>	<hr/>	<hr/>
1011111	1100010	1011101

	1	1		1
	1	0	1	1
	.	0	1	
+		1	1	.
		0	1	1
	<hr/>			
	1	1	1	0
	.	1	0	1

BINARY ADDITION - EXERCISE

1) $1111 + 10001$

2) $10101 + 1111$

3) $11011 + 01010$

4) $10.10 + 11.11$

5) $10.100 + 101.0$

BINARY SUBTRACTION

- Subtracting Binary numbers is the inverse operation of addition and it is generally simpler than addition since only two numbers are involved and the upper value representation is greater than the lower value representation.
- The problem of borrow is similar to that in decimal subtraction.
- The rules for binary subtraction are shown in below table:

	DIFFERENCE	BORROW
0-0	0	0
0-1	1	1
1-0	1	0
1-1	0	0

EXAMPLE

$$\begin{array}{r} \\ \\ - \\ \hline \end{array}$$

0 1
1 1 ~~1~~ ~~0~~ 14
- 0 1 0 1 5
1 0 0 1 9

EXPLANATION

0-1=1 Borrow 1

0-0=0

1-1=0

1-1=0

BINARY SUBTRACTION-EXERCISE

1) 100100-1111

2) 101010-111

3) 111111-1000

4) 101000-101

5) 101111-1000

BINARY MULTIPLICATION

- The binary multiplication operation is actually a process of addition and shifting operation.
- This process has to be continued until all the multiplier is done, and finally, the addition operation is made.
- Similar to the decimal system, the multiplication of the binary numbers is done by multiplying the multiplicand with the multiplier.
- The multiplication by zero makes all the bits zero, and this step may be ignored in the intermediate steps.
- The multiplication by 1 makes all the multiplicand value unchanged.

BINARY MULTIPLICATION

- Binary multiplication is achieved in a similar fashion to multiplying decimal values.
- The rules for binary multiplication are as shown in table:

	MULTIPLICATION
0X0	0
0X1	0
1X0	0
1X1	1

EXAMPLE

$$\begin{array}{r} 1001 \\ \times 101 \\ \hline 1001 \\ 0000 \\ + 1001 \\ \hline 101101 \end{array}$$

Binary	Decimal
1001	9
101	5
Verification	
1001 * 101=101101	9*5=45

BINARY DIVISION

- N bit number is divided by m bit number resulting in quotient and remainder.
- In decimal number system, division by zero is meaningless same is applicable in binary number system.
- The rules for division are shown below

	Division
0/1	0
1/0	1

EXAMPLE

Binary Division

*Divide the binary number $A = 1010_2$
by $B = 10_2$*

$$\begin{array}{r} \underline{101} \\ 10 \overline{) 1010} \\ \underline{10} \\ 010 \\ \underline{10} \\ 0 \end{array}$$

Binary	Decimal
1010	10
10	2
Verification	
$1010 / 10 = 101$	$10/2=5$

BINARY DIVISION-EXAMPLE

1) 1001 /11

2) 10101/111

3) 1111/101

4) 111/10

5) 101010/110

SIGNED MAGNITUDE FORM

- Computers can represent integers as signed and unsigned quantities having both positive and negative values.
- In general we represent negative value as “-” on left side of a digit.
- Similarly we can start the left most bit in binary as the “sign bit”.
- When the sign bit is zero the number represented is positive and when the sign bit is one the number is negative.
- A 8-bit sign-magnitude number would appear as follows:

Sign bit

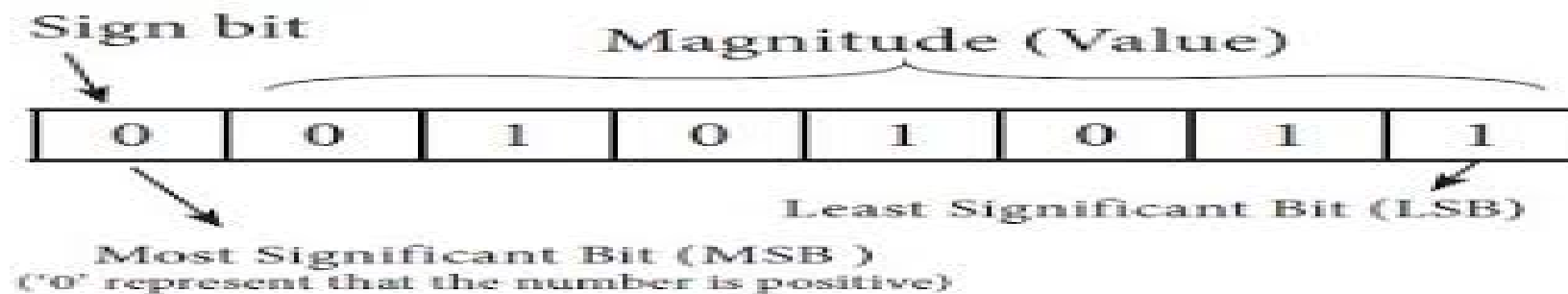
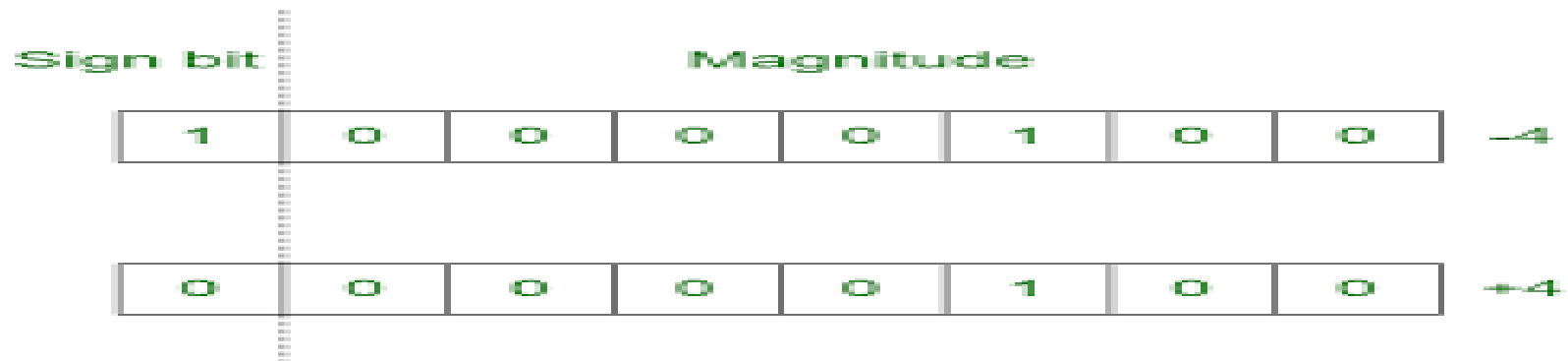
Binary Number or magnitude

7th bit

0 to 6th bit

SIGNED MAGNITUDE FORM

- A 8-bit sign-magnitude number would appear as follows:



SIGNED MAGNITUDE

- Eg:
- Represent the “-15 ” in 8-bit sign-magnitude:
 - So first of all convert 15 to binary number :
 - Thus $15_{10} = 1111_2$
- Now representing “-15 ” it in **8-bit Sign-magnitude**

8	4	2	1
1	1	1	1

Sign Bit	Binary Digits						
1	0	0	0	1	1	1	1

- Thus the answer for -15 is ***10001111***

SIGNED MAGNITUDE

- Eg:
- What are the decimal values of “**10000011**” 8-bit sign-magnitude numbers?

Sign Bit	Binary Number					
1	0	0	0	0	1	1

- Here the sign bit is 1 thus the sign is negative
- Further the decimal value of 0000011 would be 3 as :
- Thus the result is -3

8	4	2	1
0	0	1	1

SIGNED MAGNITUDE

Do as directed:

1) What are the decimal values of the following 8-bit sign-magnitude numbers?

a) $00000111 = ?$

b) $11111101 = ?$

c) $01111101 = ?$

2) Represent the following in 8-bit sign-magnitude:

a) $-48 = ?$

b) $+9 = ?$

c) $-11 = ?$

$(r-1)$'s COMPLEMENT (9's and 1's COMPLEMENT)

- The $r-1$'s complement consist 9's complement in the decimal number system and 1s complement in binary number system.
- 9's complement of a decimal number is the subtraction of it's each digits from 9.
- 1's complement of a binary number is another binary number obtained by toggling all bits in it, i.e., transforming the 0 bit to 1 and the 1 bit to 0.

9's COMPLEMENT

- The 9's complement is used to find the subtraction of the decimal numbers. The 9's complement of a number is calculated by subtracting each digit of the number by 9.
- **For example**, suppose we have a number 1423, and we want to find the 9's complement of the number. For this, we subtract each digit of the number 1423 by 9. So, the 9's complement of the number 1423 is **$9999-1423= 8576$** .
- For subtracting two numbers using 9's complement, we first have to find the 9's complement of the subtrahend and then we will add this complement value with the minuend. There are two possible cases when we subtract the numbers using 9's complement.

9's COMPLEMENT -IN DECIMAL

- **Step 1 : subtract each digit from 9**
- Eg: 9's complement of 57 = $9 - 5 = 4$ and $9 - 7 = 2$ i.e 42
- **For Sums:**
- **Step 1 : Subtract the second value's each digit from 9.**
- **Eg.** Find $55 - 23$ using 9's complement, $99 - 23 = 76$
- **Step 2 : Add the answer of 1st step to the 1st value.**
- Eg. $55 + 76 = 131$
- **Step 3: If Carry is obtained add that to the rest of the number**
- Eg $31 + 1 = 32$
- **Step 4: If carry is not obtained, subtract the digits of answer of step 2 from 9.**
- **Eg: Find $22 - 42$ using 9's complement,**
- **$99 - 42 = 57$, $57 + 22 = 79$, $99 - 79 = (-20)$**

9's COMPLEMENT- IN DECIMAL

1) 55-23

$$\begin{array}{r} 99 \\ - 23 \\ \hline 76 \end{array}$$

$$\begin{array}{r} 55 \\ + 76 \\ \hline \end{array}$$

$$\begin{array}{r} 131 \\ \bullet \swarrow 1 \\ + \\ \hline 32 \end{array} \quad \text{(the carry value 1 gets added to the rest of the digits)}$$

2) 22 - 42

$$\begin{array}{r} 99 \\ - 42 \\ \hline 57 \end{array}$$

$$\begin{array}{r} 22 \\ + 57 \\ \hline \end{array}$$

79 (no carry so finding 9's complement of this answer)

$$\begin{array}{r} 99 \\ - 79 \\ \hline \end{array}$$

- 20 (adding negative sign as there was no carry)

1's COMPLEMENT - IN BINARY

- The 1's complement in the binary number system is similar to 9's complement in decimal number system.
- To obtain 1s complement of binary number each bit of the binary number is subtracted from 1 or simply changing each 1 in the number to a 0 and each 0 in the number to a 1.

	Magnitude	1's Complement
1	1100100	0011011
2	0110	1001
3	00000011	11111100
4	110011001	001100110

Decimal Number	1's Complement
+15	0 111
-15	1 111

1's COMPLEMENT – IN BINARY

- When Subtraction is performed in the 1's complement system, if any carry is generated it is added to the least significant bit.
- **Perform the following subtraction of binary numbers using 1's complement.**

1) 11000 – 10000

Firstly, finding 1's Complement of 10000 -> 01111

Now adding it to 1's value

$$\begin{array}{r} 1\ 1\ 0\ 0\ 0 \\ +\ 0\ 1\ 1\ 1\ 1 \\ \hline 1\ 0\ 0\ 1\ 1\ 1\ (\text{carry is generated to adding it to LSB}) \\ +\ 1 \\ \hline 0\ 1\ 0\ 0\ 0\ 0 \end{array}$$

1's COMPLEMENT-IN BINARY

Perform the following subtraction of binary numbers using 1's complement.

2) 1001 – 1111

- 1's complement of 1111 is 0000.
- Adding it to 1st value

$$\begin{array}{r} 1\ 0\ 0\ 1 \\ +\ 0\ 0\ 0\ 0 \\ \hline \end{array}$$

1 0 0 1 (No carry generated so finiding 1's complement of 1001)

- 1's complement of 1001 is 1 0110 (As no carry answer is negative)

(r)'s COMPLEMENT

(10's and 2's COMPLEMENTS)

- The r's complement consists 10's complement in decimal number system and 2's complement in binary number system.
- **10's complement** of a decimal number can be found by adding 1 to the 9's **complement** of that decimal number.
- To get 2's complement of binary number is 1's complement of given number plus 1 to the least significant bit (LSB). For example 2's complement of binary number 10010 is $(01101) + 1 = 01110$.

10's COMPLEMENT –IN DECIMAL

- **10's complement can be performed by subtracting each digit of the number from 9 and then adding 1 to LSD of the given number.**
 - **Eg:** 10's complement of 54 is $99 - 54 = 45 + 01 = 46$
 - 125 is $999 - 125 = 874 + 001 = 875$.
- **To subtract a positive number from another :**
 - First the 10's complement of the subtrahend is formed, and is added to the minuend.
 - If there is a carry from the addition of the most significant digits, then it is discarded, the result is positive.
 - If there is no carry the difference is negative, the 10's complement of this number is formed again and a minus is placed before the result.

10'S COMPLEMENT –IN DECIMAL

- Performs the following subtraction of decimal numbers using 10's complement

1) $55 - 23$

$$\begin{array}{r} 99 \\ - 23 \\ \hline \end{array}$$
$$\begin{array}{r} 76 \\ + 1 \\ \hline \end{array}$$
$$\begin{array}{r} 77 \\ \hline \end{array}$$
$$\begin{array}{r} 55 \\ + 77 \\ \hline \end{array}$$

1 3 2 (Carry is dropped answer is positive)

2) $22 - 42$

$$\begin{array}{r} 99 \\ - 42 \\ \hline \end{array}$$

$57 + 1 = 58$

$$\begin{array}{r} 22 \\ + 58 \\ \hline \end{array}$$

80 (No carry answer is negative apply 10's complement on result)

$99 - 80 = 19 + 1 = -20$

2's COMPLEMENT –IN BINARY

- The 2's complement in the binary number system is similar to 10's complement in decimal number system.
- To obtain 2's complement of binary number
 - Perform 1's Complement
 - Add 1 to its least significant bit.
- For Example: 2's Complement of 1001
 - 0110 (1's Complement: Replacing 0 by 1 and 1 by 0. So 1001 is 0110)
 - $0110 + 1 = 0111$ (2's Complement: Add 1 to least significant bit. $0110 + 1$ is 0111)

2's COMPLEMENT –IN BINARY

- Find 2's Complement of following binary numbers

	Magnitude	1's Complement	2's Complement
1	01101	10010	10011
2	000001	111110	111111
3	110011	001100	001101

2's COMPLEMENT-IN BINARY

- Perform the subtraction of binary numbers using 2's Complement

1) 11000-10000 (Decimal: 24-16=8)

- Find 2's Complement of 10000

- 01111 (1's complement of 10000)

- 01111+1 =10000 (2's complement, by adding 1 to 01111)

- 2's complement of 10000 is 10000

- Now add 11000 to 10000

1 1 0 0 0

+ 1 0 0 0 0

1 0 1 0 0 0 (Carry dropped)

- Final answer is 01000

2's COMPLEMENT –IN BINARY

- Perform the subtraction of binary numbers using 2's Complement

2) 1001-1111 (Decimal: 9-15=-6)

- Find 2's Complement of 1111
 - 1111 (1's complement of 0000)
 - 0000+1 =0001 (2's complement, by adding 1 to 0000)
- 2's complement of 1111 is 0001
- Now add 1001 to 0001

$$\begin{array}{r} 1\ 0\ 0\ 1 \\ +\ 0\ 0\ 0\ 1 \\ \hline \end{array}$$

1 0 1 0 (No carry, so again 2's complement)

2's COMPLEMENT –IN BINARY (Cont..)

- Find 2's Complement of 1010
 - 0101 (1's complement)
 - $0101 + 1 = 0110$ (2's Complement)
- As no carry was generated, so
 - -0110
- Final answer is : -0110

FIXED POINT REPRESENTATION

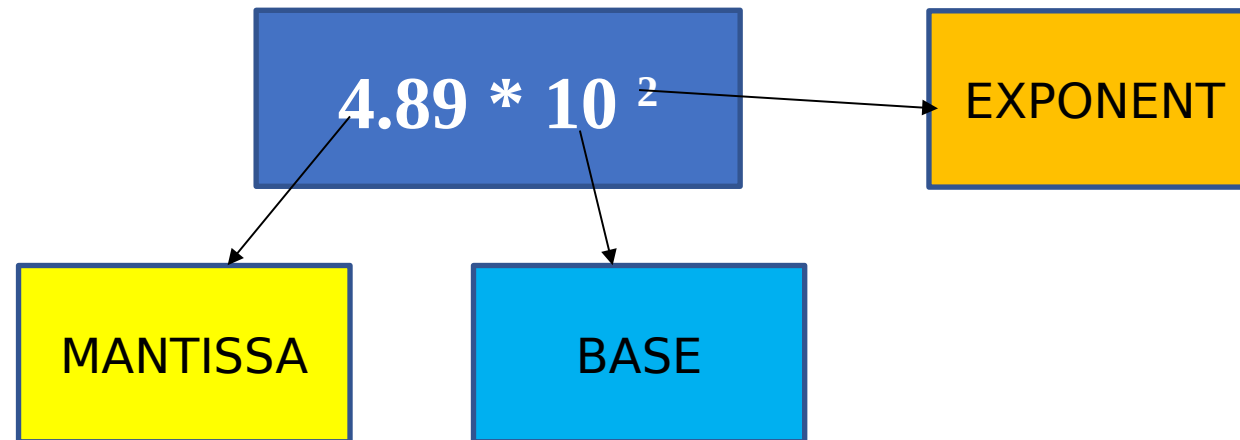
- In **fixed-point** of number representation all numbers are represented as **integers** or **fractions**.
- **Fixed point** numbers may be **signed** or **unsigned numbers**.
- Inside computer, number always has sign.
- 1 bit reserved for sign is called sign bit.
- In a number 56.897, the number to left side of decimal point (radix point) is integer part and the number to right side of decimal point is fraction part.

FLOATING POINT NUMBERS

- A number which has both an **integer part** and **fractional part** is called **real number**.
- A **floating point** number may be either **positive or negative**.
- **For example:** 43.78, -879.76 etc.
- **Examples** of real binary numbers are 11.1100, 0.101, -0.11 etc.
- Floating point number representation has following three parts
 - Mantissa
 - Base
 - Exponent

FLOATING POINT NUMBERS

- The structure of a **floating point**(real) number is as follows:



- The **mantissa** and **exponent** are **explicitly** represented in the computer but the **base** is implied (suggested but not directly expressed) for a given computer.

FLOATING POINT NUMBERS

- Following examples illustrates floating-point number system:

NUMBER	MANTISSA	BASE	EXPONENT
32×10^7	32	10	7
7652.123	0.7652123	10	4
111×2^6	111	2	6
22.11	0.2211	10	2

FLOATING POINT NUMBERS

- In general form, number N may be written as $N = M \times R^e$, where M is mantissa, e is the exponent and R is the radix or base of the number.
- Floating point number format:

Sign	Exponent	Mantissa
------	----------	----------

- IEEE standard is now widely followed for floating-point representation.

FLOATING POINT NUMBERS

- It has two formats for single precision and double precision:
- Single precision floating point number format (32 bit)

Sign	Exponent, 8 bits	Mantissa, 23 bits
-------------	-------------------------	--------------------------

Single precision floating point number format (32 bit)

- Double precision floating point number format (64 bit)

Sign	Exponent, 11 bits	Mantissa, 52 bits
-------------	--------------------------	--------------------------

Double precision floating point number format (64 bit)

FLOATING POINT ADDITION

- **Floating Point Addition:**
- Consider two floating point operands X and Y expressed as follows.
- $X = M_x \times r^{E_x}$ and $Y = M_y \times r^{E_y}$ where M_x and M_y are mantissa of X and Y , r is the radix and E_x, E_y are exponents of X and Y .

FLOATING POINT ADDITION

Example #1: $2.02631123 \times 10^{14} + 1.12651113 \times 10^{12}$

Step 1: $r^{Ex} - r^{Ey} = 14 - 12 = 2$

Step 2: Y is smaller component so Mantissa of Y has to be shifted right 2 times. $M_y = 0.0112651113$

Step 3: $M_x + M_y = 2.02631123$

+0.0112651113

$2.0375763413 \times 10^{14}$

Step 4: Result = $2.0375763413 \times 10^{14}$

Step 5: Normalizing the result $2.0375763413 \times 10^{14}$ (Already normalized)

FLOATING POINT SUBTRACTION

- **Floating Point Subtraction:**
- Subtraction algorithm is similar to the addition algorithm
- The subtraction of mantissa $M_x - M_y$ is done in place of addition and then normalize the result.

FLOATING POINT SUBTRACTION

Example #1: $3.25 \times 10^3 - 2.13 \times 10^{-1}$

Step 1: $r_{Ex} - r_{Ey} = 3 - (-1) = 4$

Step 2: Y is smaller component so Mantissa of Y has to be shifted right 4 times. $M_y = 0.000213 \times 10^3$

Step 3: $M_x - M_y = 3.250000 \times 10^3$
 -0.000213×10^3

3.249787×10^3

Step 4: Result = 3.249787×10^3

Step 5: Normalizing the result = 3.249787×10^3 (Already Normalized)

FLOATING POINT MULTIPLICATION

- **Floating Point Multiplication:** $(X=M_x \times 10^{E_x}) \times (Y=M_y \times 10^{E_y}) = M_x \times M_y \times 10^{E_x+E_y}$
- **Example #1:** $110 \times 10^{10} \times 9.200 \times 10^{-5}$

Step 1: Add the exponent = $10 + (-5) = 5$.

Step 2: Multiply the mantissa: $110 \times 9.200 = 1012$

Step 3: 1012×10^5

Step 4: Normalize the result: 1.012×10^8

FLOATING POINT DIVISION

- **Floating Point Division:** $(X=M_x/M_y)Xr^{E_x-E_y}$
- **Example #1:** $6.20 \times 10^{10}/3.123 \times 10^4$
- Step 1: Subtract the exponents = $10-4 = 6$
- Step 3: Divide the mantissa $6.20/3.123=1.98527057 = 1.98527057 \times 10^6$
- Step 4: Normalize the result = 1.98527057×10^6

IEEE REPRESENTATION

- The IEEE 754 has produced a standard for floating point arithmetic.
- This standard specifies how single precision (32 bit) and double precision (64 bit) floating point numbers are to be represented, and how arithmetic should be carried out on them.
- Following table shows layout for single (32-bit) and double (64-bit) precision floating point values.

	Sign	Exponent	Fraction	Bias
Single Precision	1[31]	8[30-23]	23[22-00]	127
Double Precision	1[63]	11[62-52]	52[51-00]	1023

IEEE REPRESENTATION

- **Sign Bit:**

- 0 denotes positive number, 1 denotes negative number.
- Flipping the value of this bits flips the sign of the number.

- **The Exponent:**

- Exponent field needs to represent both positive and negative exponents.
- To do this, bias is added to the actual exponent in order to get the stored exponent.
- For IEEE single-precision floats, this value is 127.
- Exponent of 0 means that 127 is stored in exponent field.
- Stored value of 200 indicates exponent of $(200-127)$ or 73.
- For double precision, exponent field is 11 bits, and has a bias 1023

IEEE REPRESENTATION

- **The Mantissa**

- Mantissa is also known as significand, it represents precision bits of the number.
- It is composed of implicit leading bit and the fraction bits.
- To find out value of implicit leading bit, consider that any number can be expressed in scientific notation in many different ways.
- For example:
 - 5.00×10^0
 - 0.05×10^2
 - 5000×10^{-3}

IEEE REPRESENTATION

- To maximize the quantity of representable numbers, floating point numbers are typically stored in normalized form.
- This can be done by putting radix point after the first non-zero digit.
- For Example: In normalized form,
 - 5 is represented as 5.0×10^0
 - 89.67 is represented as 8.967×10^1
 - 54896.45 is represented as 5.489645×10^4

IEEE REPRESENTATION

- IEEE floating point standard also specifies some bit patterns for following special cases:
- An exact value of 0 is represented by 0 for the exponent and 0 for the mantissa.
- An exponent of 255 and a 0 for mantissa is used to represent infinity.
- An exponent and a mantissa not equal to 0 is used to represent exception conditions which occur such as when one attempts to divide 0 by 0 or subtract infinity from infinity . This is called NaN (not a number)

IEEE REPRESENTATION

- Steps to convert a number to floating point standard format

Step 1: First convert the given number to binary

Step 2: Normalized the number so that there is one nonzero digit to the left of the binary place, adjusting the exponent as necessary.

Step 3: The digits to the right of the binary number are then stored as mantissa starting with the MSB of the mantissa field. Because all numbers are normalized, there is no need to store the leading 1.

Note: Because the leading 1 dropped, it is no longer proper to refer to the stored value as the mantissa. In IEEE terms, this mantissa minus its leading digit is called the significant.

IEEE REPRESENTATION

- Steps to convert a number to floating point standard format.

Step 4: Add 127 to the exponent and convert the resulting sum to binary for the stored exponent value. For double precision add 1023 to the exponent. Be sure to include all 8 or 11 bits of exponent.

Step 5: The sign bit is a one for negative numbers and 0 for positive numbers.

IEEE REPRESENTATION

- Example #1: Convert following binary numbers in the single precision floating point standard.

1) 3.5

Step 1: Convert number to binary

$$(11.1)_2$$

Step 2: Normalized

$$1.11 \times 2^1$$

IEEE REPRESENTATION

Step 3: Identify Sign and significand.

sign=0, significand=1100

Step 4: Add 127 to Exponent

$127+1=128=10000000$

Step 5: FPS number (3.5)

0 10000000 11000000000000000000000000000000

Step 6: 0 x 40600000H (H=Hexadecimal)

IEEE REPRESENTATION

- Example #2: Convert following binary numbers in the single precision floating point standard.

1) 100

Step 1: Convert number to binary

$$(1100100)_2$$

Step 2: Normalized

$$1.100100 \times 2^6$$

IEEE REPRESENTATION

Step 3: Identify Sign and significand.

sign=0, significand=100100

Step 4: Add 127 to Exponent

$127 + 6 = 133 = 10000101$

Step 5: FPS number (100)

0 10000101 100100000000000000000000

Step 6: 0 x 42C80000H (H=Hexadecimal)

IEEE REPRESENTATION

- Example #3: Convert following FPS numbers to decimal number.

1) 0X C2508000H

Step 1: Convert C2508000H to binary number

11000010010100001000000000000000 (binary)

Step 2: Identify Sign

1

Step 3: Identify Exponent

10000100

IEEE REPRESENTATION

Step 4: Identify Significand

101000010000000000000000

Step 5: Decimal value of exponent 10000100 is 132

Step 6: Exponent – Bias i.e. $132 - 127 = 5$

Step 7: $-1.10100001 * 2^5$

Step 8: -110100.001

Step 9: -52.125

GATES

- A logic gate is an elementary building block of a digital circuit.
- Most logic gates have two inputs and one output.
- At any given moment, every terminal is in one of the two binary conditions: low(0) or high(1), represented by different voltage levels.
- The logic state of a terminal can, and generally does, change often, as the circuit processes data.
- In most logic gates, the low state is approximately 0 volts (0 V), while the high state is approximately five volts positive (+5 V).

GATES

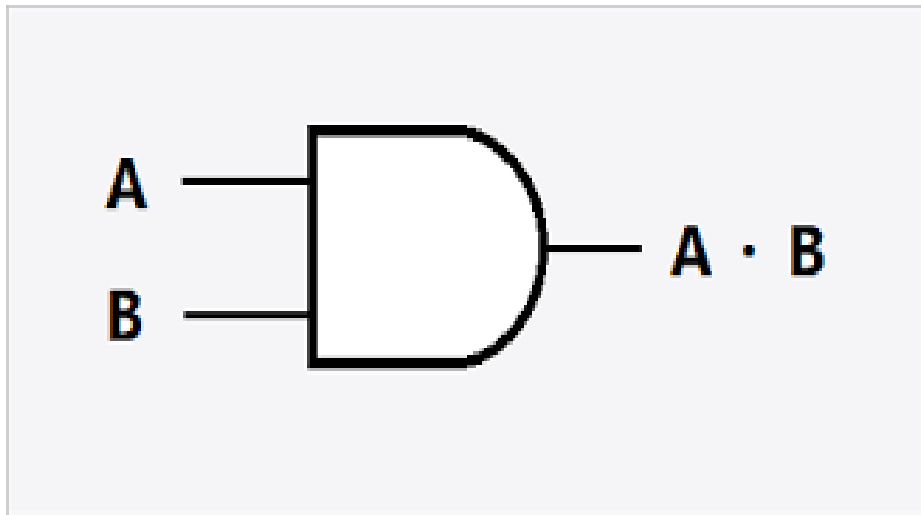
- There are 7 basic logic gates:

1. AND
2. OR
3. XOR
4. NOT
5. NAND
6. NOR
7. XNOR

GATES- AND Gate

- **AND Gate:**
- An AND gate can have two or more inputs, its output is true if all inputs are true.
- The output C is true if input A and inputs B are both true.
- It can be written as:
- $C = A \text{ AND } B$ or $C = A.B$

GATES- AND Gate




$$C = A \cdot B$$

AND GATE

INPUT A	INPUT B	OUTPUT C
0	0	0
0	1	0
1	0	0
1	1	1

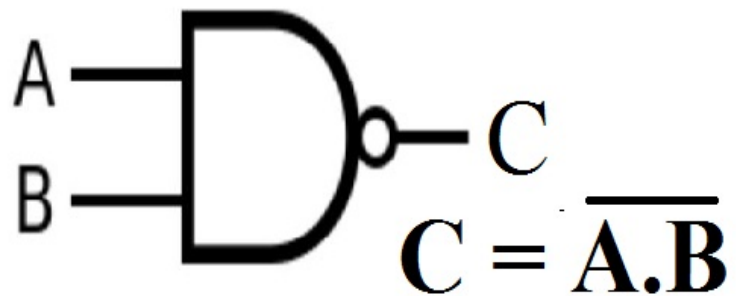
**TRUTH
TABLE**

GATES- NAND Gate

- **NAND Gate (NAND=Not AND):**
- A NAND gate can have two or more inputs; its output is true if NOT all inputs are true.
- In following figure, an AND gate with the output inverted, as shown by the 'C' in the output.

- The output is true if input A and input B are NOT both true:
- $C = \text{NOT } (A \text{ AND } B)$ or $C = A.B$

GATES- NAND Gate

NAND GATE



NAND GATE

ProjectIoT123.com

Truth Table

INPUT		OUTPUT
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

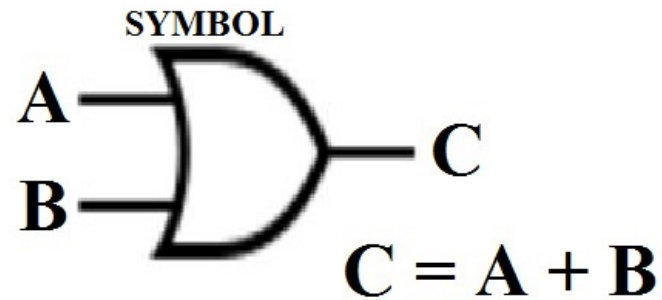
Truth Table

GATES-OR Gate

- **OR Gate**
- An OR gate can have two or more inputs, its output is true if at least one input is true.
- The output Q is true if input A OR input B is true (or both of them are true):
- $C = A \text{ OR } B$ or $C = A + B$

GATES-OR Gate

OR Gate



OR gate

TRUTH TABLE

INPUT		OUTPUT
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

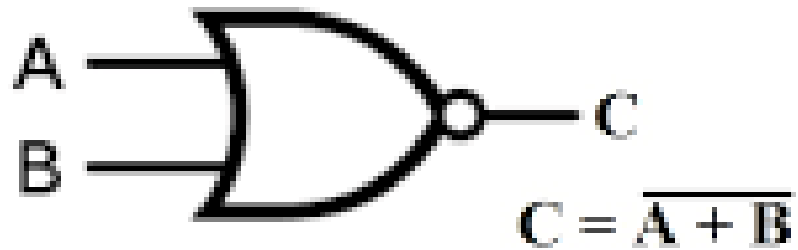
Truth Table ProjectIoT123.com

GATES-NOR Gate

- **NOR GATE (Nor=Not OR):**
- A NOR gate can have two or more inputs; its output is true if no inputs are true.
- In following figure an OR gate with the output inverted, as shown by the 'C' on the output.
- The output C is true if NOT $\overline{A \text{ OR } B}$ are true:
- $C = \text{NOT} (A \text{ OR } B)$ or $Q = A + B$

GATES- NAND Gate

NOR GATE



TRUTH TABLE

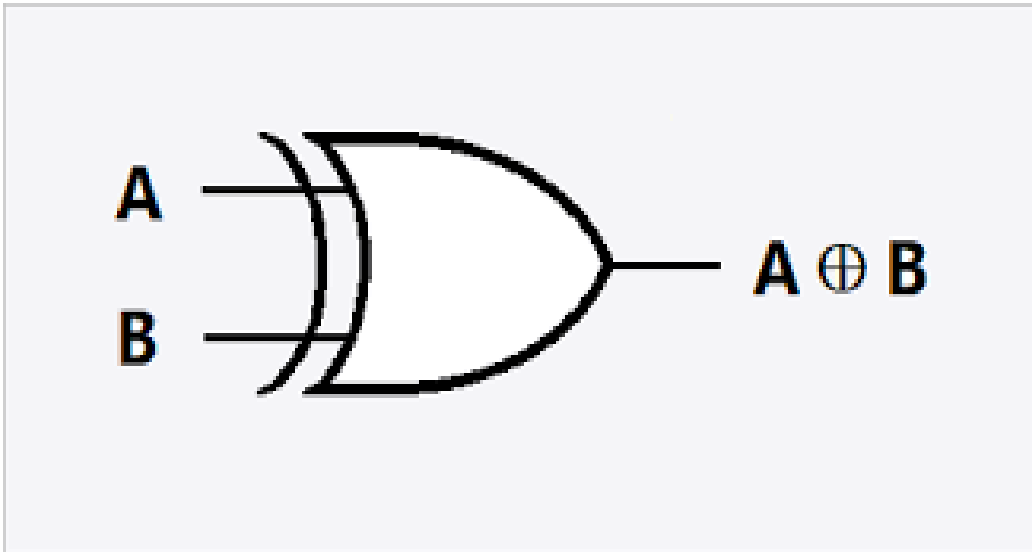
INPUT		OUTPUT
A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

Truth Table

GATES-EX-OR Gate

- **EX-OR(Exclusive –OR) Gate**
- EX-OR gates can only have 2 inputs.
- This is like an OR gate but excluding both inputs being true.
- The output C is true if either input A is true OR input B is true, but not when both of them are true:
- $Q = (A \text{ AND NOT } B) \text{ OR } (B \text{ AND NOT } A)$ or $A \oplus B$ or $A\bar{B} + \bar{A}B$

GATES-EXOR Gate



EXOR gate

INPUT A	INPUT B	OUTPUT C
0	0	0
0	1	1
1	0	1
1	1	0

Truth Table

GATES-EX-NOR Gate

- **EX-NOR(Exclusive –NOR) Gate**
- EX-NOR gates can only have 2 inputs.
- In figure below an EX-OR gate with the output inverted, as shown by the 'C' on the output.
- The output Q is true if inputs A and B are the SAME (either true or both false): $Q = (A \text{ AND } B) \text{ OR } (\text{NOT } A \text{ AND } \text{NOT } B)$ or $\overline{A} \oplus B$ or $AB + \overline{A}\overline{B}$

GATES-EX-NOR Gate



EX-NOR gate

Inputs		Output
A	B	$Y = \overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

Truth Table

GATES-Universal gates

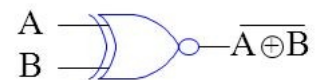
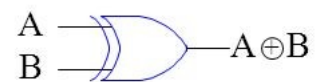
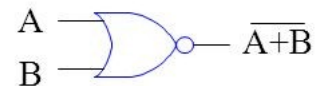
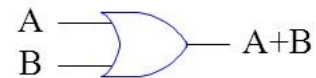
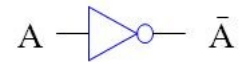
- **NAND and NOR Gate as Universal Gate**
- All other gates/functions can be implemented by NOR or NAND gates and implementing with NAND is easier when considering power and area of the chip.
- So they are called universal gates.
- NAND gate equivalent of NOT, AND, OR and NOR gates

GATES- Universal gates

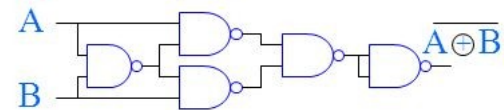
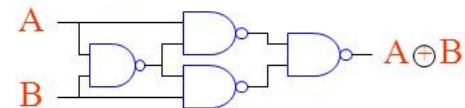
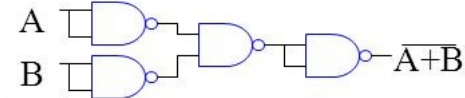
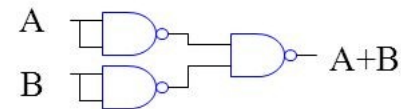
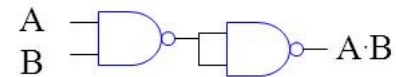
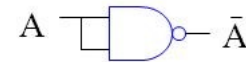


The NAND gate as a universal gate

Logic function



NAND gate only



FLIP FLOP

- **FLIP FLOP:**
- Flip flop is two-state/ bi-stable device which offer basic memory cell for sequential logic operations.
- Flip-flops are used for digital data storage and transfer.

FLIP FLOP

- **CHARACTERISTICS OF FLIP FLOP:**

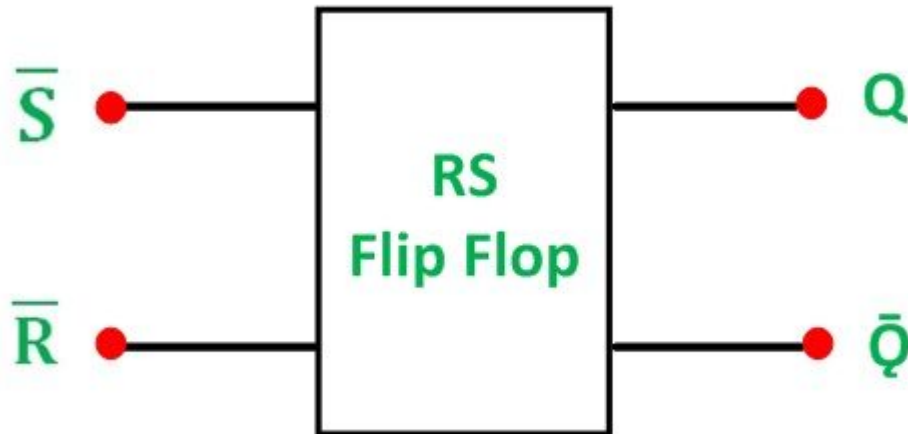
The flip-flop has two output signals, one of which is the complement of the other.

- There are basic four types of flip-flops:

1. RS flip-flop
2. JK flip-flop
3. D flip-flop
4. T flip-flop

RS FLIP FLOP

- **RS FLIP-FLOP:**
- Block Diagram of the RS Flip-Flop:

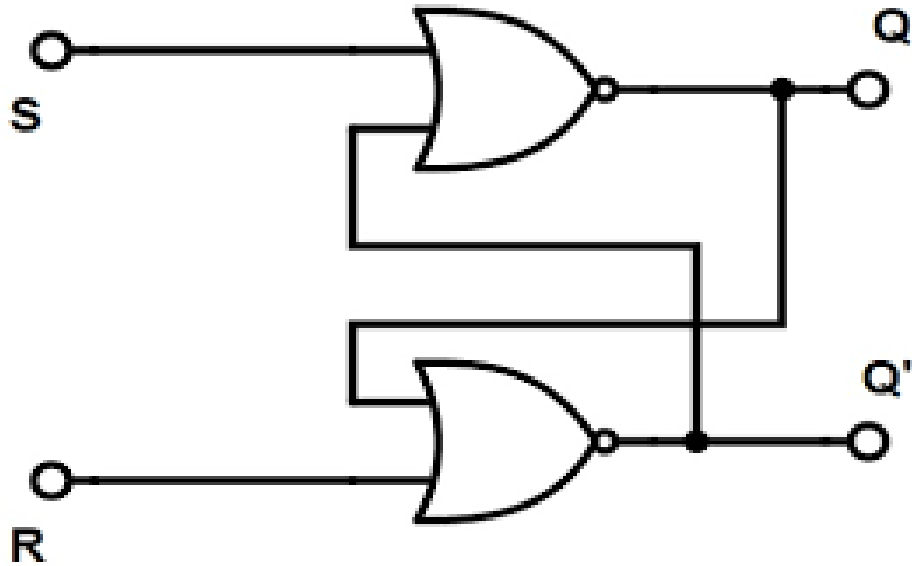


RS FLIP FLOP

- **RS FLIP-FLOP:**
- RS flip-flop is the simplest possible memory element.
- RS flipflop stands for reset-Set flip Flop.

RS FLIP FLOP

- **RS FLIP-FLOP COMPOSED OF TWO NOR GATES:**



RS FLIP FLOP

- **TRUTH TABLE:**

S	R	Q	Comment
0	0	Q	Hold State
1	0	1	Set
0	1	0	Reset
1	1	Indeterminate	Avoid

JK FLIP FLOP

- **JK FLIP-FLOP:**
- Block Diagram of JK Flip-Flop:



JK FLIP FLOP

- **JK FLIP-FLOP:**
- JK (Jack-Kilby) Flip-flop is the most versatile flip-flop, and the most commonly used flip-flop when discrete devices are used to implement arbitrary state machines.
- Like RS it has two data inputs, J and K, and a clock input.
- It has no undefined states.

JK FLIP FLOP

- **TRUTH TABLE**

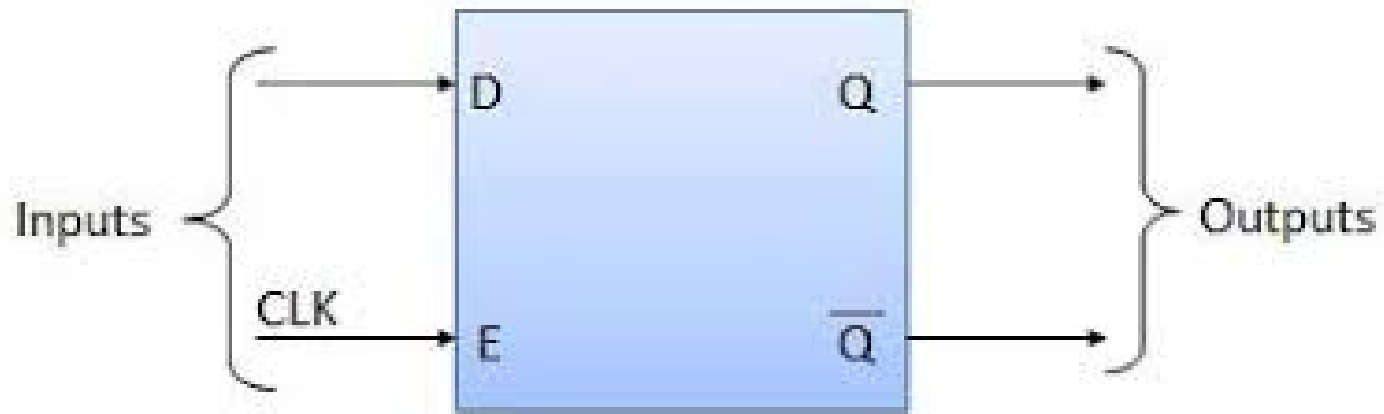
J	K	CLK	Q
0	0	↑	Q_0 (No change)
1	0	↑	1
0	1	↑	0
1	1	↑	Q' (toggles)

D FLIP FLOP

- **D FLIP FLOP:**
- An RS flip-flop is rarely used in actual sequential logic.
- It is fundamental building block for the very useful D flip-flop.
- The D flipflop has only single data input.

D FLIP FLOP

- **BLOCK DIAGRAM OF D FLIP FLOP :**



D FLIP FLOP

- **TRUTH TABLE:**

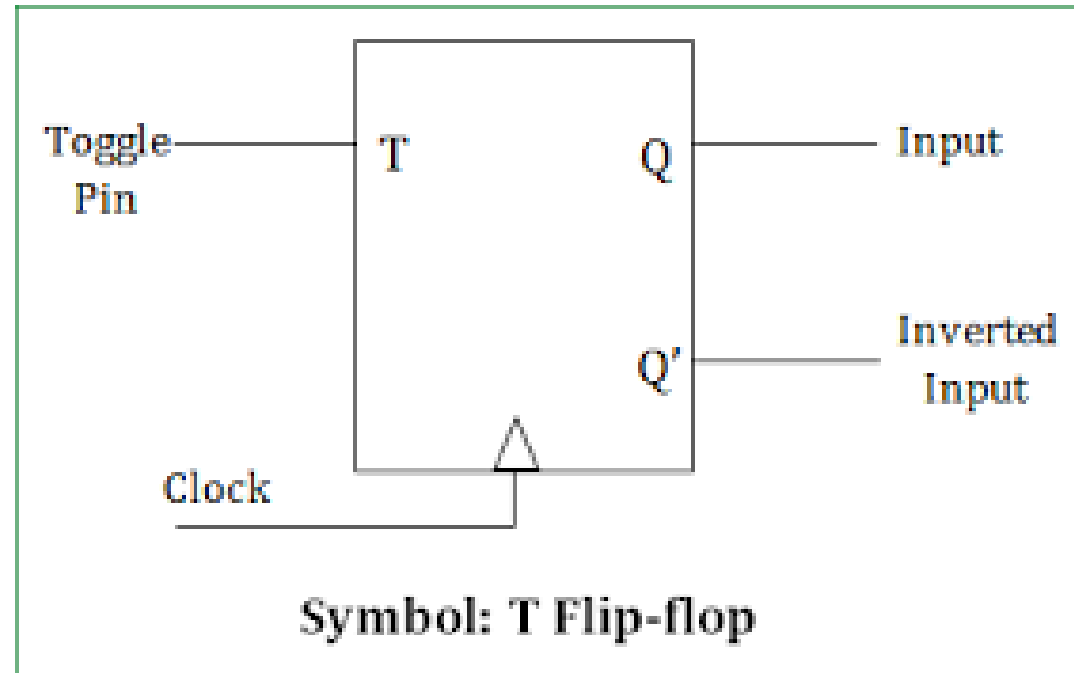
CLK	D	Q _{n+1}
↓	X	Q _n
↑	0	0
↑	1	1
↓	X	Q _n

T FLIP FLOP

- **T FLIP FLOP:**
- **T STANDS FOR TOGGLE**
- This type of flip-flop is simplified version of the JK Flip-flop.

T FLIP FLOP

- **BLOCK DIAGRAM OF T FLIP-FLOP:**



T FLIP FLOP

- **TRUTH TABLE:**

T	Q_{n+1}
0	Q_n
1	Q'_n