INDEX

Unit 1

Classifications of Computer:

1. **Microcomputers (Personal Computer)**

A microcomputer is the smallest general purpose processing system. The older pc started 8 bit processor with speed of 3.7MB and current pc 64 bit processor with speed of 4.66 GB.
Examples: - **IBM PC**s, **APPLE** computers

Microcomputer can be classified into 2 types:

1. Desktops
2. Portables

The difference is portables can be used while travelling whereas desktops computers cannot be carried around.

The different portable computers are: -

1) Laptop
2) Notebooks
3) Palmtop (hand held)

2. **Minicomputer**: - A minicomputer is a medium-sized computer. That is more powerful than a microcomputer. These computers are usually designed to serve multiple users simultaneously (Parallel Processing). They are more expensive than microcomputers. Some small scale industries use these computers.

Examples: Digital Alpha, Sun Ultra, AS400.

3. **Mainframe computers**: - Computers with large storage capacities and very high speed of processing (compared to mini- or microcomputers) are known as mainframe computers. They support a large number of terminals for simultaneous use by a number of users like ATM transactions. They are also used as central host computers in distributed data processing system.

Examples: - **IBM 370, S/390, Z/890.**

4. **Supercomputer**: - Supercomputers have extremely large storage capacity and computing speeds which are many times faster than other computers. A supercomputer is measured in terms of tens of millions Instructions per second (MIPS), an operation is made up of numerous instructions. The supercomputer is mainly used for large scale numerical problems in scientific and engineering disciplines such as Weather analysis.

Examples: - **IBM Deep Blue, PARAM**

Key features of Mainframe Computer:

- Large Number of CPU with Greatest Processing Power (Million Instruction Per second).
- Huge Memory Capacity.
- Increased Performance by sharing workload.
- Centralized Computing.
- Ability to Run in Multiple Operating Systems.
- Supports Time Sharing Ability.
- Supports Sophisticated Operating system.
- Reliability:

Benefits of Mainframe Computer:

- Reliability
- Scalability
- Security
- Redundancy
- Availability
- Compatibility

Reliability:

Reliability is a measurement of a system to continue processing without failure. The "Z" in system Z's brand-name stand for zero down time. System Z servers have reported an "Mean to failure" of 40 years, that means they are guaranteed to run continuously for 40 years without any failure.

Scalability:

The ability of system to expand as resources, such as processors, memory, or storage, are added.

Security:

Modern mainframes have security built into them from the ground up, both in operating system and cryptographic hardware acceleration.

Redundancy:

Redundancy is the use of several identical functional units, such as several disk drives or power supply systems, within one computer system in order to provide data security and certain degree of fault tolerance in case of hardware failures.

Availability:

The recent Mainframe Systems Z series are designed to provide an availability of 99.999%.

Compatibility:

Programs written for IBM OS/360 and later models will still run quiet happily on the latest version of IBM Z/OS.

Evolution of Mainframe:

1) 1960- system 360
    a. 24 bit addressing.
    b. 16 Megabytes addressing capability
    c. Single Processor
2) 1970 – System 370
    a. 24 bit addressing.
    b. Multiple CPUs(Max 2)
    c. Introduced virtual storage.
    d. Exa. 9370.
3) 1983 – System 370/XA (Extended Architecture)
    a. 32 bit addressing. 2 GB addressing capability
    b. True multiprocessing(up to 6 CPUs)
    c. Better I/O subsystems
    d. Exa. 3090.
4) 1990 System 370/ESA
    a. 31 bit capability
    b. LPAR-capability to run multiple OS
    c. Data space to handle larger volumes of data.
5) System 390/ESA
    a. Up to 10 CPUs
    b. ESCON channel
        i. Optical fiber connections
        ii. High data transfer rate over long distances
        iii. Very reliable
    c. Dynamic device configuration
        i. No downtime
        ii. No need to re-start
    d. Exa. ES/900
6) IBM Z architecture
    a. Up to 32-64 CPUs
    b. 64 bit O/S

c. High speed connectivity
d. Dynamic device configuration
e. Exa. Z/890, Z/900

Operating systems on Mainframe:

1) Disk Operating System / 360
    a. Single task(Only one program at a time)
    b. Simple OS, with minimum storage and CPU requirements.
2) OS/360 for System/360
    a. Primary Control Program ran only one task at a time.
    b. Multiple fixed tasks – up to 15 tasks.
    c. Partitions have to be configured before start up.
3) MVS(Multiple Virtual Storage) for System/370
    a. Each user task can have virtually all the memory of 16 MB.
4) MVS/XA(eXtended Architecture) for System370/XA
    a. Each user/task can have virtually all the memory of 2GBs
    b. Concept of data spaces – areas for storing huge file in memory.
5) Z/OS
    a. It is IBMs high end server OS
    b. Highly secure, scalable.
    c. No virus attack.

Batch processing vs. online processing

Batch Processing:

When you use Batch processing, your work is processed in units called jobs. A job may cause one or more programs to execute in sequence. It is a technique by which any task is accomplished in units called jobs.

The necessary information for batch processing is provided through JCL.

It is a non-interactive and offline mode of data processing.

Online Processing: An online system handles transactions when occur and provides output directly to users. Because it is interactive, online processing avoids delays and allows a constant dialog between the user and the system. The system processes transactions completely when and where they occur. Users interact directly with the information system. Users can access data randomly. The information system must be available whenever necessary to support business functions.

Concept of Address Space:

The range of virtual addresses that the operating system assigns to a user or separately running program is called an **address space**. This is the area of contiguous virtual addresses available for executing instructions and storing data.

The range of virtual addresses in an address space starts at zero and can extend to the highest address permitted by the operating system architecture.

Z/OS provides each user with a unique address space and maintains the distinction between the programs and data belonging to each address space. Within each address space, the user can start multiple tasks, using task control blocks or TCBs that allow multiprogramming.

In some ways a z/OS address space is like a UNIX process and the address space identifier (ASID) is like a process ID (PID). Further, TCBs are like UNIX threads in that each operating system supports processing multiple instances of work concurrently.

However, the use of multiple virtual address spaces in Z/OS holds some special advantages. Virtual addressing permits an addressing range that is greater than the central storage capabilities of the system. The use of multiple virtual address spaces provides this virtual addressing capability to each job in the system by assigning each job its own separate virtual address space. The potentially large number of address spaces provides the system with a large virtual addressing capacity.

With multiple virtual address spaces, errors are confined to one address space, except for errors in commonly addressable storage, thus improving system reliability and making error recovery easier. Programs in separate address spaces are protected from each other. Isolating data in its own address space also protects the data.

Z/OS uses many address spaces. There is at least one address space for each job in progress and one address space for each user logged on through TSO, telnet, rlogin or FTP (users logged on Z/OS through a major subsystem, such as CICS or IMS™, are using an address space belonging to the subsystem, not their own address spaces). There are many address spaces for operating system functions, such as operator communication, automation, networking, security, and so on.

The use of address spaces allows Z/OS to maintain the distinction between the programs and data belonging to each address space. The private areas in one user's address space are isolated from the private areas in other address spaces, and this **address space isolation** provides much of the operating system's security.

Yet, each address space also contains a common area that is accessible to every other address space. Because it maps all of the available addresses, an address space includes system code and

data as well as user code and data. Thus, not all of the mapped addresses are available for user code and data.

The ability of many users to share the same resources implies the need to protect users from one another and to protect the operating system itself. Along with such methods as "keys" for protecting central storage and code words for protecting data files and programs, separate address spaces ensure that users' programs and data do not overlap.

Buffer Manager:

IBM MQ for Z/OS uses buffers and buffer pools to temporarily cache data. Use this topic to further understand how buffers are organized, and used.

For efficiency, IBM MQ uses a form of caching whereby messages (and object definitions) are stored temporarily in buffers before being stored in page sets on DASD. Short-lived messages, that is, messages that are retrieved from a queue shortly after they are received, might only ever be stored in the buffers. This caching activity is controlled by a buffer manager, which is a component of IBM MQ.

The buffers are organized into *buffer pools*. You can define up to 16 buffer pools (0 through 15) for each queue manager.

If OPMODE is set to OPMODE=(NEWFUNC, 800), you can define up to 100 buffer pools (0 through 99) for each queue manager.

You are recommended to use the minimal number of buffer pools consistent with the object and message type segregation, and any data isolation requirements your application might have. Each buffer is 4 KB long. Buffer pools use 31 bit storage by default, in this mode, the maximum number of buffers is determined by the amount of 31 bit storage available in the queue manager address space; do not use more than about 70% for buffers. Alternatively, buffer pool storage allocation can be made from 64 bit storage (use the LOCATION attribute of the DEFINE BUFFPOOL command). Using LOCATION(ABOVE) so that 64 bit storage is used has two benefits. Firstly, there is much more 64 bit storage available so buffer pools can be much bigger, and secondly, 31 bit storage is made available for use by other functions. Typically, the more buffers you have, the more efficient the buffering and the better the performance of IBM MQ.

Virtual storage:

Virtual storage is a technique that lets a processor simulate an amount of main storage that is larger than the actual amount of real storage. For example, a processor that has 2M bytes of real storage might use virtual storage to simulate 16M bytes of main storage. To do this, the computer uses disk storage as an extension of real storage.

It is a facility provided to make efficient use of the main storage.

- It is a technique that lets a processor simulate a large amount of main storage from aa smaller installed main storage.
- Implementation: Uses disk storage as an extension of main storage.
- At any point in time, only one program and its data is required in the main storage.

Paging:

Moving pages between real storage frames and auxiliary storage slots is called paging.

Swapping:

Swapping is the process of physically or logically removing a user from central storage. This means moving all of his currently active data and programs to expanded storage or writing them to swap or page data sets (physical swapping) or making them as swapped (logical swapping).

Dataset management in Mainframe:

In the context of IBM mainframe computers, a **data set** (IBM preferred) or **dataset** is a computer file having a record organization. Use of this term began with OS/360 and is still used by its successors, including the current z/OS.

A data set is typically stored on a direct access storage device (DASD) or magnetic tape.

Data sets are not unstructured streams of bytes, but rather are organized in various logical record and block structures determined by the DSORG (data set organization), RECFM (record format), and other parameters. These parameters are specified at the time of the data set allocation (creation), for example with Job Control Language DD statements. Inside a job they are stored in the Data Control Block (DCB), which is a data structure used to access data sets, for example using access method.

PS is single flat files where data can be stored.

Partitioned data sets

A **partitioned data set** (**PDS**) is a data set containing multiple *members*, each of which holds a separate sub-data set, similar to a directory in other types of file systems. This type of data set is often used to hold executable programs (*load modules*), source program libraries (especially Assembler macro definitions), and Job Control Language. A PDS may be compared to a Zip file or COM Structured Storage.

A Partitioned Data Set can only be allocated on a single volume and have a maximum size of 65,535 tracks.

Besides members, a PDS consists also of their directory. Each member can be accessed directly using the directory structure. Once a member is located, the data stored in that member is handled in the same manner as a PS (sequential) data set.

Whenever a member is deleted, the space it occupied is unusable for storing other data. Likewise, if a member is re-written, it is stored in a new spot at the back of the PDS and leaves wasted "dead" space in the middle. The only way to recover "dead" space is to perform frequent file compression, that moves all members to the front of the data space and leaves free usable space at the back. (Note that in modern parlance, this kind of operation might be called defragmentation or garbage collection; data compression nowadays refers to a different, more complicated concept.) PDS files can only reside on disk in order to use the directory structure to access individual members, not on tape. They are most often used for storing multiple JCL files, utility control statements and executable modules.

Unit 2

Virtual storage:

Virtual storage is a technique that lets a processor simulate an amount of main storage that is larger than the actual amount of real storage. For example, a processor that has 2M bytes of real storage might use virtual storage to simulate 16M bytes of main storage. To do this, the computer uses disk storage as an extension of real storage.

It is a facility provided to make efficient use of the main storage.

- It is a technique that lets a processor simulate a large amount of main storage from aa smaller installed main storage.
- Implementation: Uses disk storage as an extension of main storage.
- At any point in time, only one program and its data is required in the main storage.

Paging:

Moving pages between real storage frames and auxiliary storage slots is called paging.

Storage Managers:

DFSMS, XOS System Managed Storage

DFSMS is a policy based storage management system used by IBM mainframes. The purpose of DFSMS is to automate as much as possible the management of physical storage. DFSMS can reduce user concerns about physical details of performance, space, and device management. SMS was designed to:

- Make storage more efficient
- Automate data management
- Allow users to allocate data by service requirements, without needing to know the physical implementation of those requirements
- Improve performance management
- Automate disk space management
- Improve management of data availability
- Simplify data movement

Multiple Virtual System:

Multiple Virtual Storage, more commonly called **MVS**, was the most commonly used operating system on the System/370 and System/390 IBM mainframe computers. It was developed by IBM, but is unrelated to IBM's other mainframe operating systems, e.g., VSE, VM, TPF.

First released in 1974, MVS was extended by program products with new names multiple times:

- first to MVS/SE (MVS/System Extensions),
- next to MVS/SP (MVS/System Product) Version 1,
- next to MVS/XA (MVS/eXtended Architecture),
- next to MVS/ESA (MVS/Enterprise Systems Architecture),
- then to OS/390 and
- finally to z/OS (when 64-bit support was added with the zSeries models).

MVS Address Space:

Address Space	Description
CATALOG	Catalog Address Space – Handles DISP=SHR, DISP=OLD, and DISP=(…,CATLG)
CONSOLE	Console Task – operator's interface
JES2	Job Entry Subsystem
JES2AUX	JES2 Additional Support
ES2MON	JES2 Monitor
RACF	RACF Started Address Space
RMF	Resource Measurement Facility – performance monitoring

SMF	System Management Facility – system usage
SMS	Storage Management Subsystem – handles allocation of DISP=NEW datasets
TCPIP	Internet Protocol (part of VTAM)
TSO	Time Sharing Option
VLF	Virtual Look aside Facility (part of LLA)
WLM	Workload Manager
VTAM	Virtual Telecommunications Access Method – green screens & more. Controls TCPIP; talks to the communication hardware.

Sequential and Partitioned dataset:

Physical Sequential (PS)

PS stands for physical sequential. Data in these files can be read sequentially by a program. A Physical Sequential file, DSORG=PS, is a simple file with records stored in the order that they are written. You can think of a record as a line of text. When you insert a new line of text, you start a new record. PS files are typically used for text and logs, and are so simple that there is little to say about them. Large PS files which are only ever required by one task at a time are very suitable for tape, in fact the only suitable file type for tape is PS. It is possible to improve the performance of PS files by striping them over several volumes, but this is only useful if they are used by several tasks concurrently. Extended format PS files must be SMS managed and can consist of 123 extents on each volume, to maximum of 7,257 extents over 59 volumes.
Large format PS files were introduced in z/OS 1.7 and are not the same as extended format. They do not need to be SMS managed and can grow beyond the 65535 tracks per volume limit for

normal PS files, with a maximum of 16,777,215 tracks per volume. They can only consist of 16 extents per volume, with a maximum of 944 extents over 59 volumes.
You allocate a large format PS dataset using the parameter DSNTYPE=LARGE

Partitioned dataset (PDS)

The minimum size of data allocation under z/OS is 1 track, or 56,664 bytes. This means that a Physical Sequential file which contained three 80 byte records will contain 240 bytes, but occupy 56,664 which is very wasteful. It also means that two datasets cannot exist on the same track. A Partitioned Dataset (PDS) solves this problem by combining a lot of small files into one large container. The individual files are stored as members within the PDS. Each member must have a unique 1-8 character name and the members are located by an index that is called a PDS directory. PDS files have DSORG=PO and DSTYPE=PDS.
A PDS can also be used to collect a set of related files together into a single 'library'.

Direct Access Storage Device:

DASD (Direct access storage device), is a general term for magnetic disk storage devices. The term has historically been used in the mainframe and minicomputer (mid-range computer) environments and is sometimes used to refer to hard disk drives for personal computers. A redundant array of independent disks (RAID) is also a type of DASD.

The "direct access" means that all data can be accessed directly in about the same amount of time rather than having to progress sequentially through the data

Access Methods:

Some the familiar access methods for storage in the mainframe world include:

- BSAM – Basic Sequential Access Method.
- QSAM – Queued Sequential Access Method.
- BDAM – Basic Direct Access Method.
- BPAM – Basic Partitioned Access Method.
- ISAM – Index Sequential Access Method.
- VSAM – Virtual Storage Access Method.

Record Format:

Traditional z/OS data sets have one of five record formats, as follows:
F (Fixed):

Fixed means that one physical block on disk is one logical record and all the blocks and records are the same size. This format is seldom used.

FB (Fixed Blocked):

This format designation means that several logical records are combined into one physical block. This format can provide efficient space utilization and operation. This format is commonly used for fixed-length records.

V (Variable):

This format has one logical record as one physical block. A variable-length logical record consists of a record descriptor word (RDW) followed by the data. The record descriptor word is a 4-byte field describing the record. The first 2 bytes contain the length of the logical record (including the 4-byte RDW). The length can be from 4 to 32,760 bytes. All bits of the third and fourth bytes must be 0, because other values are used for spanned records. This format is seldom used.

VB (Variable Blocked):

This format places several variable-length logical records (each with an RDW) in one physical block. The software must place an additional Block Descriptor Word (BDW) at the beginning of the block, containing the total length of the block.

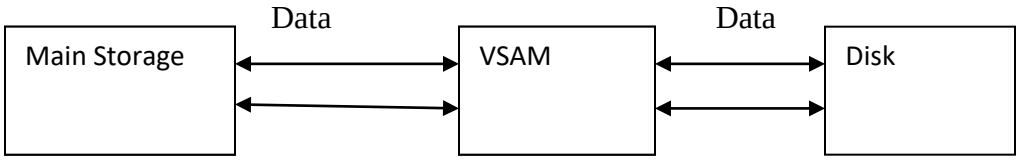
U (Undefined):

This format consists of variable-length physical records and blocks with no predefined structure. Although this format may appear attractive for many unusual applications, it is normally used only for executable modules.

Virtual Storage Access Method (VSAM):

VSAM stands for Virtual Storage Access Method. VSAM is a file storage access method used in MVS, ZOS and OS/390 operating systems. It was introduced by IBM in 1970's. It is a high performance access method used to organize data in form of files in Mainframes. VSAM is used by COBOL and CICS in Mainframes to store and retrieve data. VSAM makes it easier for application programs to execute an input-output operation.

VSAM acts as an interface between Operating System and Application Program.



VSAM provides four types of file organizations along with their respective access methods and utilities. Access methods are system software that provides technical details to system developers.

- Entry Sequential Dataset (ESDS): Is like a standard sequential (QSAM) dataset.
- Relative Record Dataset (RRDS): Is like a direct file (BDAM).
- Key-Sequenced Dataset (KSDS): Is like an indexed sequential access method file.
- Linear Dataset (LDS): With no record organization.

Catalog:

Catalog is a dataset which contains information about other dataset. It provides user with ability to locate the dataset, without knowing where dataset resides. In z/OS, the master *catalog* and user *catalogs* store the locations of data sets. Both disk and tape data sets can be cataloged.

- **The Master Catalog:**

There is no structural difference between a master catalog and a user catalog. What make a master catalog different is how it is used, and what data sets are cataloged in it. Each system has one active master catalog. The master catalog does not have to reside on the system residence volume.

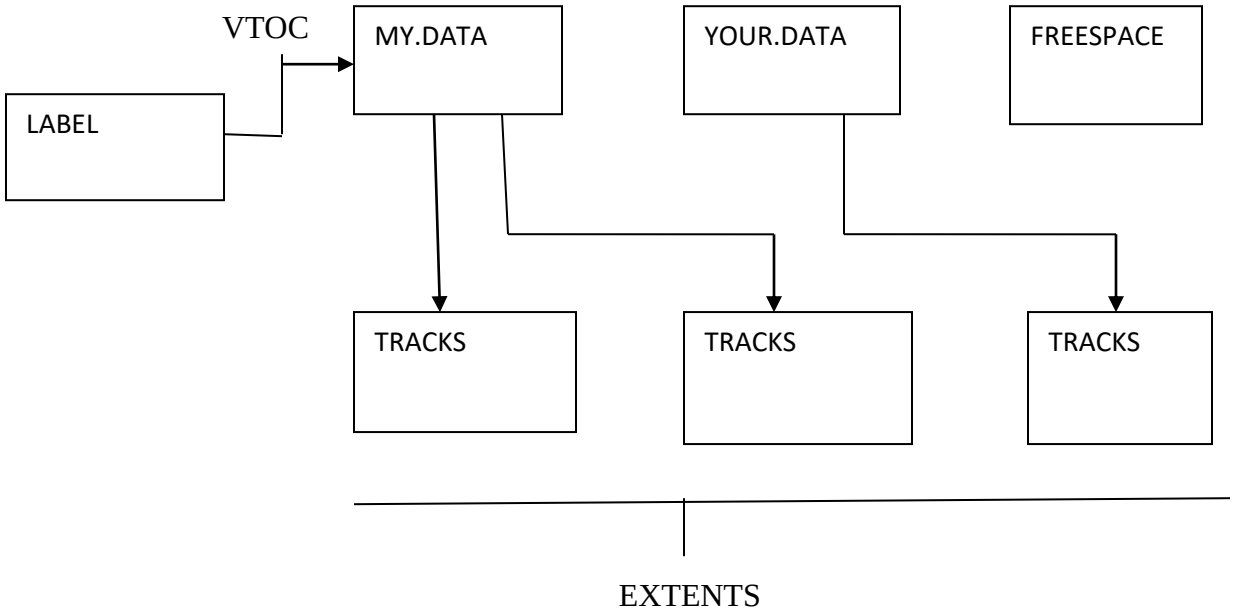
- **User Catalog:**

The user catalogs are pointed to by the Master Catalog
There can be a various amount of user catalogs depending on the installation procedures of your data center.

Volume Table of Content (VTOC):

Z/OS uses a catalog and a volume table of contents (VTOC) on each DASD to manage the storage and placement of data sets.

z/OS requires a particular format for disks, which is shown in Figure 1. Record 1 on the first track of the first cylinder provides the label for the disk. It contains the 6-character volume serial (volser) number and a pointer to the **volume table of contents** (VTOC), which can be located anywhere on the disk.



The VTOC lists the data sets that reside on its volume, along with information about the location and size of each data set, and other data set attributes. A standard z/OS utility program, ICKDSF, is used to create the label and VTOC. When a disk volume is initialized with ICKDSF, the owner can specify the location and size of the VTOC. The size can be quite variable, ranging from a few tracks to perhaps 100 tracks, depending on the expected use of the volume. More data sets on the disk volume require more space in the VTOC.

The VTOC also has entries for all the free space on the volume. Allocating space for a data set causes system routines to examine the free space records, update them, and create a new VTOC entry. Data sets are always an integral number of tracks (or cylinders) and start at the beginning of a track (or cylinder).

You can also create a VTOC with an index. The VTOC index is actually a data set with the name SYS1.VTOCIX.volser, which has entries arranged alphabetically by data set name with pointers to the VTOC entries. It also has bitmaps of the free space on the volume. A VTOC index allows the user to find the data set much faster.

Unit 3

Introduction to JOB control language:-

JCL:

Job Control Language is a set of control statements that provides specification to process a job.

There are 3 types of JCL statements- JOB, EXEC, DD

JCL Format:

Identifier [name] [operation] [operand] [comments]

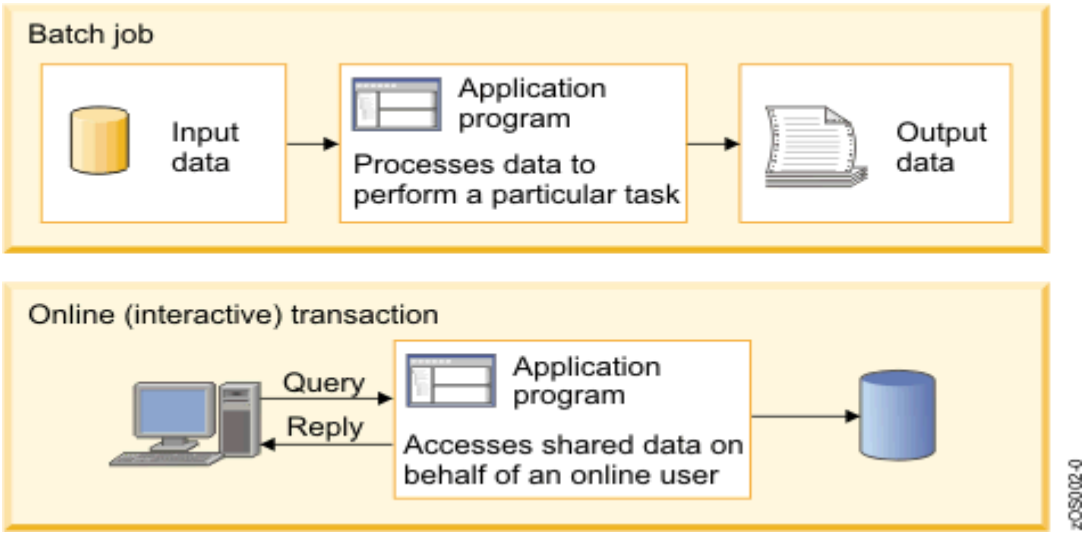
JOB processing:

Most mainframe workloads fall into one of two categories: Batch processing or online transaction processing, which includes Web-based applications.

One key advantage of mainframe systems is their ability to process terabytes of data from high-speed storage devices and produce valuable output. For example, mainframe systems make it possible for banks and other financial institutions to perform end-of-quarter processing and produce reports that are necessary to customers (for example, quarterly stock statements or pension statements) or to the government (for example, financial results). With mainframe systems, retail stores can generate and consolidate nightly sales reports for review by regional sales managers. The applications that produce these statements are ***batch* applications**, which are illustrated at the top of Figure 1.

In contrast **to batch processing**, **transaction processing** occurs interactively with the end user. This interaction is outlined at the bottom of Figure 1. Typically, mainframes serve a vast number of ***transaction systems***. These systems are often mission-critical applications that businesses depend on for their core functions. Transaction systems must be able to support an unpredictable number of concurrent users and transaction types. Most transactions are executed in short time periods— fractions of a second in some cases.

Figure 1. Typical mainframe workloads



Structure of JCL statement:

JCL structure defines the predefined way of JCL creation.

JOB
|
|
EXEC
|
|
DD

JCL consists of 3 statements -

JOB – Identifies a JOB and supplies accounting information.
EXEC – Identifies a JOB step by indicating the name of program to be executed.
DD (Data Descriptor) – Identifies a dataset to be collected for the JOB step.

Normally JCL structure mainly contains two major parts.
1. JOB card/JOB statement.
2. JOB body/JOB steps.

JOB card:

- JOB card is used to identify the Job or the unit of work the operating system to perform.
- JOB card is the first statement to the JOB.
- JOB card should always be coded as a first statement in the JOB.
- JOB card should be coded only once in the specific JOB.
- JCL language requires JOB card to identify the particular JOB.JOB card started with slashes ("//") in the first two positions.
- JOB card has the accounting information and JOB related information.
- JOB card has positional parameters and keyword parameters.

JOB body:

- Except the JOB card, remaining part of the JOB is JOB body.
- JOB body contains set of steps.
- Each step contains one EXEC statement and/or one or more DD statements.
- EXEC statement is mandatory for the particular step of the JOB body.
- A step can contain up to 255 DD statements.

Every step in JOB body can be divided into two parts.
1. EXEC.
2.DD.

Various statements in JCL:

JOB statements:

Statement	Name	Purpose
// JOB	job	Marks the beginning of a job; assigns a name to the job.

Syntax:

Following is the basic syntax of a JCL JOB statement:

//Job-name JOB Positional-param, Keyword-param

//	Name	Operation	Operands	Comments
//	JOBname	JOB	<positional/Keyword parameters>	Comments

JOB card parameters can be divided into two types based on their behavior.

1. Positional parameter.
2. Keyword parameter.

Positional parameter:
a. Account information.
b. Additional accounting information.

Keyword parameter:

a. CLASS.
b. MSGCLASS (Message class).
c. MSGLEVEL (Message level).
d. PRTY (Priority).
e. RESTART.
f. TYPERUN.
g. NOTIFY.
h. TIME.
i. REGION.
j. COND (condition).

Keyword parameter should follow positional parameter.

EXEC statement:

Statement	Name	Purpose
// EXEC	execute	Marks the beginning of a job step; assigns a name to the step; identifies the program or the cataloged or in-stream procedure to be executed in this step.

Syntax

Following is the basic syntax of a JCL EXEC statement:

//Step-name EXEC Positional-param, Keyword-param

//	Name	Operation	Operands	Comments
//	Stepname	EXEC	<Prog/Proc>	Comments

Description
Let us see the description of the terms used in above EXEC statement syntax.

STEP-NAME
This identifies the job step within the JCL. It can be of length 1 to 8 with alphanumeric characters.

EXEC
This is the keyword to identify it as an EXEC statement.

POSITIONAL-PARAM
These are positional parameters, which can be of two types:

Positional Parameter	Description
PGM	This refers to the program name to be executed in the job step.
PROC	This refers to the procedure name to be executed in the job step.

KEYWORD-PARAM
Following are the various keyword parameters for EXEC statement. You can use one or more parameters based on requirements and they are separated by comma:

Keyword Parameter	Description
PARM	Used to provide parametrized data to the program that is being executed in the job step.
ADDRSPC	This is used to specify whether the job step require virtual or real storage for execution.
ACCT	This specifies the accounting information of the job step. Following is the syntax: **ACCT=(userid)**

DD statement:

Datasets are mainframe files with records organized in a specific format. Datasets are stored on the Direct Access Storage Device (DASD) or Tapes of the mainframe.
The definition of each dataset used in the JCL is given using the **DD statement**.

Syntax

Following is the basic syntax of a JCL DD statement:

//DD-name DD Parameters

//	Name	Operation	Operands	Comments
//	ddname	DD	<Positional/Keyword parameters>	Comments

Description

DD-NAME
A DD-NAME identifies the dataset or input/output resource. If this is an input/output file used by a COBOL/Assembler program, then the file is referenced by this name within the program.

DD
This is the keyword to identify it as an DD statement.

PARAMETERS
Following are the various parameters for DD statement. You can use one or more parameters based on requirements and they are separated by comma:

Parameter	Description
DSN	The DSN parameter refers to the physical dataset name of a newly created or existing dataset. Following is the syntax: **DSN=Physical Dataset Name**
DISP	The DISP parameter is used to describe the status of the dataset, disposition at the end of the job step on normal and abnormal completion. Following is the syntax: **DISP=(status, normal-disposition,**

	abnormal-disposition) Following are valid values for **status**: • **NEW** : The dataset is newly created by the job step. OUTPUT1 in the example above. • **OLD** : The dataset is already created and will be overwritten in the job step. The job step gains exclusive access on the dataset and no other job can access this dataset until the completion of the job step. • **SHR** : The dataset is already created and will be read in the job step. The dataset can be read by multiple jobs at the same time. Example: INPUT1 and INPUT2. • **MOD** : The dataset is already created. This disposition will be used when there is a need to append new records to the existing dataset (existing records will not be overwritten). A **normal-disposition** parameter can take one of the following values • CATLG, UNCATLG, DELETE, PASS and KEEP A **abnormal-disposition** parameter can take one of the following values • CATLG, UNCATLG, DELETE and KEEP Here is the description of CATLG, UNCATLG, DELETE, PASS and KEEP parameters: • **CATLG** : The dataset is retained with a entry in the system catalog. • **UNCATLG** : The dataset is retained but system catalog entry is removed. • **KEEP** : The dataset is retained without changing any of the catalog entries.

	KEEP is the only valid disposition for VSAM files. This is to be used only for permanent datasets. • **DELETE** : Dataset is deleted from user and system catalog. • **PASS** : This is valid only for normal disposition. This is used when the dataset is to be passed and processed by the next job step in a JCL When any of the sub-parameters of DISP are not specified, the default values are as follows: • **status** : NEW is the default value. • **normal-disposition** : If status is NEW, default normal-disposition is DELETE, else it is KEEP. • **abnormal-disposition** : Same as normal disposition.
DCB	The Data Control Block (DCB) parameter details the physical characteristics of a dataset. This parameter is required for datasets that are newly created in the job step. LRECL is the length of each record held within the dataset. RECFM is the record format of the dataset. RECFM can hold values FB, V or VB. FB is a fixed block organization where one or more logical records are grouped within a single block. V is variable organization where one variable length logical record is placed within one physical block. VB is Variable Block organization where one or more variable length logical records are placed within one physical block. BLKSIZE is the size of the physical block. The larger the block, greater is the number of records for a FB or VB file. DSORG is the type of dataset organisation.

	DSORG can hold values PS (Physical Sequential), PO (Partitioned Organisation) and DA (Direct Organisation).
SPACE	The SPACE parameter specifies the space required for the dataset in the DASD (Direct Access Storage Disk). Following is the syntax: **SPACE=(spcunits, (pri, sec, dir), RLSE)** Here is the description of all the used parameters: • **spcunits** : This can be one of the CYL(Cylinder), TRK(Tracks) or BLKSIZE(Block Size). • **pri** : This is the primary space required for the dataset. • **sec** : This is the additional space required, when the primary space is not being sufficient. • **dir** : This is the directory blocks required, if the dataset is a PDS (Partitioned Dataset) with members within it. • **RLSE** : This is used to release the unused space at job completion.
UNIT	The UNIT and VOL parameters are listed in the system catalog for catalogued datasets and hence can be accessed with just the physical DSN name. Following is the syntax: **UNIT=DASD | SYSDA** Where DASD stands for Direct Access Storage Device and SYSDA stands for System Direct Access and refers to the next available disk storage device.
VOL	The VOL parameter specifies the volume number on the device identified by the UNIT

	parameter. Following is the syntax: **VOL=SER=(v1,v2)** Where v1, v2 are volume serial numbers.

JCL Procedures:

The **JCL Procedures** are set of statements inside a JCL grouped together to perform a particular function. Usually, the fixed part of the JCL is coded in a procedure. The varying part of the Job is coded within the JCL.

Syntax

Following is the basic syntax of a JCL procedure definition:

//*
//*Step-name* **EXEC** *procedure name*

Instream Procedure

When the procedure is coded within the same JCL member, it is called an Instream Procedure. It should start with a PROC statement and end with a PEND statement.

Cataloged Procedure

When the procedure is separated out from the JCL and coded in a different data store, it is called a **Cataloged Procedure**. A PROC statement is not mandatory to be coded in a cataloged procedure

Nested Procedures

Calling a procedure from within a procedure is called a **nested procedure**. Procedures can be nested up to 15 levels. The nesting can be completely in-stream or cataloged. We cannot code an instream procedure within a cataloged procedure.

IBM Utility Programs:

Utility programs are pre-written programs, widely used in mainframes by system programmers and application developers to achieve day-to-day requirements, organising and maintaining data.

Dataset Utility Programs:

- IEFBR14
- IEBGENER
- IEBCOPY
- IEBCOMPR
- IEBEDIT

System Utility Programs:

- IIEHLIST
- IEHMOVE
- IEHPROGM
- IEHINITT

Utility Name	Functionality
IEFBR14	It is a dummy utility to create sequential dataset (PS) and partitioned dataset (PDS).
IEBGENER	Copy one sequential file data(PS) to another sequential file data(PS)
IEBCOPY	a. Copy one Partitioned Dataset file(PDS) to another PDS. b. Compress the dataset. c. Include the member. d. Exclude the member.
IEBCOMPR	Compares data in sequential datasets.
IDCAMS	Create, delete, rename, catalog, uncatalog datasets (other than PDS). Usually used to manage VSAM datasets.
IEHMOVE	Moves or copies sequential datasets.
IEHPROGM	Deleting and renaming datasets; catalog or uncatalog datasets other than VSAM.

IEFBR14:Example:

```
//KDKCE01A JOB (ACCT01) ,'A.KUNDU',CLASS=A,
// MSGCLASS=A,MSGLEVEL=(1,1),TIME=1440
//STEP1 EXEC PGM=IEFBR14
//DD1 DD DSN=KDKCE01A.JCL.PS,DISP=(NEW,CATLG,DELETE),
// DCB=(DSORG=PS,RECFM=FB,LRECL=80,BLKSIZE=800),
// SPACE=(TRK,(3,2)),UNIT=SYSDA
//SYSPRINT DD SYSOUT=*
```

```
//SYSOUT DD SYSOUT=*
//SYSIN DD DUMMY
//
```

IEBGENER: Example:

```
//KDKCE01A JOB (ACCT01) ,'A.KUNDU',CLASS=A,
// MSGCLASS=A,MSGLEVEL=(1,1),TIME=1440
//STEP1 EXEC PGM= IEBGENER
//SYSUT1  DD DSN=KDKCE01A.XYZ.PS,DISP=SHR
// SYSUT2  DD DSN=KDKCE01A.WXY.PS,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD DUMMY
//
```

IEBCOPY: Example:

```
//KDKCE01A JOB (ACCT01) ,'A.KUNDU',CLASS=A,
// MSGCLASS=A,MSGLEVEL=(1,1),TIME=1440
//STEP1 EXEC PGM= IEBcopy
//SYSUT1  DD DSN=KDKCE01A.XYZ.PDS,DISP=SHR
// SYSUT2 DD DSN=KDKCE01A.WXY.PDS,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSIN DD *
    COPY INDD=SYSUT1,
         OUTDD=SYSUT2
    SELECT MEMBER=(M1,M2)
    EXCLUDE MEMBER=(M3,M4)


/*
//
```

UNIT 4

History of COBOL

• COBOL (Common Business Oriented Language) was one of the earliest high-level programming languages. · COBOL was developed in 1959 by the Conference on Data Systems Languages (CODASYL). This committee was a formed by a joint effort of industry, major universities, and the United States Government. This committee completed the specifications for COBOL as the year of 1959 came to an end. These were then approved by the Executive Committee in January 1960, and sent to the government printing office, which edited and printed these specifications as Cobol60. COBOL was developed within a six month period, and yet is still in use over 40 years later. · Since 1960, the American National Standards Institute (ANSI) was responsible for developing new COBOL standards.

• Three ANSI standards for COBOL have been produced: in 1968, 1974 and 1985.

• Object-oriented COBOL is the fourth edition in the continuing evolution of ANSI/ISO standard COBOL. · The government contributed to COBOL's initial popularity by insisting that computers sold or leased to the government had to have COBOL software available.

Evolution of COBOL

During 1950s, when the businesses were growing in the western part of the world, there was a need to automate various processes for ease of operation and this gave birth to a high-level programming language meant for business data processing.

- In 1959, COBOL was developed by CODASYL (Conference on Data Systems Language).
- The next version, COBOL-61, was released in 1961 with some revisions.
- In 1968, COBOL was approved by ANSI as a standard language for commercial use (COBOL-68).
- It was again revised in 1974 and 1985 to develop subsequent versions named COBOL-74 and COBOL-85 respectively.
- In 2002, Object-Oriented COBOL was released, which could use encapsulated objects as a normal part of COBOL programming.

Features of COBOL

Standard Language

COBOL is a standard language that can be compiled and executed on machines such as IBM AS/400, personal computers, etc.

Business Oriented

COBOL was designed for business-oriented applications related to financial domain, defense domain, etc. It can handle huge volumes of data because of its advanced file handling capabilities.

Robust Language

COBOL is a robust language as its numerous debugging and testing tools are available for almost all computer platforms.

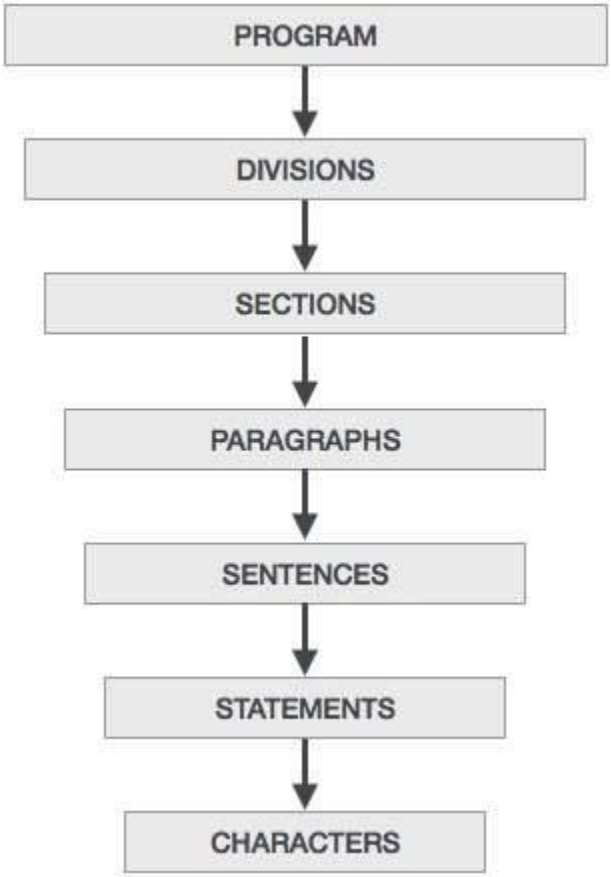
Structured Language

Logical control structures are available in COBOL which makes it easier to read and modify. COBOL has different divisions, so it is easy to debug.

Why COBOL is called Common Business Oriented Language? W-12 (3M)

- COBOL is a language for writing programs for execution on computers. It provides concepts and facilities well-suited for business applications.
- COBOL is an acronym for Common Business Oriented Language.
- A COBOL is a higher level language. Thus a COBOL program can be transferred to another computer with only marginal changes.
- It provides a powerful framework for data management and processing.
- Programs written in COBOL are readable due to the English like form of COBOL.

COBOL program structure **S-13(5M) W-8(7M)**



A brief introduction of these divisions is given below –

- **Sections** are the logical subdivision of program logic. A section is a collection of paragraphs.
- **Paragraphs** are the subdivision of a section or division. It is either a user-defined or a predefined name followed by a period, and consists of zero or more sentences/entries.
- **Sentences** are the combination of one or more statements. Sentences appear only in the Procedure division. A sentence must end with a period.
- **Statements** are meaningful COBOL statements that perform some processing.
- **Characters** are the lowest in the hierarchy and cannot be divisible.

There are 4 main divisions and each division provides an essential part of the information required by the complier. At the top of the COBOL hierarchy are the four divisions. The sequence in which they are specified is fixed, and must follow the order:

- **IDENTIFICATION DIVISION** supplies information about the program to the programmer and the compiler. This division consists of number of standard paragraphs showing the name of the program, name of its author, date on which the program is compiled.
- **ENVIRONMENT DIVISION** is used to describe the environment in which the program will run. This division consist of two sections:
    a. **CONFIGURATION SECTION.**
    b. **INPUT-OUTPUT SECTION.**
  - **CONFIGURATION SECTION:**
    This section describes the particular computer used to compile and execute the COBOL program.
  - **INPUT-OUTPUT SECTION:**
    This section describes various peripheral devices used by the programs.
- **DATA DIVISION** provides descriptions of the data-items processed by the program. This division consists of two sections:
    a. **FILE SECTION.**
    b. **WORKING STORAGE SECTION.**
  - **FILE SECTION:**
    This section includes the description of all data items that should be read from or written onto some external file.
  - **WORKING STORAGE SECTION:**
    The data items which are developed internally as intermediate results as well as constants are described in this section.
- **PROCEDURE DIVISION** contains the code used to manipulate the data described in the DATA DIVISION.

HelloWorld Example

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLOWORLD.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. RM-COBOL.
OBJECT-COMPUTER. RM-COBOL.
DATA DIVISION.
FILE SECTION.
PROCEDURE DIVISION.
MAIN-LOGIC SECTION.
DISPLAY "Hello world!"
STOP RUN.
```

LANGUAGE FUNDAMENTALS.

COBOL coding sheet W-7(3M) S-8(4m)

There are 80 character positions on each line of the coding sheet and these positions are grouped into following fields:

Position	Field
1-6	Sequence
7	Indicator
8-11	Area A
12-72	Area B
73-80	Identification

This area is divided into two fields such as area A and area B. COBOL requires that some of the entries must begin in are A and others must be confined to area B only.

The sequence field may be used to assign sequence numbers to the coding lines.

Page number is written in first three positions and line numbers Aare in next three positions.

Anything written in the Identification field will be ignored by the compiler.

The indicator field may contain an asterisk, a slash, or a hyphen.

Explain IDENTIFICATION DIVISION. S-13(3m)

The IDENTIFICATION DIVISION is the first division of every COBOL source program.
The paragraph PROGRAM-ID is essential in most of the machines. The other paragraphs are optional.
The structure of IDENTIFICATION DIVISION is given below.

IDENTIFICATION DIVISION
PROGRAM-ID. Entry
[AUTHOR . entry]
[INSTALLATION. Entry]
[DATE-WRITTEN. Entry]
[DATE-COMPILED. Entry]

The division heading and paragraph names should be coded as margin A entries.

PROGRAM-ID:

The entry in the PROGRAM-ID paragraph contains the program name to be used to identify the object program.

AUTHOR:

The entry for the AUTHOR paragraph may include the name of the programmer.

DATE-WRITTEN:

The entry of the DATE-WRITTEN paragraph may contain the date of written of program.

DATE-COMPILED:

The entry of the DATE-COMPLED paragraph may contain the date of compilation.

Explain ENVIRONMENT DIVISION. S-9(2M) W-9(5M)

The ENVIRONMENT DIVISION is the second division in a COBOL source program. It is the most machine-dependant division. The computer and all peripheral devices required by the program are described in this division.

This division contains two sections

1. CONFIGURATION SECTION
2. INPUT-OUTPUT SECTION.

The outline of the section and paragraph of this division is shown below.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION
SOURCE-COMPUTER. Entry
OBJECT COMPUTER. Entry
INPUT-OUTPUT SECTION.
FILE CONTROL. {file-control-entry}
[I-O-CONTROL. Input-output-control-entry]

CONFIGURATION SECTION

This section contains an overall specification of the computer used for the purpose of compilation and execution of the program.

SOURCE-COMPUTER

This paragraph specifies the name of the computer used to compile the COBOL program.
SOURCE COMPUTER. Computer name

OBJECT COMPUTER

The OBJECT COMPUTER paragraph describes the computer on which the program is to be executed.
OBJECT COMPUTER. Computer name

INPUT-OUTPUT SECTION

This section contains information regarding files to be used in the program. There are two paragraphs in this section-FILE-CONTROL and I-O-CONTROL.

FILE-CONTROL

The FILE-CONTROL paragraph names each file and identifies the first medium through file control entries. The simplified format of a file control entry is given below.
SELECT file-name ASSIGN TO hardware-name.

Explain DATA DIVISION. W-6(8M) S-11(5M) S-7(9M) S-9(2M)

The DATA DIVISION is that part of a COBOL program where every data item processed by the program is described.
The DATA DIVISION is divided into a number of sections such as File Section, Working-storage section, screen section, Linkage section, and Report section.

a. **FILE SECTION**
   The FILE SECTION includes the description of all data items that should be read from or written onto some external file.
b. **WORKING-STORAGE SECTION**
   The data items which are developed internally as intermediate result as well as the constants are described in this section of the DATA DIVISION.

- **The format of the DATA DIVISION is as follows**
  DATA DIVISION.
  [FILE SECTION.
  file section entries.]
  [WORKING-STORAGE SECTION.
  working storage entries]