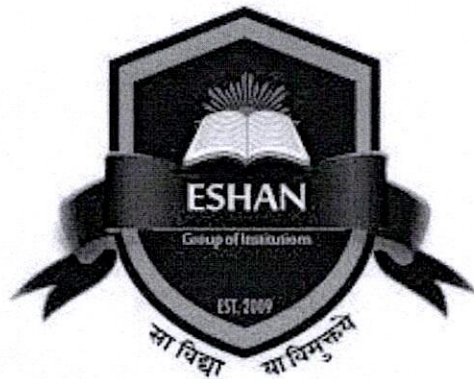


Eshan College of Engineering

(Approved by AICTE, New Delhi, Affiliated to Dr. A.P.J Abdul Kalam Technical University, Lucknow)
Sahzadpur Pauri, NH-2, Agra-Mathura Highway, Mathura-281122, Uttar Pradesh
Website: www.eshancollege.com



Sample of Content Beyond Syllabus offered in the Institute



Eshan College of Engineering, Farah

Department of Computer Science & Engineering

Subject Name (Code): OPERATING SYSTEM (KCS401)

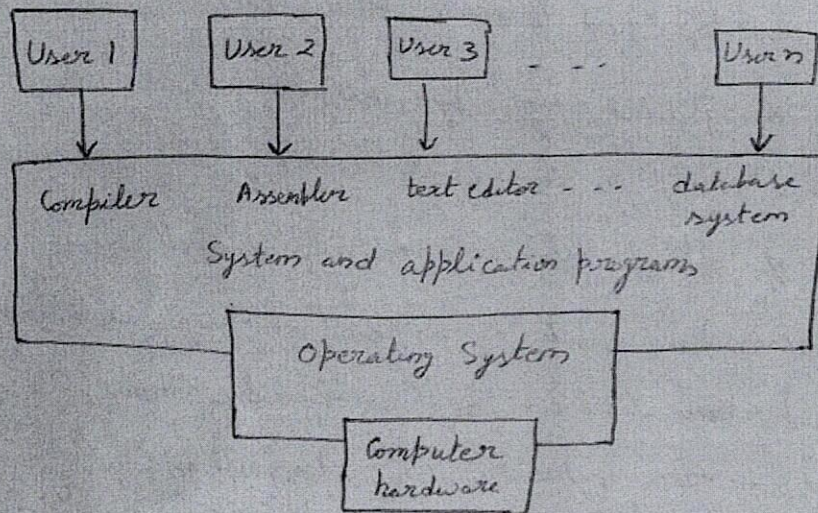
Contents beyond Syllabus

1. Abstract view of the components of a computer system
2. Computer system structure
3. I/O interrupt
4. Monitors
5. Address binding mechanism

Course Teacher



Abstract view of the components of a computer system



The components of a computer system are its hw , sw , and data. The OS provides the means for the proper use of these resources in the opⁿ of the computer system.

User View - User view varies by the interface being used.

For PC having one user, the OS is designed for 'ease of use' with no attention paid to resource utilization, and some attention paid to performance.

For users sit at a terminal connected to a mainframe or minicomputer while other users are accessing the same computer through other terminals, OS is designed to maximize 'resource utilization'.

Users sit at workstations, connected to other workstations or servers, have dedicate resources at their disposal, but they also share common resources such as networking and

servers. So OS is designed to compromise between individual usability and resource utilization.

For handheld devices used singly by individual users, the OS are designed mostly for individual usability, but performance per amount of battery life is important as well.

Some computers have little or no user view eg. embedded computers in home devices and automobiles. These OS are designed to run without user intervention.

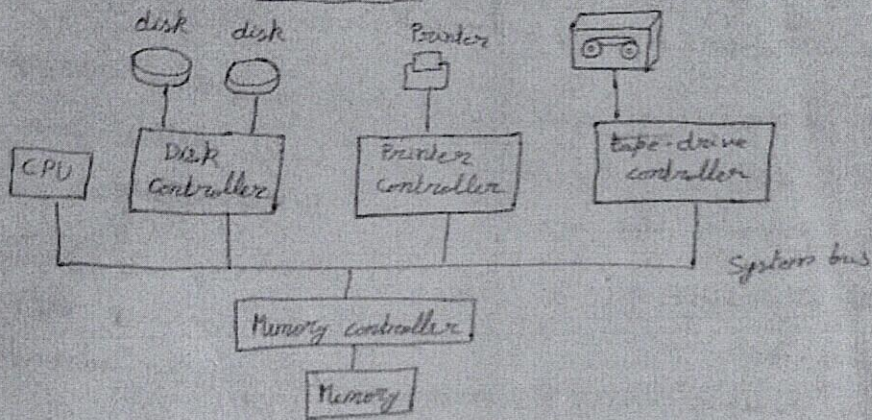
System view - The OS is the program that is most intimate with the hardware. OS can be viewed as a 'resource allocator'.

An OS is a 'control program' that manages the execution of user programs to prevent errors and improper use of the computer.

Computer System Structures

9

Computer System Operation

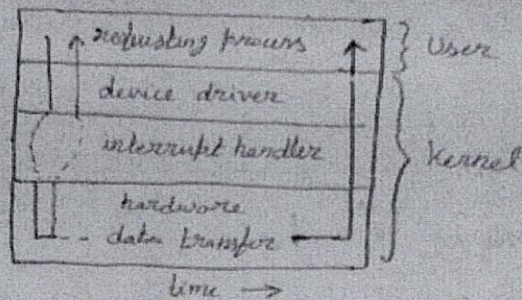


A computer system has a CPU & no. of device controllers that are connected through a common bus that provides access to shared memory. The CPU & device controllers can execute concurrently, competing for memory cycles. Memory controllers synchronize access to the memory.

Bootstrap Program - When a computer is powered up or reboot, it needs to have an initial program to run. This is bootstrap program which stores in ROM. It initializes all aspects of the system, from CPU registers to device controllers to memory contents. This program must know how to load the OS and to start executing the system. For this, bootstrap program must locate & load into memory the OS kernel. The OS then starts executing the first process.

Interrupts - The occurrence of an event is signalled by an interrupt from either the HW or the SW. Hardware may

Asynchronous I/O



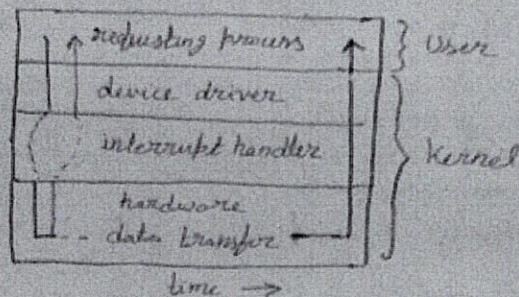
In asynchronous I/O, the control is returned to user program without waiting for the I/O to complete. The I/O then can continue while other system operations occur. Asynchronous I/O increases system efficiency.

Dual-Mode operation - This approach allows us to differentiate among various modes of operation. We need two operation modes: user mode and monitor mode (also called supervisor mode, system mode, or privileged mode). A bit, called the mode bit is added to the b/w of computer to indicate the current mode: monitor (0) or user (1). With mode bit, we are able to distinguish b/w a task that is executing on behalf of OS, and one that is executed on behalf of user.

At system boot time, the b/w starts in monitor mode. The OS is then loaded, and starts user processes in user mode. Whenever a trap or interrupt occurs, the b/w switches from user mode to monitor mode. So, whenever OS gains control of computer, it is in monitor mode. The system always switches to user mode before passing control to a user program.

User mode. Creating a text is using any application prog.
When a user application requests for a service from the OS or an interrupt occurs or system call, then transition from user to kernel mode.

Asynchronous I/O



In asynchronous I/O, the control is returned to user program without waiting for the I/O to complete. The I/O then continues while other system operations occur. Asynchronous I/O increases system efficiency.

Dual-Mode operation - This approach allows us to differentiate among various modes of operations. We need two operation modes: user mode and monitor mode (also called supervisor mode, system mode, or privileged mode). A bit, called the mode bit, is added to the bit of computer to indicate the current mode: monitor (0) or user (1). With mode bit, we are able to distinguish b/w a task that is executing on behalf of OS, and one that is executed on behalf of user.

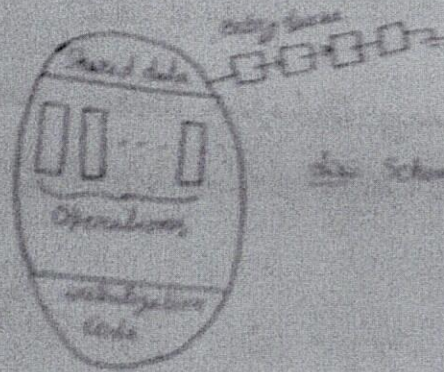
At system boot time, the bit starts in monitor mode. The OS is then loaded, and starts user processes in user mode. Whenever a trap or interrupt occurs, the bit switches from user mode to monitor mode. So, whenever OS gains control of computer, it is in monitor mode. The system always switches to user mode before passing control to a user program.

User mode - Creating a task is using any application prog
when the user application requests for a service from the OS or an interrupt occurs or system call, then transition from user to kernel mode.

Monitors - Monitor type is a high-level synchronization construct. A monitor is characterized by a set of programmer-defined operators. The representation of monitor type consists of declaration of variables whose value defines the state of an instance of the type, as well as the bodies of procedures or functions that implement operations on the type. The syntax of a monitor is -

```
monitor monitor-name
{
    shared variable declarations
    procedure body P1 (--)
    {
        --
    }
    procedure body P2 (--)
    {
        --
    }
    :
    procedure body Pn (---)
    {
        --
    }
    {
        initialization code
    }
}
```

The monitor construct ensures that only one process at a time can be active within the monitor. The programmer does not need to code this synchronization constraint explicitly. However, the monitor construct is not sufficiently powerful for modeling some synchronization schemes.

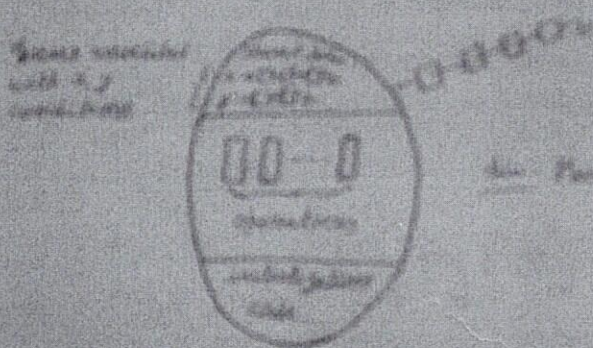


Schematic view of a monitor

To model synchronization schemes, we use condition constructs.
A programmer who wants to write his own synchronization scheme can define one or more variables of type 'condition':
(condition 2.1)

The only op's that can be executed on a condition variable are 'wait' and 'signal'. The op? $x\text{-wait}()$, means that the process executing this op? is suspended until another process executes $x\text{-signal}()$.

The $x\text{-signal}()$, op? resumes exactly one suspended process.



Monitor with condition variables

When a process (monitor) $x\text{-signal}()$ op? there is a suspended process & associated with condition x . There are two possibilities:

1. P enters with wait & leaves the monitor, or waits for another condition.
2. Q enters with wait & leaves the monitor, or waits for another condition.

Memory Management

Address Binding Mechanism A program may be moved into main memory during its execution. The collection of programs on the disk that is waiting to be brought into memory for execution forms the input queue.

Each system often has some process to search in any part of the physical memory although the physical address space of the computer does not access the first address of user process does not need to be changed. A user program goes through following steps before being executed.

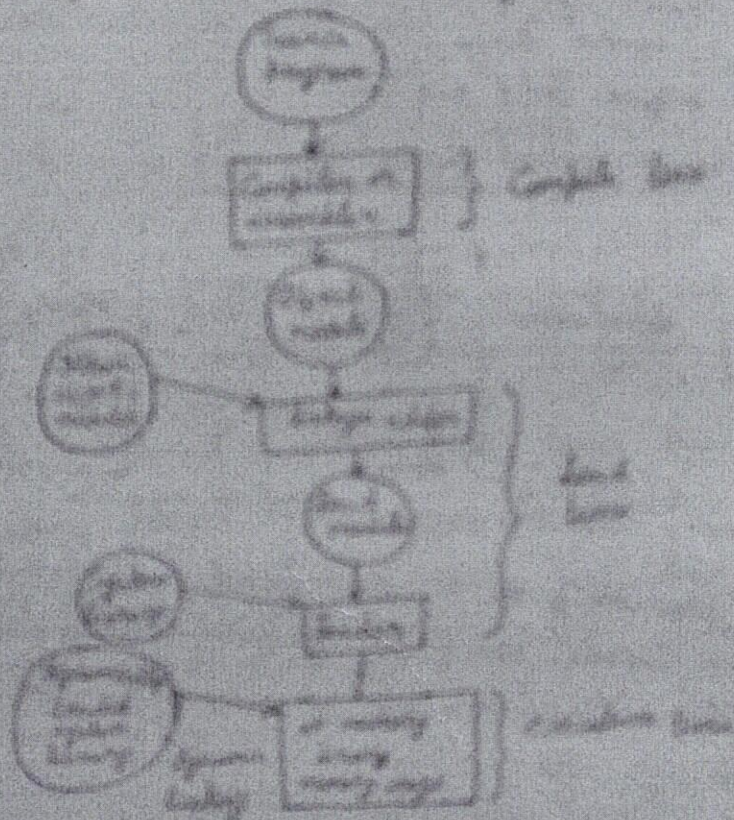


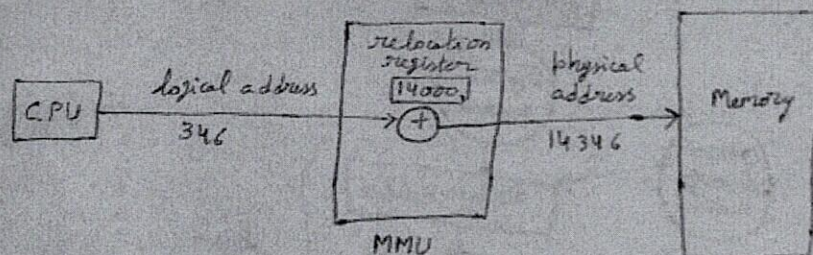
Fig. Methods processing of a user program

Addresses may be represented in different ways during these steps.

The binding of instructions and data to memory addresses can be done at any step along the way.

1. Compile time - If you know at the compile time where the process will reside in memory, then 'absolute code' can be generated.
2. Load time - If we don't know at compile time where the process will reside in memory, then compiler must generate 'relocatable code'. In this case final binding is delayed until load time.
3. Execution time - If a process can be moved from one memory segment to another during its execution, then binding must be delayed until run time.

Logical - versus Physical - Address Space



dia - Dynamic relocation using a relocation register

An address generated by the CPU is referred to as 'logical address'. An address seen by the computer memory unit - i.e., the address loaded into memory-address register of the memory is called as 'physical address'.

The set of all logical addresses generated by a program is a "logical-address space". The set of all physical addresses

corresponding to these logical addresses is a 'physical-address space'.

The run time mapping from virtual (logical) to physical addresses is done by 'memory-management unit' (MMU). The value in the relocation register is added to every address generated by a user process.

The user program deals with logical addresses. Logical addresses range from 0 to max and physical address ranges from $R+0$ to $R+max$ for a base value R .

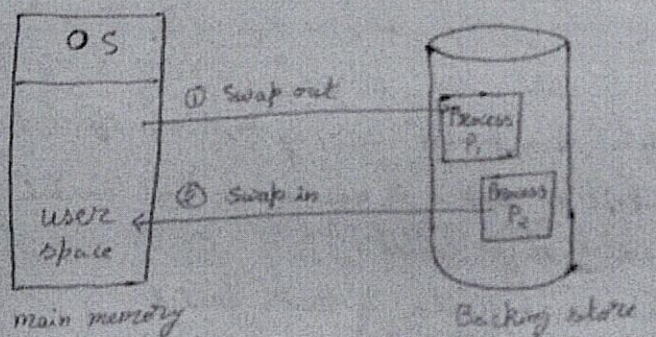
Dynamic Loading - To obtain better memory-space utilization, dynamic loading is used. Here, instead of loading the entire program and data of a process in physical memory, all routines are kept on disk. A routine is not loaded into memory until it is called.

- Advantages - 1. An unused routine is never loaded.
2. This method is useful when large amounts of code are needed to handle infrequently occurring cases.

Overlay - To enable a process to be larger than the amount of memory allocated to it, we can ~~use~~ ^{use} overlays. The idea is to keep in memory only those instructions and data that are needed at any given time. When other instructions are needed, they are loaded into space occupied by previously by instructions that are no longer needed.

Swapping - A process needs to be in memory to be executed. It can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution. eg in multiprogramming environment with a round-robin CPU scheduling.

Swapping may also occur in case higher priority processes arrive and wants the service. When the higher-priority process finishes, the lower-priority process can be swapped back in and continued. This is called "roll out, roll in".



Swapping requires a backing store. It must be large enough to accommodate copies of all memory images for all users, and it must provide direct access to these memory images.

Contiguous memory allocation - Each logical object is placed in a set of memory locations with consecutive addresses.

Non-Contiguous Memory allocation -

Eshan College of Engineering, Farah

Department of _MECHANICAL ENGINEERING

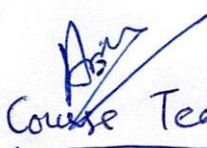
Course Code_KME-503

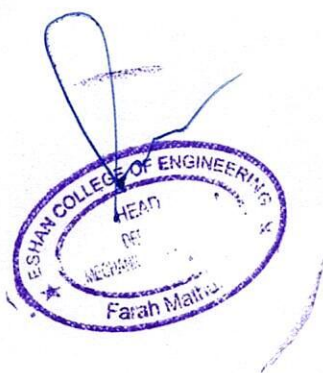
Course Name: B.TECH

Subject: Industrial Engineering

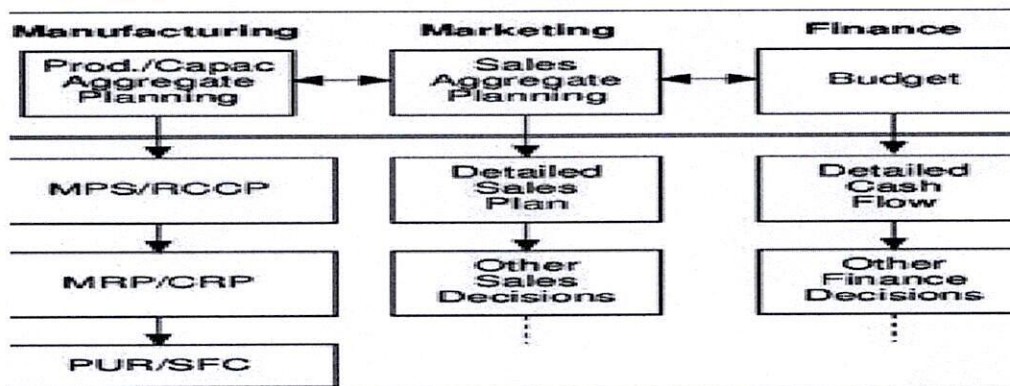
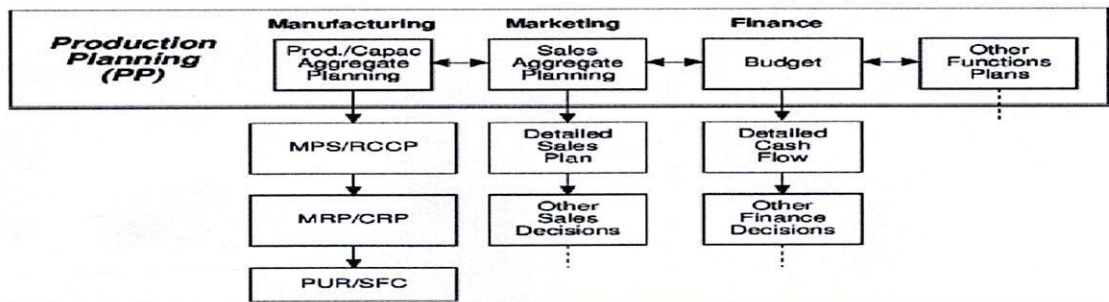
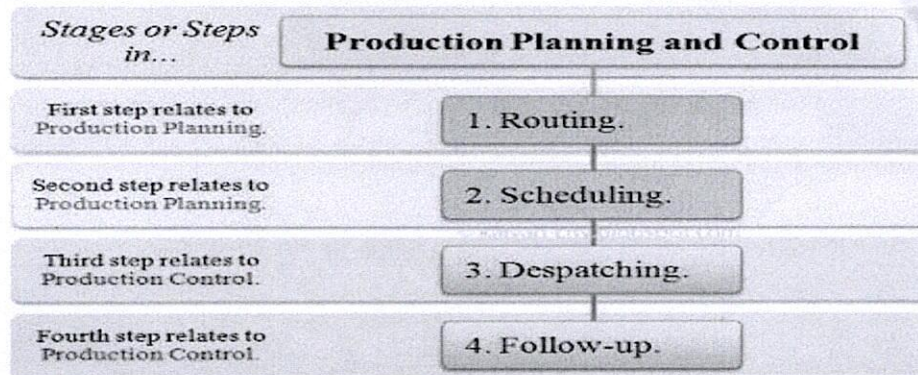
Contents Beyond Syllabus

- Production and Operations planning
- International production and operation management
- Materials handling
- Manufacturing industries


Course Teacher



Production and Operations planning



International production and operation management

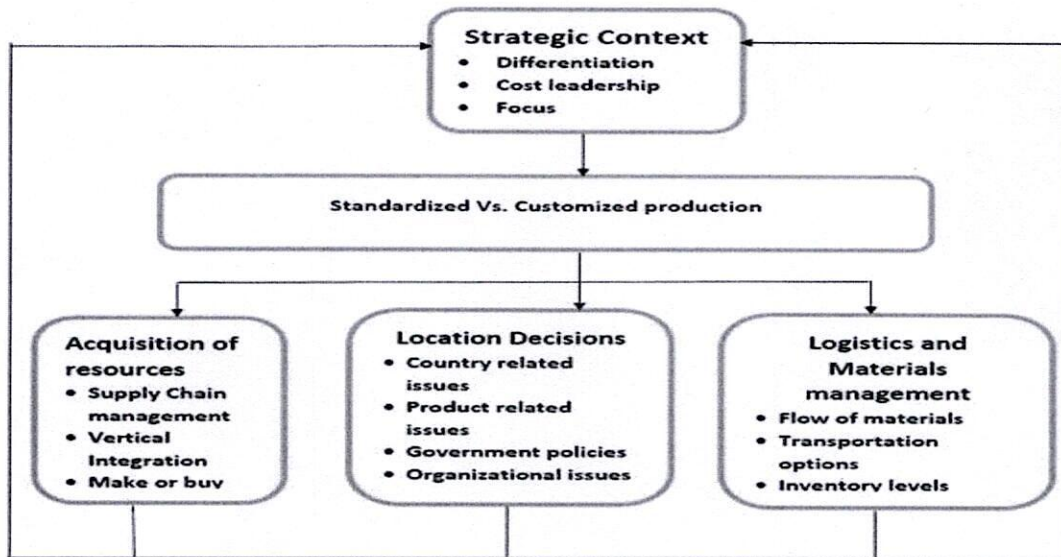


Figure 17.1 The International Operations Management Process



Operations Management

Operations management is the set of activities an organization uses to transform different kinds of inputs (materials, labor, and so on) into final goods and services.

International operations management refers to the transformation-related activities of an international firm.

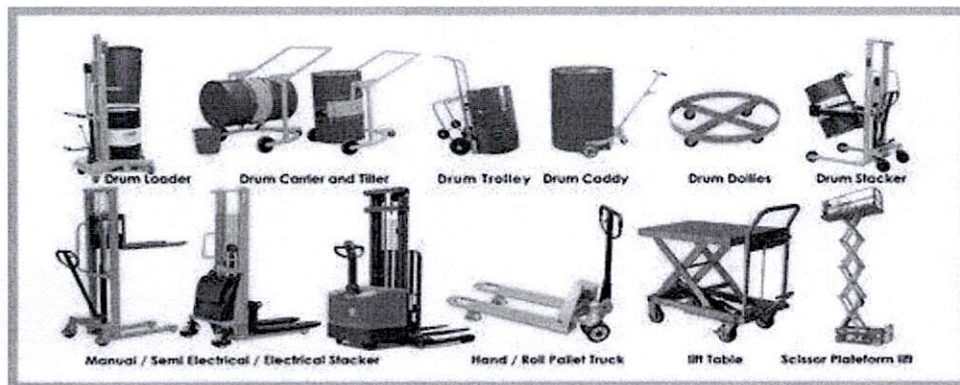
Materials handling

materials handling, the movement of raw goods from their native site to the point of use in manufacturing, their subsequent manipulation in production processes, and the transfer of finished products from factories and their distribution to users or sales outlets. materials



Dos and Don'ts

handling.

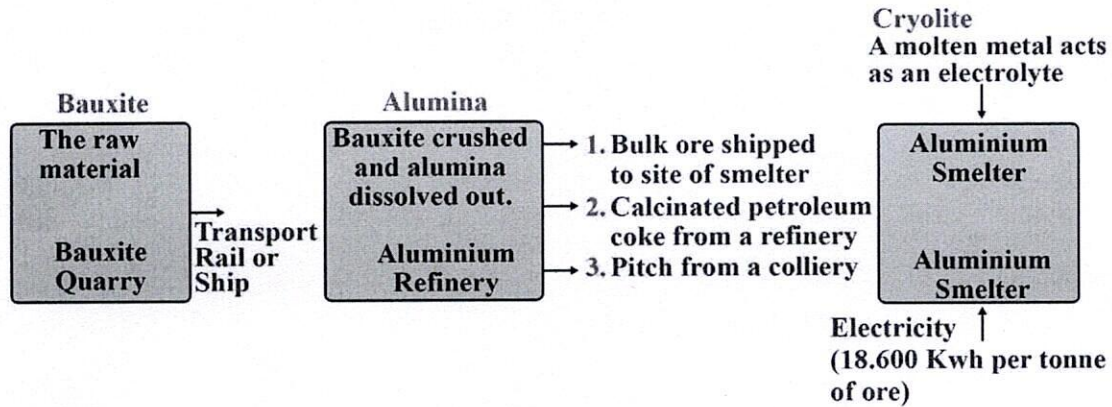


Material handling is the movement, protection, storage and control of materials and products throughout manufacturing, warehousing, distribution, consumption and disposal. As a process, material handling incorporates a wide range of manual, semi-automated and automated equipment and systems that support logistics and make the supply chain work. Their application helps with:

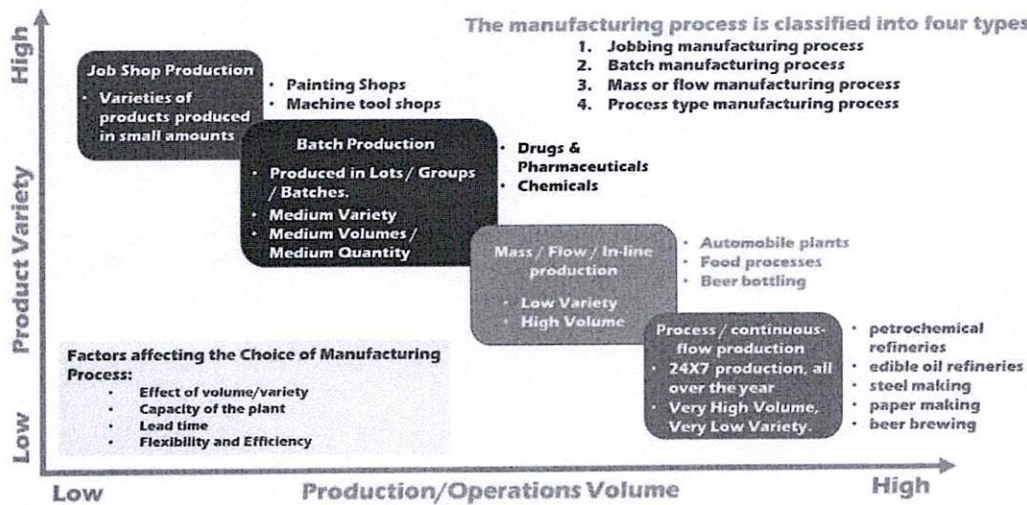
- Forecasting
- Resource allocation
- Production planning
- Flow and process management
- Inventory management and control
- Customer delivery
- After-sales support and service

Manufacturing industries

Process of Manufacturing in Aluminium Industry



Types Of Manufacturing Processes



Eshan College of Engineering, Farah

Department of Computer Science & Engineering

Subject Name (Code): Discrete Structures and Theory of Logic(KCS303)

Contents beyond Syllabus

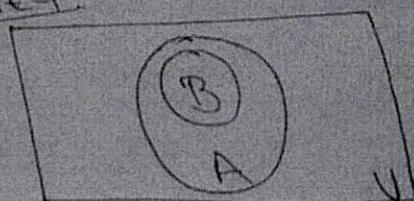
1. Venn Diagram
2. Properties of Permutations
3. Half Adder , Full Adder and Binary Adder
4. Derived Connectives
5. Traveling Salesman Problem

Signature
Course Teacher

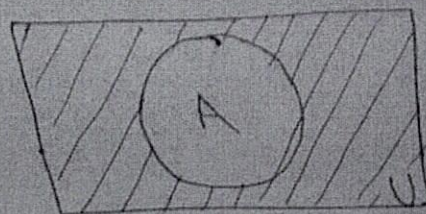


Venn Diagram. Symbol. Unit-L.

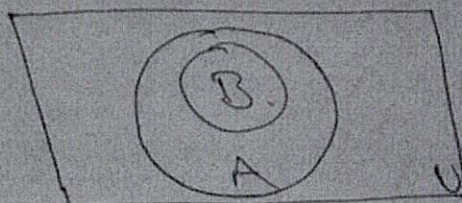
Set B is a proper subset of A $B \subset A$



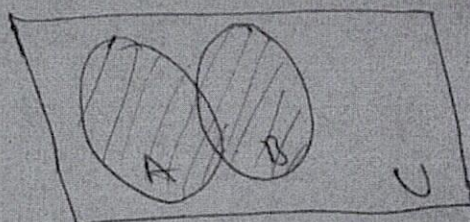
The Complement of Set A A'



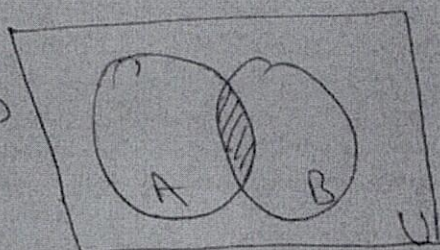
The difference of Set A and B $A - B$



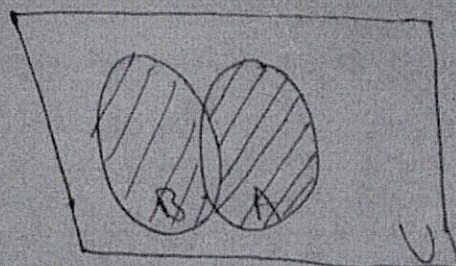
The Union of Sets A and B $A \cup B$



The intersection of sets A and B $A \cap B$



The Symmetric difference of Sets A and B $A \Delta B$



In general, a permutation f on the set $\{1, 2, 3, \dots, n\}$ can be written as

$$f = \begin{pmatrix} 1 & 2 & 3 & \dots & n \\ f(1) & f(2) & f(3) & \dots & f(n) \end{pmatrix}$$

Obviously, the order of the column in the symbol is immaterial so long as the corresponding elements above and below in that column remain unchanged.

Equality of Two Permutations

Let f and g be two permutations on a set X . Then $f = g$ if and only if $f(x) = g(x)$ for all x in X .

Example 40. Let f and g be given by

$$f = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix} \quad g = \begin{pmatrix} 3 & 2 & 1 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix}$$

Evidently $f(1) = 2 = g(1)$, $f(2) = 3 = g(2)$

$f(3) = 4 = g(3)$, $f(4) = 1 = g(4)$

Thus $f(x) = g(x)$ for all $x \in \{1, 2, 3, 4\}$ which implies $f = g$.

Identity Permutation

If each element of a permutation be replaced by itself. Then it is called the identity permutation and is denoted by the symbol I . For example,

$$I = \begin{pmatrix} a & b & c \\ a & b & c \end{pmatrix} \text{ is an identity permutation.}$$

Product of Permutations (or Composition of Permutation)

The product of two permutations f and g of same degree is denoted by $f \circ g$ or fg , meaning first perform f and then perform g .

$$f = \begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ b_1 & b_2 & b_3 & \dots & b_n \end{pmatrix}$$

$$g = \begin{pmatrix} b_1 & b_2 & b_3 & \dots & b_n \\ c_1 & c_2 & c_3 & \dots & c_n \end{pmatrix}$$

Then

$$f \circ g = \begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ c_1 & c_2 & c_3 & \dots & c_n \end{pmatrix}$$

For, f replaces a_1 by b_1 and then g replaces b_1 by c_1 so that $f \circ g$ replaces a_1 by c_1 . Similarly $f \circ g$ replaces a_2 by c_2 , a_3 by c_3 , ..., a_n by c_n .

Clearly $f \circ g$ is also a permutation on S .

It should be observed that the permutation g has been written in such a manner that the second row of f coincides with the first row of g . This is most essential in order to find $f \circ g$.

If we want to write gf , then f should be written in such a manner that the second row of g must coincide with the first row of f .

Example 41. Find the product of two permutations and show that it is not commutative.

$$f = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix} \quad \text{and} \quad g = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 4 \end{pmatrix}$$

Solution.

$$fg = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 4 \end{pmatrix}$$

$$\begin{aligned}
 &= \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix} \begin{pmatrix} 2 & 1 & 4 & 3 \\ 2 & 3 & 4 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix} \\
 gf &= \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 4 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix} \\
 &= \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 4 \end{pmatrix} \begin{pmatrix} 3 & 2 & 1 & 4 \\ 4 & 1 & 2 & 3 \end{pmatrix} \\
 &= \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 2 & 3 \end{pmatrix}.
 \end{aligned}$$

We observe that $fg \neq gf$.

This shows that the product of two permutations is not commutative.

But it can be shown that permutation multiplication is associative.

$$\text{Let } P_1 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} \quad P_2 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix} \quad P_3 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$$

$$\begin{aligned}
 \therefore P_1 (P_2 P_3) &= \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} \left[\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} \right] \\
 &= \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}
 \end{aligned}$$

and

$$\begin{aligned}
 (P_1 P_2) P_3 &= \left[\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix} \right] \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} \\
 &= \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}
 \end{aligned}$$

$$\therefore P_1 (P_2 P_3) = (P_1 P_2) P_3$$

Inverse Permutation

Since a permutation is one-one onto map and hence it is invertible, i.e., every permutation on a set

$$P = \{a_1, a_2, \dots, a_n\}$$

has a unique inverse permutation denoted by f^{-1} .

Thus if

$$f = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \end{pmatrix}$$

then

$$f^{-1} = \begin{pmatrix} b_1 & b_2 & \dots & b_n \\ a_1 & a_2 & \dots & a_n \end{pmatrix}.$$

Total Number of Permutations

Let X be a set consisting of n distinct elements. Then the elements of X can be permuted in $n!$ distinct ways. If S_n be the set consisting of all permutations of degree n , then the set S_n will have $n!$ distinct permutations of degree n . This set S_n is called the symmetric set of permutations of degree n .

For example, if $A = \{1, 2, 3\}$, then $S_3 = \{p_0, p_1, p_2, p_3, p_4, p_5\}$ where

$$p_0 = I_A = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} \quad p_1 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix} \quad p_2 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} \quad p_3 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}$$

$$P_4 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix} \quad P_5 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}$$

The multiplication table for the composition of permutations in S_3 is as given below:

Multiplication Table for S_3

\circ	P_0	P_1	P_2	P_3	P_4	P_5
P_0	P_0	P_1	P_2	P_3	P_4	P_5
P_1	P_1	P_2	P_0	P_5	P_3	P_4
P_2	P_2	P_0	P_1	P_4	P_5	P_3
P_3	P_3	P_4	P_5	P_0	P_1	P_2
P_4	P_4	P_5	P_3	P_2	P_0	P_1
P_5	P_5	P_3	P_4	P_1	P_2	P_0

The table shows that

- (i) The multiplication of any two permutations of S_3 gives a permutation of S_3 . So, S_3 is closed with respect to multiplication.
- (ii) Associativity law holds for $(P_1 P_2) P_3 = P_1 P_2 P_3 = P_0$ and $P_1 (P_2 P_3) = P_1 P_0 = P_0$.
- (iii) Identity element exists, P_0 when composed with any permutation gives that permutation.
- (iv) Every permutation has its own inverse.

Hence S_3 is a group. It is a non-commutative group since $P_1 P_2 \neq P_2 P_1$, $P_2 P_3 \neq P_3 P_2$.

Let A be a set of degree n . Let P_n be the set of all permutations of degree n on A . Then $(P_n, *)$ is a group, called a **permutation group** and the operation $*$ is the composition (multiplication) of permutations. This is proved in the following theorem.

Theorem 12.21. The set P_n of all permutation on n symbols is finite group of order $n!$ with respect to the binary composition of permutations. For $n \leq 2$, P_n is abelian and for $n \geq 2$ it is always non-abelian.

Proof Let $X = \{a_1, a_2, a_3, \dots, a_n\}$ is a finite set. Since the different arrangements of the elements of X are $n!$, then the number of distinct permutations of degree n will be $n!$. If P_n is the set of all such permutations, then P_n has $n!$ distinct elements.

Closure Property : Let f and g be any two permutations in P_n where

$$f = \begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ b_1 & b_2 & b_3 & \dots & b_n \end{pmatrix} \text{ and } g = \begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ c_1 & c_2 & c_3 & \dots & c_n \end{pmatrix}$$

are two permutations of degree n . Then,

$$fg = \begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ c_1 & c_2 & c_3 & \dots & c_n \end{pmatrix} \begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ b_1 & b_2 & b_3 & \dots & b_n \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ c_1 & c_2 & c_3 & \dots & c_n \end{pmatrix}$$

Since $c_1, c_2, c_3, \dots, c_n$ are also of arrangement of n elements a_1, a_2, \dots, a_n of X , then fg is a permutation of degree n .

Thus $fg \in P_n$ for all $f, g \in P_n$.

Hence, P_n is closed for the composition known as product of two permutations.

Associativity

$$\text{Let } f = \begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ b_1 & b_2 & b_3 & \dots & b_n \end{pmatrix}, g = \begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ c_1 & c_2 & c_3 & \dots & c_n \end{pmatrix} \text{ and } h = \begin{pmatrix} a_1 & a_2 & a_3 & \dots & a_n \\ d_1 & d_2 & d_3 & \dots & d_n \end{pmatrix}$$

be any three permutations of degree n , then

$$fg = \begin{pmatrix} c_1 & c_2 & \dots & c_n \\ d_1 & d_2 & \dots & d_n \end{pmatrix} \begin{pmatrix} b_1 & b_2 & \dots & b_n \\ c_1 & c_2 & \dots & c_n \end{pmatrix} = \begin{pmatrix} b_1 & b_2 & \dots & b_n \\ d_1 & d_2 & \dots & d_n \end{pmatrix}$$

$$(fg)h = \begin{pmatrix} b_1 & b_2 & \dots & b_n \\ d_1 & d_2 & \dots & d_n \end{pmatrix} \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ d_1 & d_2 & \dots & d_n \end{pmatrix}$$

Also $gh = \begin{pmatrix} b_1 & b_2 & \dots & b_n \\ c_1 & c_2 & \dots & c_n \end{pmatrix} \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ c_1 & c_2 & \dots & c_n \end{pmatrix}$

$$f(gh) = \begin{pmatrix} c_1 & c_2 & \dots & c_n \\ d_1 & d_2 & \dots & d_n \end{pmatrix} \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ c_1 & c_2 & \dots & c_n \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ d_1 & d_2 & \dots & d_n \end{pmatrix}$$

Now from (1) and (2), we get $(fg)h = f(gh)$

Hence, the composition is associative in P_n .

Existence of Identity

The identity permutation of degree n is the identity element of P_n .

Let $f = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \end{pmatrix}$ and $I = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ a_1 & a_2 & \dots & a_n \end{pmatrix}$

Then $fI = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \end{pmatrix} \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ a_1 & a_2 & \dots & a_n \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \end{pmatrix} = f$

Also $If = \begin{pmatrix} b_1 & b_2 & \dots & b_n \\ b_1 & b_2 & \dots & b_n \end{pmatrix} \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \end{pmatrix} = f$

Thus $fI = If = f$

Existence of Inverse

Let $f = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \end{pmatrix}$ be a permutation of degree n then the permutation

$f^{-1} = \begin{pmatrix} b_1 & b_2 & \dots & b_n \\ a_1 & a_2 & \dots & a_n \end{pmatrix}$ is also a permutation of degree n .

Now, $ff^{-1} = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \end{pmatrix} \begin{pmatrix} b_1 & b_2 & \dots & b_n \\ a_1 & a_2 & \dots & a_n \end{pmatrix} = \begin{pmatrix} b_1 & b_2 & \dots & b_n \\ b_1 & b_2 & \dots & b_n \end{pmatrix} = I$

Also, $f^{-1}f = \begin{pmatrix} b_1 & b_2 & \dots & b_n \\ a_1 & a_2 & \dots & a_n \end{pmatrix} \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ a_1 & a_2 & \dots & a_n \end{pmatrix} = I$

Therefore, f^{-1} is the inverse of f .

Therefore $(P_n, *)$ is a group of order $n!$ with respect to composition of permutations. For the set P_n has only one element and for $n = 2$, the number of elements in P_n is 2.

We know that every group of order one or of order two is abelian. Thus $(P_n, *)$ is a group for $n \leq 2$.

For $n > 2$, $(P_n, *)$ is not an abelian group as composition of permutation is not a commutative operation i.e. $fg \neq gf$.

The group $(P_n, *)$ is also called **symmetric group** of degree n and denoted by S_n .

Half Adder

A half adder is a logic circuit that adds 2 bits. Table 3.21 shows the addition of two bits. Columns 1 and 2 of Table 3.12 give the values of two input bits, column 3 gives the sum of these two bits, and column 4 the carry bit. This table is called a *half adder truth table*. Even though the inputs and outputs are binary numbers, they may be taken as depicting truth values of 0 and 1 for developing the boolean expressions for C and S. By inspection of Table 3.20.

$$S = A' \cdot B + A \cdot B' = A \oplus B$$

$$C = A \cdot B$$

Input		Output	
A	B	S (Sum)	C (Carry)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 3.20. A half adder truth table

A logic circuit which uses logic gates to implement the half adder is shown in Fig. 3.55.

It can also be seen from Table 3.21 that the sum of two binary digits can be represented by the output of an XOR gate, and the carry output can be represented by the output of an AND gate, i.e., if same two inputs are applied to XOR and AND gates, the output of XOR gate will represent the sum and the AND gate will represent the carry.

Suppose $A = 1100$ and $B = 1011$. The half adder produces a sum of 1 and carry of 0, then first full adder produces a sum of 1 and a carry of 0, the second full adder produces a sum of 1 and a carry of 0, and the third full adder produces a sum of 0 and a carry of 1. The overall output is 10111.

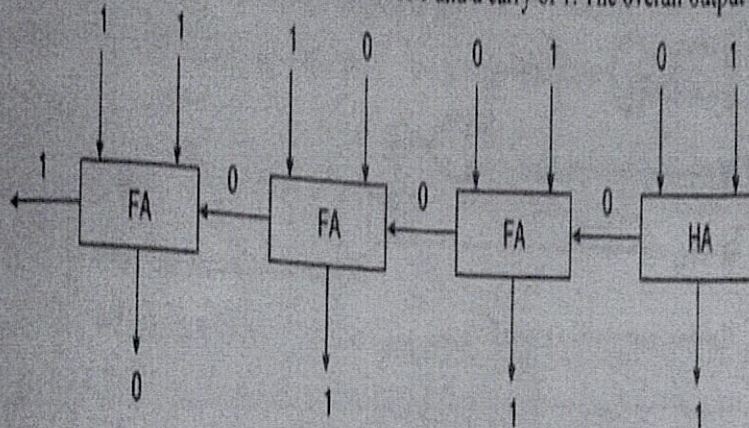


Fig. 3.64. Adding 12 and 11 to get 23

A four bit binary adder can also be built by combining four full adders (Fig. 3.65) the first full adder acts as a half adder as its carry input is held at logic 0 level.

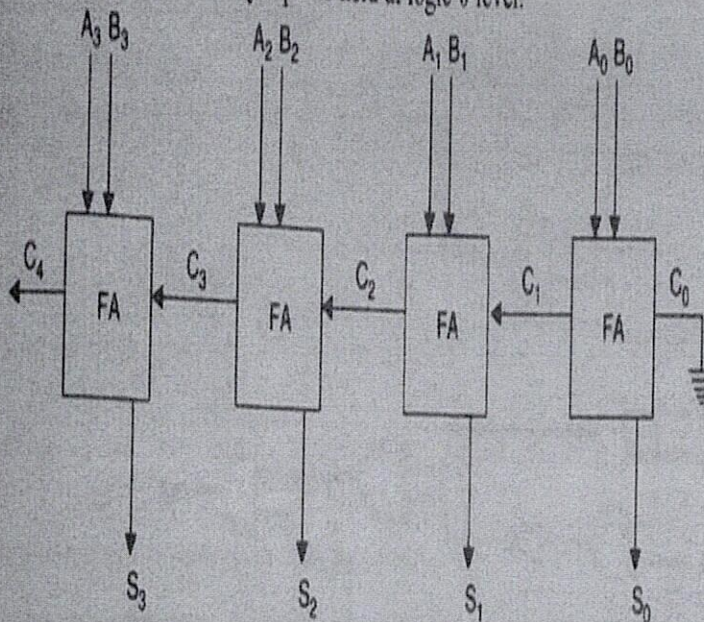


Fig. 3.65. Four-bit parallel adder using four full adders

Served Connectives:-

NAUD:- It means negation of conjunction of two statements. Assume p and q be two propositions. NAUD of p and q is a proposition which is false when both p and q are true otherwise true. It is denoted by $\neg(p \wedge q)$.

p	q	$\neg(p \wedge q)$
T	T	F
T	F	T
F	T	T
F	F	T

NOR:- It means negation of disjunction of two statements. Assume p and q be two propositions. NOR of p and q is a proposition which is true when both p and q are false, otherwise false. It is denoted by $\neg(p \vee q)$.

p	q	$\neg(p \vee q)$
T	T	F
T	F	F
F	T	F
F	F	T

XOR (Exclusive OR): \rightarrow Assume p and q be two propositions.
 The exclusive or (XOR) of p and q , denoted by $p \oplus q$ is a proposition that is true when exactly one of p and q is true but not both and is false otherwise.

Properties of OR

1) $p \oplus q = q \oplus p$ Commutative

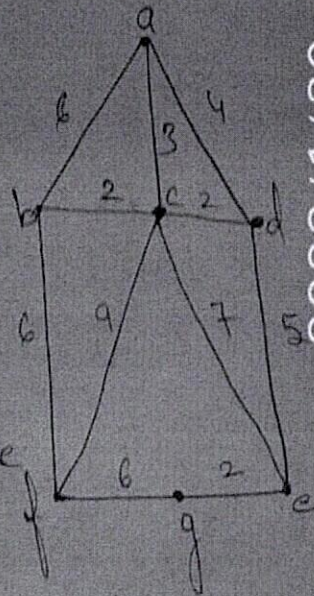
2) $(p \oplus q) \oplus r = p \oplus (q \oplus r)$ Associative

3) $p \wedge (q \oplus r) = (p \wedge q) \oplus (p \wedge r)$ Distributive

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

Travelling - Salesman Problem

The travelling Salesman Problem is simply a Combinational problem based upon the Concept of Hamiltonian Circuit. It states that "A Salesman wants to visit some Cities allotted to him Cyclically. The distance between every pair of Cities is known to him. His problem is to Construct a tour that starts from his home town (city) passing through each city exactly once and return back to the starting point by covering the shortest distance".



Solution →

- 1) Let's start with Vertex a
- 2) Choose the Vertex close to Vertex a, by Calculating the weights of the edges incident from a.

The adjacent vertices are b, c and d

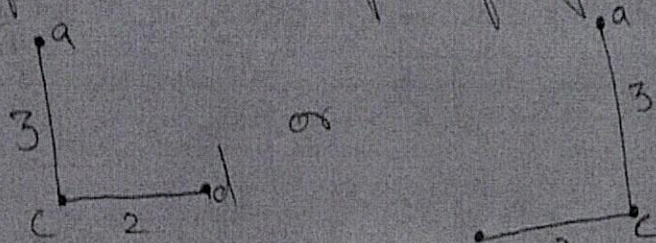
$$w(a,b) = 6, w(a,c) = 3, w(a,d) = 4$$

Min. weight = 3 i.e. between a and c. So next vertex is c.

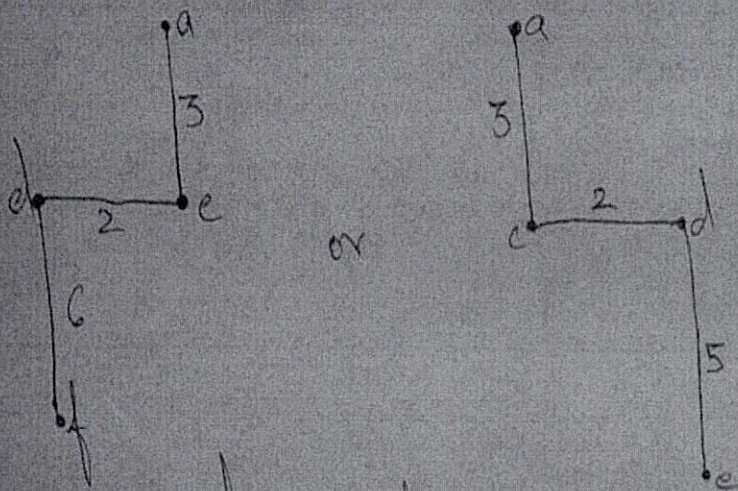
Join a and c to obtain a path of length one.

a

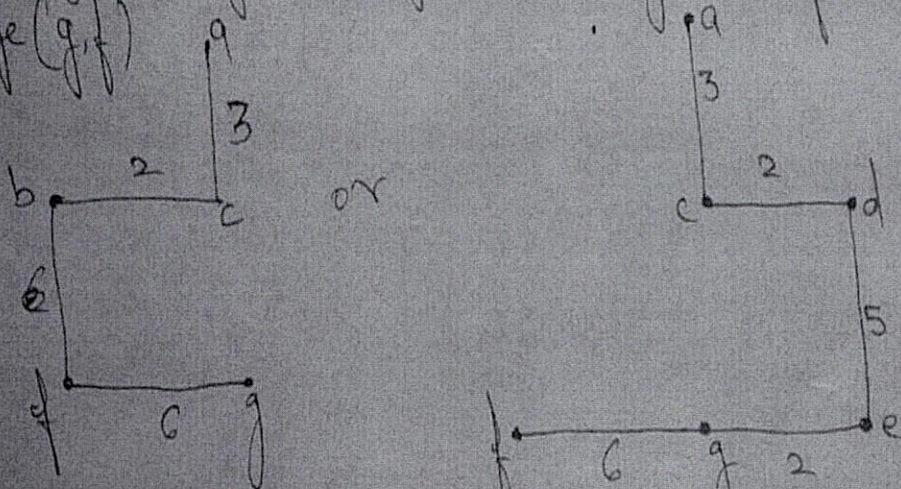
As we can see the weights are equal for both the edges so we can choose one of the edges arbitrary. Let's join the vertex d from c to obtain a path of length 2



Now from vertex d there is a single choice to trace the vertex e through edge (d, e)



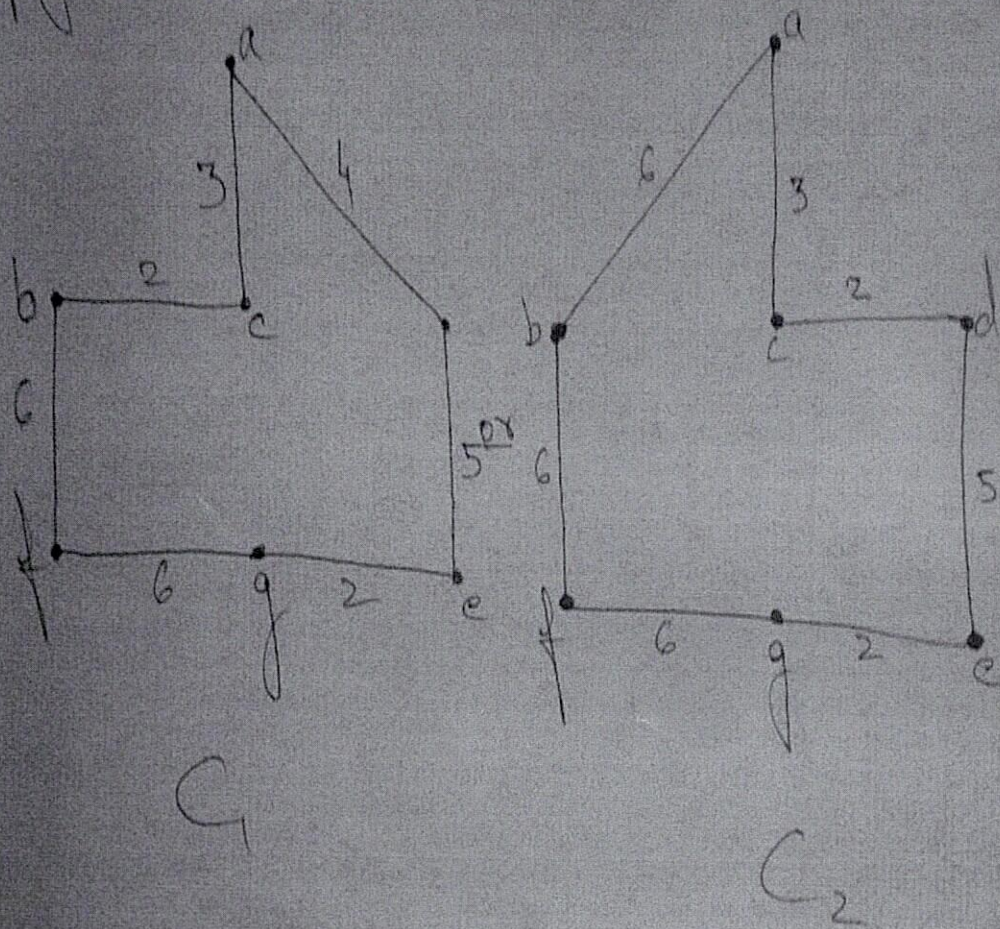
From e again only one edge that is (e, g) and from g only one edge (g, f)



The vertices adjacent to f are b and c minimum

It is clear that the path shown in above fig is the Hamiltonian Path as it includes all the vertices of G . To obtain Hamiltonian Circuit Join the two terminal vertices of the Hamiltonian path and we are done.

The Hamiltonian Circuit Constructed is shown in fig below



Eshan College of Engineering, Farah

Department of _MECHANICAL ENGINEERING

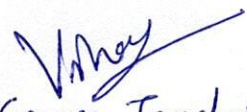
Course Code_KME-302

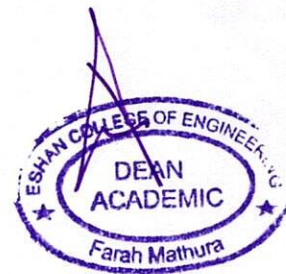
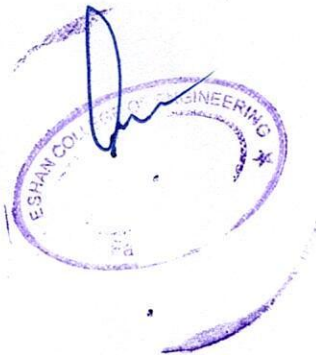
Course Name: B.TECH

Subject: Fluid Mechanics

Contents Beyond Syllabus

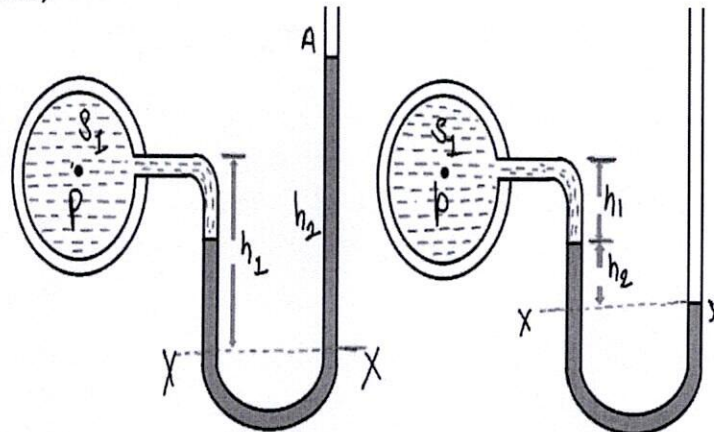
- Measurements of Pressure using Manometers.
- Energy losses in fluids and flow measuring devices.


Course Teacher



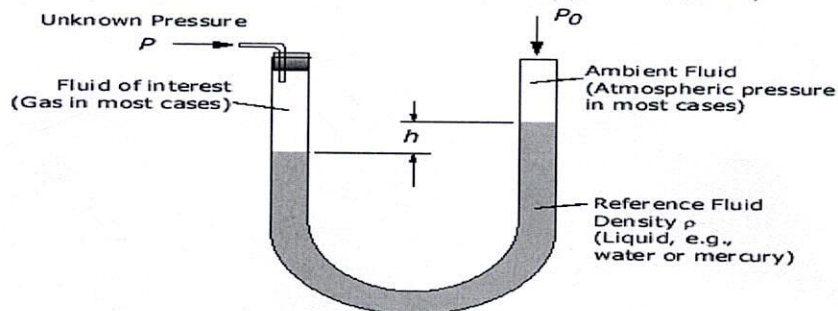
Measurements of Pressure using Manometers

The most common form of a sealed-tube manometer is the conventional mercury barometer used to measure atmospheric pressure. A manometer can be designed to directly measure absolute pressure. The manometer in Figure 5 measures the pressure compared to zero absolute pressure in a sealed leg above a mercury column,

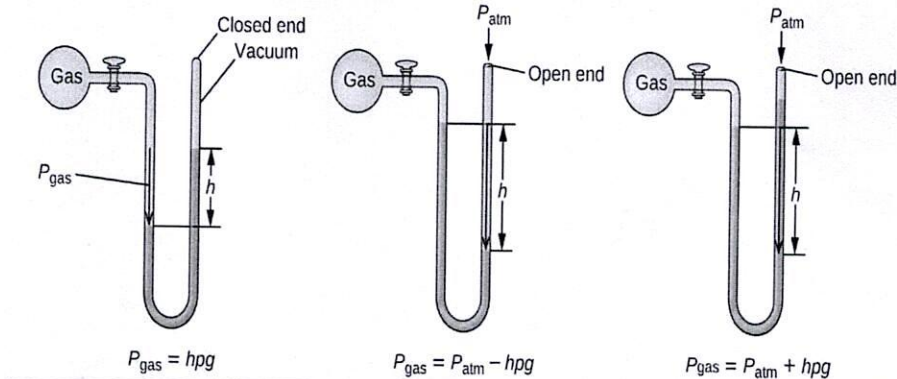


For Gauge Pressure
 $p = g(\rho_2 h_2 - \rho_1 h_1)$

For Vacuum Pressure
 $p = -g(\rho_1 h_1 + \rho_2 h_2)$



$$\text{Gage Pressure } \Delta P = P - P_0 = \rho g h$$



Energy losses in fluids and flow measuring devices

ENERGY LOSS

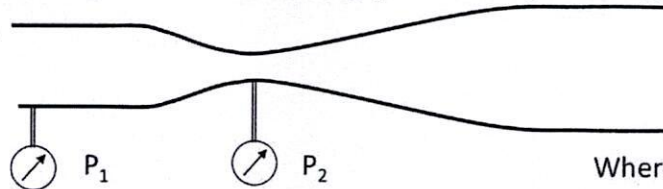
According to the law of conservation of energy, energy balance have to be properly calculated fluids experiences energy losses in several ways while flowing through pipes, they are

- Ø Frictional losses
- Ø Losses in the fitting
- Ø Enlargement losses
- Ø Contraction losses

Flow Measuring Devices

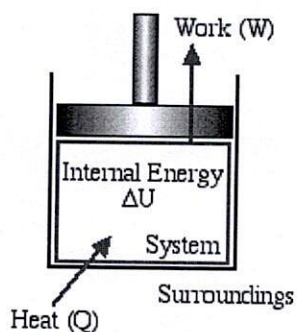
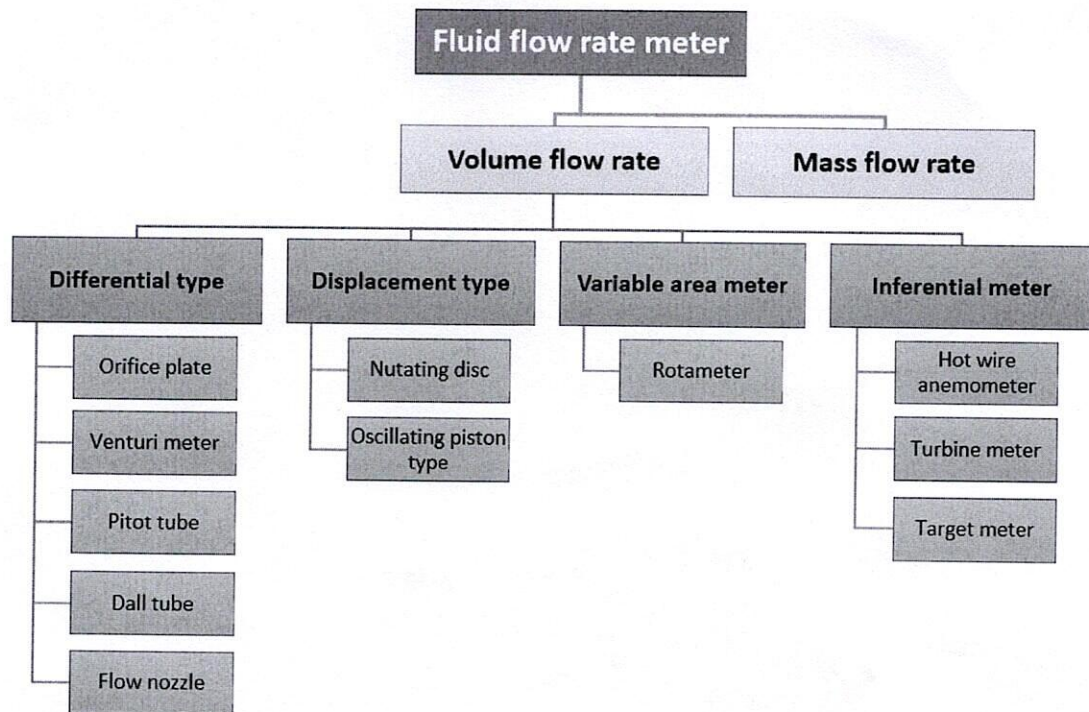
Venturi Meter:

- This device consists of a conical contraction, a short cylindrical throat and a conical expansion.
- The fluid is accelerated by being passed through the converging cone.
- The velocity at the “throat” is assumed to be constant and an average velocity is used.
- The venturi tube is a reliable flow measuring device that causes little pressure drop. It is used widely particularly for large liquid and gas flows.



$$V_2 = C_d \sqrt{\frac{2(P_1 - P_2)}{\rho [1 - (A_2 / A_1)^2]}}$$

Where the discharge coefficient, $C_d = f(Re)$, can be found in Figure shown later¹⁰



Energy Equation for Stationary Closed Systems:

$$Q - W = \Delta U \quad [\text{kJ}]$$

where: Q is the Heat Transferred to the System

W is the Work Done by the System

ΔU is the Change of Internal Energy

Dividing each term by the system mass m [kg] we obtain the specific form of the Energy Equation:

$$q - w = \Delta u \quad [\text{kJ/kg}]$$