Computer Organization and Architecture

- **Computer architecture** refers to those attributes of a system visible to a programmer.
- Those attributes have direct impact on logical execution of a program.
- **Computer organization** refers to the operational units and their interconnections.
- Architecture may survive many years, but computer organization changes with changing technology.
- Changes in technology brings changes in organization but also results in more powerful and richer architecture.

Examples:

- Architectural attributes include the instruction set, the number of bits used to represent various data types (e.g. numbers, characters), I/O mechanisms and techniques for addressing memory.
- Organizational attributes include those hardware details transparent to the programmer, such as control signals, interfaces b/w computer and peripherals, and memory technology used.
- e.g. it is an architectural design issue whether a computer will have a multiply instruction, but it is an organizational issue how that instruction will be implemented.

Digital Logic Gates

Sets of switches with different ways of turning them on and off. These switches are called "Gates".

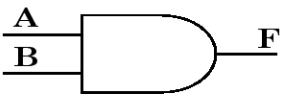
Three Basic Types -

AND Gate
OR Gate
NOT Gate

Other gates are-

NAND Gate
NOR Gate
XOR Gate
XNOR Gate

AND Gate



Truth Table

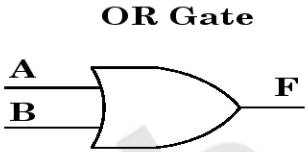
A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

AND:

- The O/P is "true" when both the I/Ps are "true".
- Otherwise, the O/P is "false".
- The AND gate implements the AND function. With the gate shown to the left, both inputs must have logic 1 signals applied to them in order for the output to be a

logic 1. With either input at logic 0, the output will be held to logic 0

OR Gate:



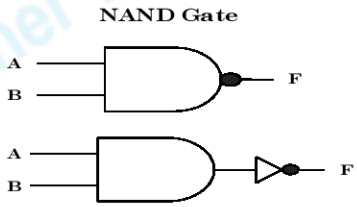
- The O/P is "true" if either or both of the I/Ps are "true".
- If both inputs are "false", then the O/P is "false".

The OR gate is sort of the reverse of the AND gate. The OR function, like its verbal counterpart, allows the output to be true (logic 1) if any one or more of its inputs are true.

Truth Table

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

All other Digital Logic Gates are a combination of the basic gates.



NAND gate operates as an *AND* gate followed by a *NOT* gate. O/P is "false" if both I/Ps are "true" else the O/P is "true".

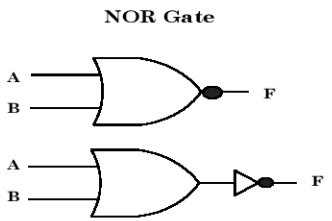
Truth Table
AND Gate

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table
NAND Gate

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

NOR gate operates as an *OR* gate followed by a *NOT* gate. O/P is "true" if both I/Ps are "false" else the O/P is "false".



Truth Table
OR Gate

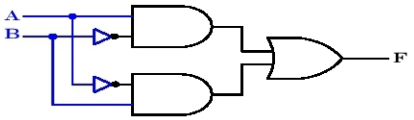
A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

Truth Table
NOR Gate

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

Exclusive-OR gate
The O/P is "true" if only **one** of its inputs are a **"true"**

Truth Table for XOR Gate

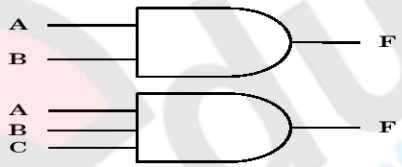


Truth Table

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

- **Exclusive-NOR** gate is a combination of XOR gate followed by an inverter.
- The O/P is "false" if only **one** of its inputs are a **"true"** .

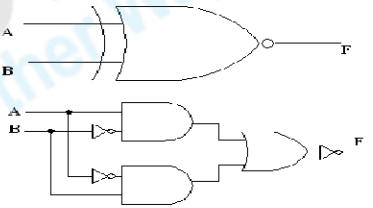
Multi-Input Gates



Truth Table
3 Input AND Gate

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

XNOR (Exclusive NOR)



Truth Table

A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

- NAND and NOR gates are said to be universal gates as any digital system can be implemented with them.
- The NOT operation is obtained from a one I/P NAND gate.
- The AND operation is requires 2 NAND gates, first to produce inverted AND and second acts as inverter to produce normal O/P.
- The OR operation is achieved through a NAND gate with additional inverters in each input

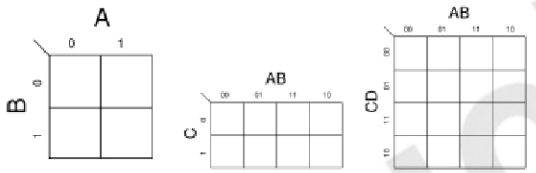
Karnaugh Maps –

The **Karnaugh map**, also known as a **Veitch diagram** (K-map or KV-map for short). It is a simplification tool for managing Boolean algebraic expressions.

k-map method may be regarded as pictorial arrangement of truth table.Each combination of variables in a truth table is called as **Minterm.**

(A function of n variables will have 2^n minterms. A Boolean function is equal to 1 for some minterms and to 0 for others.)

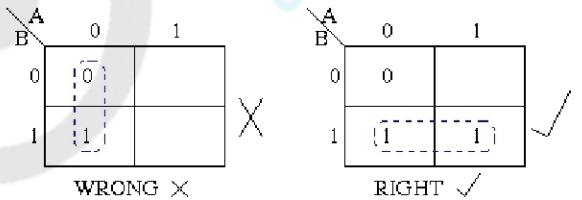
The map is a diagram made of squares with each square representing a minterm.The rows and columns are ordered according to the principles of **Gray code**.



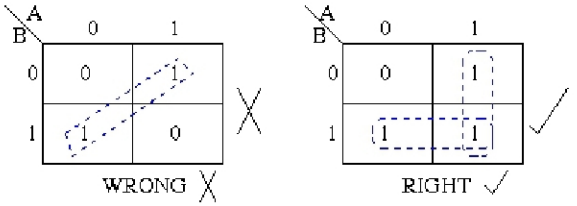
Rules of Simplification

The K-Map uses the following rules for simplification of expressions by grouping together adjascent cells containing ones.

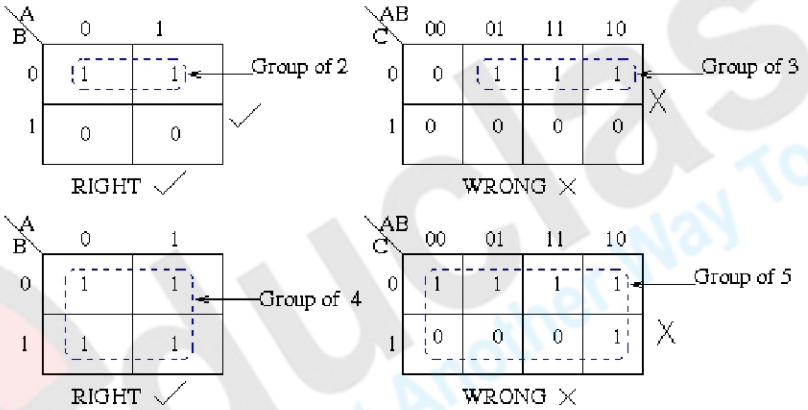
1. **Groups may not include any cell containing a zero**



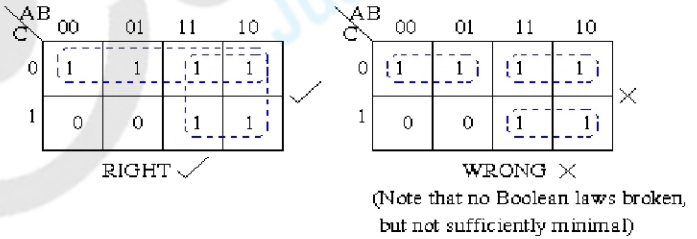
2. **Groups may be horizontal or vertical, but not diagonal.**



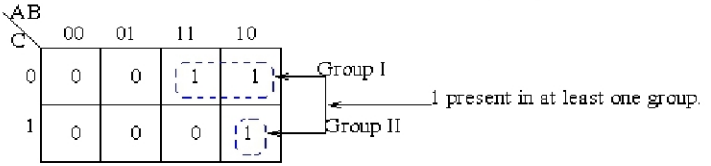
3. **Groups must contain 1, 2, 4, 8, or in general 2^n cells.**
That is if n = 1, a group will contain two 1's since $2^1 = 2$.
If n = 2, a group will contain four 1's since $2^2 = 4$.



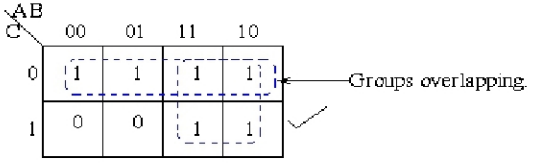
4. **Each group should be as large as possible.**

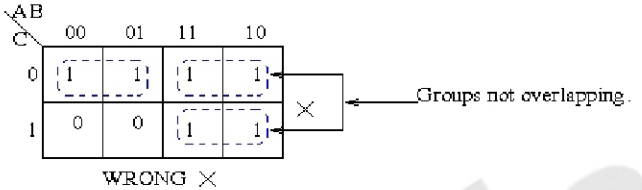


5. **Each cell containing a *one* must be in at least one group.**

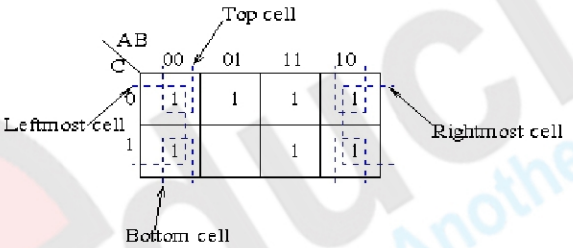


6. **Groups may overlap.**

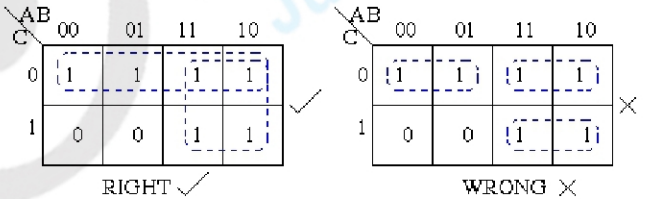




7. **Groups may wrap around the table. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.**



8. **There should be as few groups as possible, as long as this does not contradict any of the previous rules.**

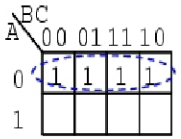


Summmary:

1. No zeros allowed.
2. No diagonals.
3. Only power of 2 numbers of cells in each group.

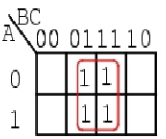
4. Groups should be as large as possible.
5. Each one must be in at least one group.
6. Overlapping allowed.
7. Wrap around allowed.
8. Fewest numbers of groups possible.

Out= $\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}BC + \bar{A}B\bar{C}$



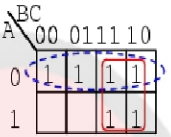
Out= $\bar{A}$

Out= $\bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + ABC$



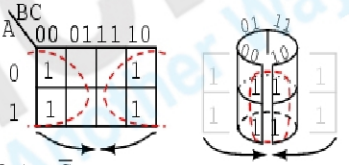
Out= C

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



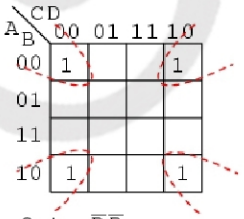
Out= $\bar{A} + B$

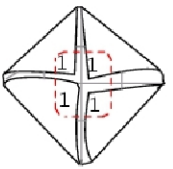
Out= $\bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + \bar{A}B\bar{C} + AB\bar{C}$



Out= $\bar{C}$

Out= $\bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}C\bar{D} + A\bar{B}C\bar{D} + A\bar{B}C\bar{D}$





Out= $\bar{B}\bar{D}$

Combinational Circuit

Any circuit can be configured under either combinational or sequential circuit.

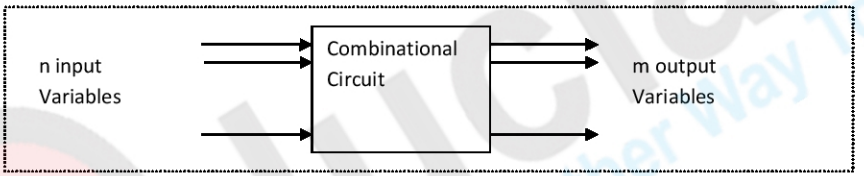
Combinational circuit is one whose O/P at a specified time is function of the inputs at that time.

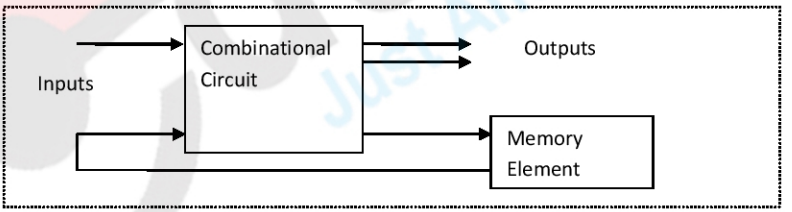
O/P of combinational circuit depends purely upon PRESENT input; hence it doesn't require memory to store past inputs (preceding inputs).

Sequential circuit is a logic circuit whose O/P at specified time is a function of inputs at that time as well as finite number of inputs of preceding time.

Sequential circuit requires Memory element, because its O/P depends upon present as well as preceding inputs.

Basically Sequential circuit is combination of Memory element and combinational circuit.





A combinational circuit consists of input variables, logic gates and output variables. The logic gates accept input and generate signals to output. For n input variables there are 2^n possible input combinations. For each input combination there is only one possible output combination. A combinational circuit can be described by m Boolean functions, one for each output variable. Each output function is expressed in terms of n input variables.

Half Adder

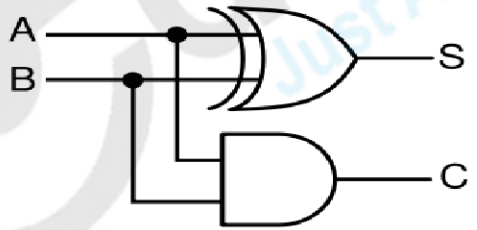
It performs addition of two, 'single' bit numbers.

It performs addition of two bits.

Tuth table is shown below. A and B are two one bit inputs and which produces sum(S) and carry(C) with C being the most significant of these two outputs.

INPUTS		OUTPUTS	
A	B	CARRY	SUM
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Well, this looks familiar, doesn't it? The Carry output is a simple AND function, and the Sum is an Exclusive-OR. Thus, we can use two gates to add these two bits together. The resulting circuit is shown below.



The half adder produces a sum and a carry value which are both binary digits.

$$S = A \oplus B$$

$$C = A \cdot B$$

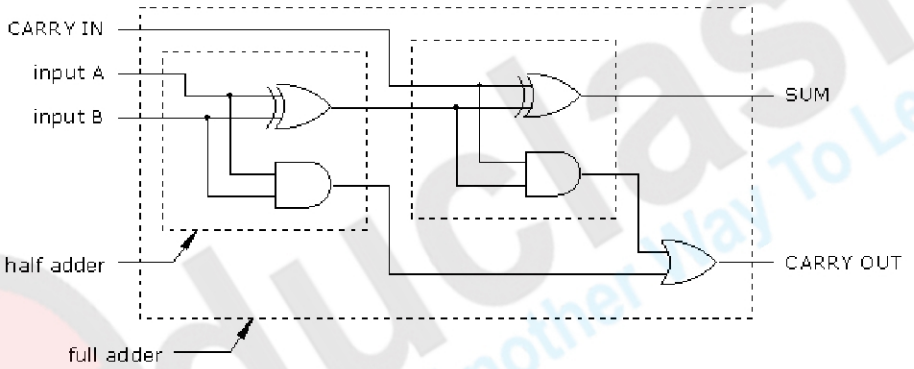
Drawback of half adder is, it doesn't cater to carry i.e. it fails in multibit addition.It will add A_0 and B_0, but in next stage we have to add three bits, which will not be done by the circuit, hence we have to require FULL ADDER.

Full Adder

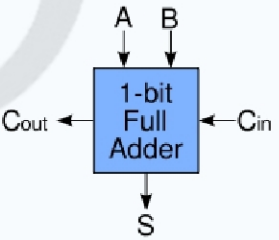
To overcome the drawback of half adder, full adder is designed. It is a 3 single bit adder circuit. It considers **Carry** also. The full adder produces a sum and carry value, which are both binary digits. It can be combined with other full adders (see below) or work on its own.

$$S = (A \oplus B) \oplus C_i$$

$$C_o = (A \cdot B) + (C_i \cdot (A \oplus B)) = (A \cdot B) + (B \cdot C_i) + (C_i \cdot A)$$



A full adder can be constructed from two half adders by connecting A and B to the input of one half adder, connecting the sum from that to an input to the second adder, connecting C_i to the other input and OR the two carry outputs.



Schematic symbol for a 1-bit full adder

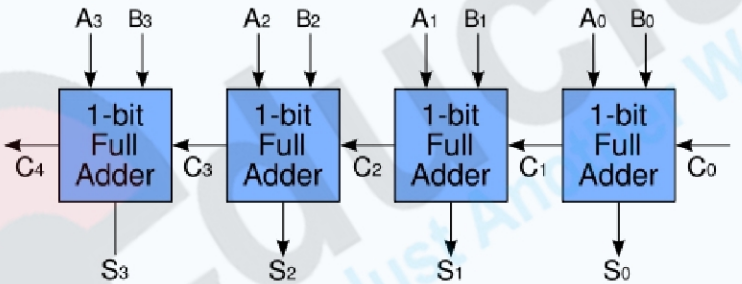
Multiple-bit adders (Ripple carry adder)

Normally we deal with multibit operand (4 bit, 8 bit, 16 bit and so on).

When multiple full adders are used with the carry ins and carry outs chained together then this is called a **ripple carry adder** because the correct value of the carry bit ripples from one bit to the next.

It is possible to create a logical circuit using several full adders to add multiple-bit numbers. Each full adder inputs a C_{in}, which is the C_{out} of the previous adder. This kind of adder is a *ripple carry adder*, since each carry bit "ripples" to the next full adder. Note that the first (and only the first) full adder may be replaced by a half adder.

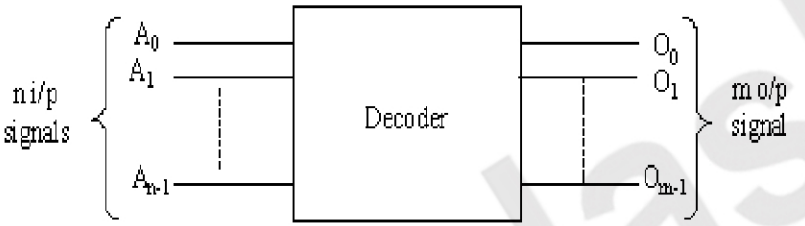
However, the ripple carry adder is relatively slow, since each full adder must wait for the carry bit to be calculated from the previous full adder. The **gate delay** can easily be calculated by inspection of the full adder circuit



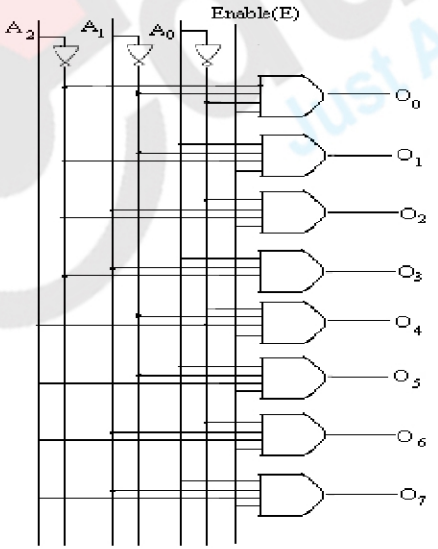
4-bit ripple carry adder circuit diagram

Decoder

- n digital electronics this would mean that a decoder is a multiple-input, multiple-output logic circuit that converts coded inputs into coded outputs, where the input and output codes are different. e.g. n-to-m lines, BCD-to-decimal decoders.
- Decoder is a digital logic circuit that has N input lines and M output lines.



- M and N can be integers such that M<=2N , because 2N possible codes are there.
- Only one of the M lines will be activated depending upon input combination. All the other lines will be deactivated.
- In some of the decoders all 2N possible combinations are not used. But only certain are used. If unused codes are applied, none of the output will be activated.
- So we can have 2-to-4 line decoder, 3-to-8 line decoder, 4-to-16 line decoder



for 3-to-8 line decoder

- Here the 3-to 8 line decoder has 3 I/P lines A0, A1 and A2 one 8 O/P lines.
- E is the enable input to control the operation of the ciruit.

  When E=1, decoder is enabled

  When E=0, decoder is disabled

Encoder

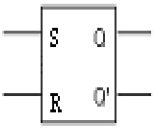
- Performs inverse operation of a decoder.
- An encoder converts an active input signal into a coded binary output signal, e.g. octal-to-binary encoder, and then there will be 8 I/P lines and 3 O/P lines. Out of 8 lines only one will be activated at a time and we get 3 bit O/P code corresponding to activated I/P.
- Encoder has M input lines and N output lines.
- An **encoder** is a device used to change a signal (such as a bitstream) or data into a code. The code may serve any of a number of purposes such as compressing information for transmission or storage, encrypting or adding redundancies to the input code, or translating from one code to another.

Flip-Flops Overview

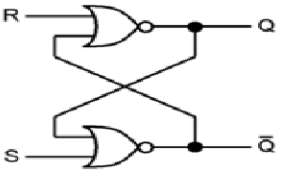
- The most fundamental sequential components are the latch and flip-flop
- They store one bit of data and make it available to other components
- The main difference between a latch and a flip-flop is that the first are level triggered and the latter are edge triggered
- Flip-flops and latches have a clock input
- Flip-Flop is a memory element used in sequential circuit
- FF is an electronic device which has two stable states hence it is a bi-stable device.
- FF has two outputs, one is 0 and other is +5 Vdc
- When the FF has its o/p set at 0 Vdc, it can be regared as stroring a logic 0 and when its o/p is set at +5 Vdc it stores logic 1
- FF is capable of serving as one bit of memory

SR latch

- The most fundamental latch is the simple *SR latch*, where S and R stand for *set* and *reset*.
- It can be constructed from a pair of cross-coupled NOR (negative OR) logic gates.
- It is called NOR-Gate Latch
- The stored bit is present on the output marked Q



Graphical symbol

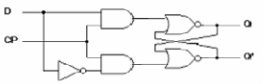


- When both S and R are 0 the output remains unchanged
  I.e. here next state=previous state

- When S=1, R=0, irrespective of previous condition latch is set to 'SET' state
- Similarly when S=0, R=1,it resets the latch to 0
- When both the S and R are 1, the output of the latch is undefined; here both the outputs will try to reach 0, in other words Q=0 and Q'=0 at the same time! which violets the basic definition of FF that requires Q to be complement of Q', it leads to RACE condition of the circuit
- the designer has to ensure that S and R inputs are never set to 1, if this condition is imposed then the result state is unpredictable
- Doesn't have a clock input

Clocked D FF

Logic Diagram :



Level triggered D FF

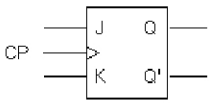
- Below is truth table of Positive level triggered D FF
- When CLK level=0 irrespective of D input, Output remains the same
- When CLK=1 output Q changes as D changes

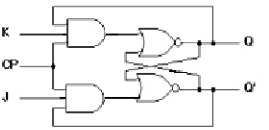
Truth table of positive Level triggered D FF

Input		Output	
CLK	D	Q_{n+1}	Q'_{n+1}
0	X	Q_n	Q'_n
1	0	0	1
1	1	1	0

JK flip-flop

- Resolves the problem of undefined outputs associated with SR latch
- It is often used instead of SR latch.





Truth table of positive edge triggered JK FF

Inputs			Outputs	
CLK	J	K	Q_{n+1}	Q'_{n+1}
0	X	X	Q_n	Q'_n
1	X	X	Q_n	Q'_n
↓	X	X	Q_n	Q'_n
↑	0	0	Q_n	Q'_n
↑	0	1	0	1
↑	1	0	1	0
↑	1	1	Q'_n	Q_n

- When CLK is at HIGH or LOW level, both O/Ps remains in previous state
- When CLK performs transition from 1-to-0, both O/Ps remains in previous state
- When CLK performs transition from 0-to-1, I/P determines the O/P :
  - when J=0, K=0 then state remains unchanged
  - when J=0, K=1 then O/P is in Reset state
  - when J=1, K=0 then O/P is in Set state
  - when J=0, K=1 then O/P is in Toggle state i.e. if Q=1, it switches to Q=0 and vice versa

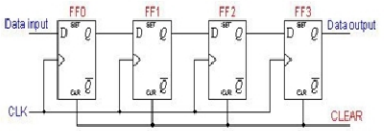
Observations

- Most commonly used flip-flops are D FFs because they are useful for temporary storage of data
- Counter circuits can be implemented efficiently by using T FFs
- JK FF combines the behaviors of SR and T FFs
- The JK FF is versatile. It can be used to store data just as D FF. It can also be used to build counters, as it behaves like T FF if J and K input terminals are connected together.
- All of the flip-flops and latches shown so far are positive edge triggered or positive level triggered. They also have active high load, set and clear inputs.
- It is possible for those components to be negative edge triggered or negative level triggered and has active low control signals as well.
- Flips-flops and latches can be combined in parallel to store data with more than one bit

Registers

- A Register is a group of flip-flops that can be used to store a binary number
- They are a group of flip-flops connected in a chain so that the output from one flip-flop becomes the input of the next flip-flop
- All the flip-flops are driven by a common clock, and all are set or reset simultaneously
- A register that stores 5-bit binary number must have 5 FFs. So in general n-bit register has a group of n flip-flops and it can store n bit binary information
- In addition to flip-flops, a register may have combinational gates to perform certain data processing tasks
- A register capable of shifting binary info either to right or to left is called a **shift register**.
- They are a group of flip-flops connected in a chain so that the output from one flip-flop becomes the input of the next flip-flop
- All the flip-flops are driven by a common clock, and all are set or reset simultaneously
- The clock pulse causes the shift from one stage to next.
- Bits can be moved from one place to another in two ways.
- **Serial shifting:** shifts the data 1 bit at a time in serial fashion, starting with either MSB or LSB
- **Parallel shifting:** shifts all data bits simultaneously
- **Types of Registers**
    - Serial in-Serial out
    - Serial in-parallel out
    - Parallel in-Serial out
    - Parallel in-parallel out

Serial In - Serial Out Shift Register



	FF0	FF1	FF2	FF3
CLEAR	0	0	0	0

- A basic four-bit shift register can be constructed using four D flip-flops
- The register is first cleared, forcing all four outputs to zero.
- The input data is then applied sequentially to the D input of the first flip-flop on the left (FF0). During each clock pulse, one bit is transmitted from left to right.
- Assume a data word to be 1001. The least significant bit of the data has to be shifted through the register from FF0 to FF3

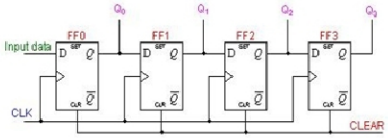
- To readout the data, bits must be shifted out serially
- **destructive readout**: the original data is lost and at the end of the read cycle, all flip-flops are reset to zero

	FF0	FF1	FF2	FF3	
0000	1	0	0	1	0000

- **Non-destructive readout**: any data shifted out of the register at the right becomes the next input at the left, and is kept in the system

WRITE	FF0	FF1	FF2	FF3
1001	0	0	0	0

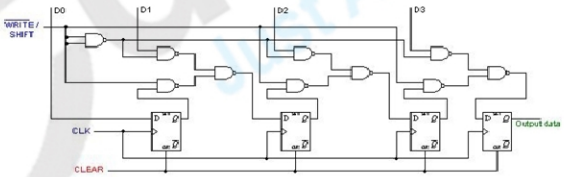
Serial In - Parallel Out Shift Register



	Q0	Q1	Q2	Q3
10	0	1	0	0

- A basic four-bit shift register can be constructed using four D flip-flops
- The register is first cleared, forcing all four outputs to zero.
- The input data is then applied sequentially to the D input of the first flip-flop on the left (FF0). During each clock pulse, one bit is transmitted from left to right.
- Once the data are stored, each bit appears on its respective output line, and all bits are available simultaneously

Parallel In - Serial Out Shift Register

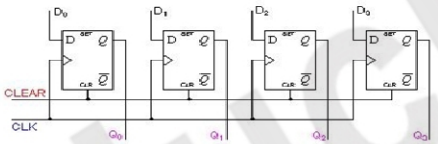


- The circuit uses D flip-flops and NAND gates for entering data (i.e. writing) to the register.
- D0, D1, D2 and D3 are the parallel inputs, where D0 is the most significant bit and D3 is the least significant bit.

- To write the data, mode control line is taken to LOW, to shift the data, mode control line is taken to HIGH.
- The register performs right shift operation on the application of a clock pulse

	Q0	Q1	Q2	Q3
CLEAR	0	0	0	0

Parallel In - Parallel Out Shift Register



- D0, D1, D2, D3 are parallel inputs and corresponding Q's are parallel outputs.
- All data bits are entered simultaneously and once the register is clocked, all the data at the D inputs appear at the corresponding Q outputs simultaneously.

USES:

- **One of the most common uses of a shift register is to convert between serial and parallel interfaces.**
- **Shift registers can be used as simple delay circuits.**
- **Several bi-directional shift registers could also be connected in parallel for a hardware implementation of a Stack.**

- **Shift registers can be used as pulse generator.**
- **Random bit generator**

Counters

- A **counter** is a device which stores the number of times a particular event or process has occurred, often in relationship to a clock signal
- It counts the number of clock pulses applied to it.
- A counter is always specified by list of *count sequence*, i.e. the sequence of binary states that it undergoes. The count sequence repeats after it reaches its last value.
- Counter that follows binary sequence is called a binary counter.
- An n-bit counter consists of n FF and can count in binary from 0 to 2^n-1.
- A BCD counter counts the binary sequence from 0000 to 1001 and returns to 0000 to repeat the sequence.
- We can design counters that follow some arbitrary sequence.
- They can be divided into 2 groups:

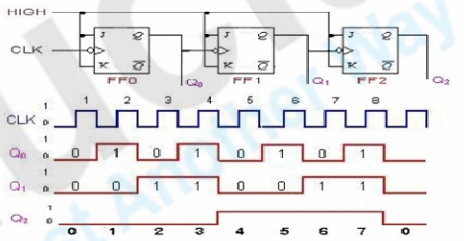
  --Asynchronous counter (serial / ripple counter)

  --Synchronous counter (parallel counter)

Asynchronous (Ripple) counter

- A binary ripple counter can be constructed using clocked JK or T flip-flops.
- When the output of a flip-flop is used as clock input for the next flip-flop, then the counter is called a ripple or asynchronous counter.
- The A FF must change state before it can trigger B FF, and B FF has to change state before it can trigger C FF.
- The triggers move through FFs like a ripple in water.
- Usually, all the CLEAR inputs are connected together, so that a single pulse can clear all the flip-flops before counting starts
- The 3-bit ripple counter circuit above has 3 different states, each one corresponding to a count value. Similarly, a counter with *n* flip-flops can have 2 *to the power n* states. The number of states in a counter is known as its mod (modulo) number. Thus a 3-bit counter is a mod-8 counter.

Three bit asynchronous binary counter 6



Timing diagram

- Here FFs are negatively edge triggered. CLK signal is supplied to only one flip-flop FF0. Every time on clock NT, FF0 will change state. So we have waveform of Q_0
- FF0 will act as the clock for FF1 and FF1 will act as clock for FF2. Each time the waveform Q_0 goes low Q_1 will change the state.

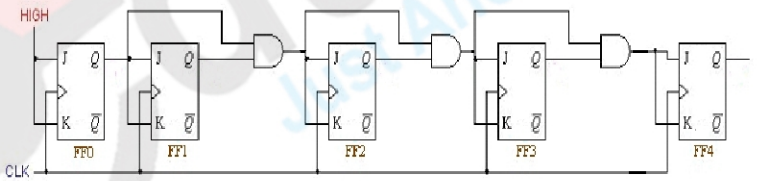
A binary counter with a reverse count is called a binary down-counter.

Such counter is decremented by 1 with every input count pulse. So a 4 bit counter starts from binary 15 i.e. 1111 and continues to binary counts 14,13,12,…..,0 then back to 15.

A binary up counter will work as a binary down counter if outputs are taken from the complement terminals Q'of all flip-flops.

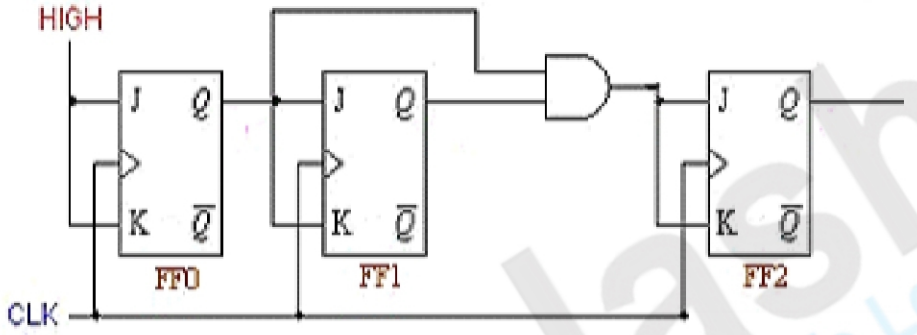
Synchronous (Parallel) counter

- In synchronous counters, every flip-flop is triggered in synchronism with the clock. Thus, all the flip-flops change state simultaneously.
- The circuit below is a 5-bit synchronous counter. The J and K inputs of FF0 are connected to HIGH. FF1 has its J and K inputs connected to the output of FF0, and the J and K inputs of FF2 are connected to the output of an AND gate that is fed by the outputs of FF0 and FF1.
- Synchronous counters can be designed to count up or down, or both according to a direction input



In synchronous binary Up-Counter, the flip-flop in the lowest order position is complemented with every pulse. So its J & K inputs must be maintained at logic1.

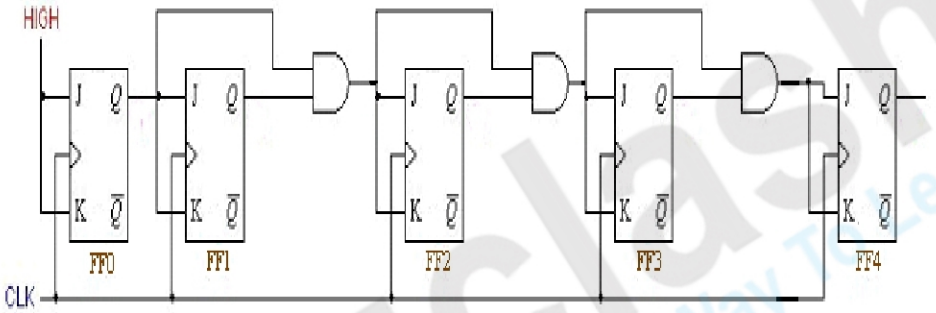
A flip-flop in any other position is complemented with a pulse provided all the bits in the lower order positions are equal to 1.



3-bit synchronous counter

- The circuit below is a 3-bit synchronous counter. The J and K inputs of FF0 are connected to HIGH. FF1 has its J and K inputs connected to the output of FF0, and the J and K inputs of FF2 are connected to the output of an AND gate that is fed by the outputs of FF0 and FF1.
- A binary Down-Counter can be constructed just like binary up counter except that the inputs to AND gates must come from Q′ and not from normal outputs Q
- In synchronous binary Down-Counter the flip-flop in the lowest order position is complemented with every pulse. So its J & K inputs must be maintained at logic1.

- **A flip-flop in any other position is complemented with a pulse provided all the bits in the lower order positions are equal to 0.**
- **5-bit synchronous counter**



Comparison

- Ripple counter is simplest to build, but there is a limit to its highest operating frequency.
- Each FF has a delay time, these delays are additive in ripple counters.
- There is some possibility of glitches occurring at the output of decoding gates used with a ripple counter.
- These problems can be overcome by the use of synchronous counter.

Up-Down Counters

- It is a combination of up counter and down counter, counting in straight binary sequence. There is an up-down selector. If this value is kept high, counter increments binary value and if the value is low, then counter starts decrementing the count.

Ring Counters

- A ring counter is a counter that counts up and when it reaches the last number that is designed to count up to, it will reset itself back to the first number. For example, a ring counter that is designed using 3 JK Flip Flops will count starting from 001 to 010 to 100

and back to 001. It will repeat itself in a 'Ring' shape and thus the name Ring Counter is given