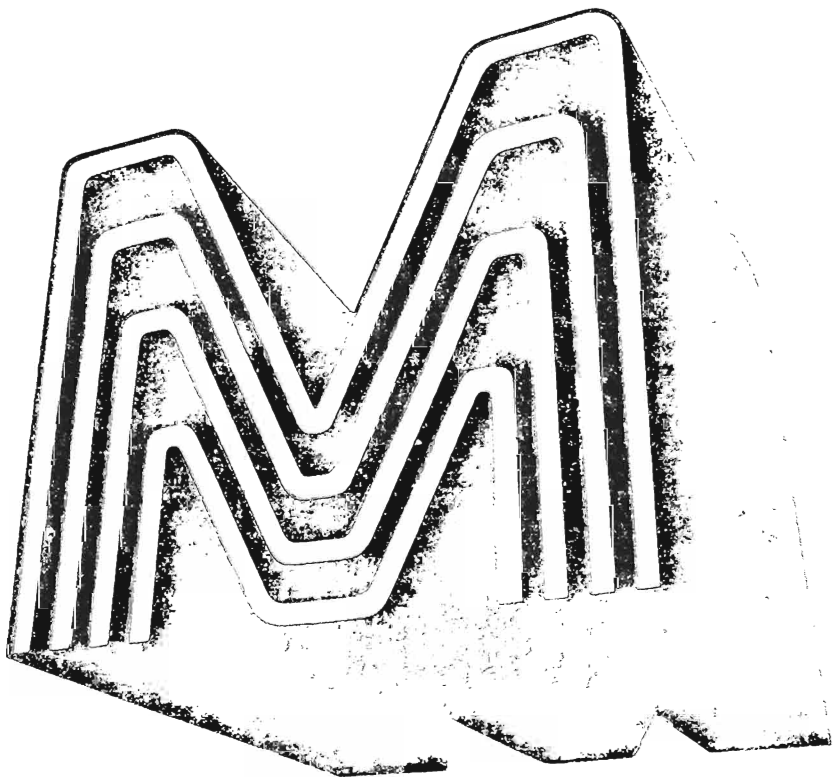
64SG-1000E-1

FACOM OS IV/F4

GENERAL DESCRIPTION



FUJITSU

First Edition December, 1976

PREFACE

This publication describes the FACOM OS IV/F4 operating system for the M series of computers, which are manufactured and sold by Fujitsu Limited. OS IV/F4 is the largest OS furnished for M series in terms of sizes of hardware configurations it supports, variety of functions it furnishes to users and installation managers, and its complexity and state-of-art design. OS IV/F4 compares favorably with state-of-art OS offered by other manufacturers for their large-scale general-purpose computers.

This **General Description** manual is prerequisite reading for all users and operators of OS IV/F4. It furnishes a comprehensive but rapid overview of the entire software system, although it does not describe M series hardware. The user should become familiar with various hardware manuals appropriate to his configuration.

Throughout this manual, reference is made to other OS IV/F4 manuals. Some are recommended for all users; others are required reading for specific programming languages, I/O devices, and application areas; others are appropriate primarily for operators, system programmers, etc. These references should prove useful to readers of the **General Description** manual who need full details on these aspects of M series hardware and OS IV/F4 software.

This manual is divided into four self-contained parts, each with its own set of chapters, figures, and tables. These parts do not cross-reference one another; hence, all cross-references to chapters, sections (of chapters), figures, and tables are within the current part unless explicitly directed to another part of this manual.

Part 1 Overview of OS IV/F4
Part 2 Control Program
Part 3 Processing Programs for OS IV/F4
Part 4 TSS(Time Sharing System)

CONTENTS

ILLUSTRATIONS

PART 3 PROCESSING PROGRAMS

Table No. **Title** **Page**

PART 3 PROCESSING PROGRAMS

PART 4 TSS (Time Sharing System)

PART 1
OVERVIEW OF OS IV/F4

CHAPTER 1
OBJECTIVES OF OS IV/F4

During the past three decades, general-purpose digital computers have been developed rapidly and utilized in increasingly varied applications. Batch processing (including remotely-submitted jobs) and online systems (such as for time-sharing and inquiry) are now utilized worldwide.

Most enterprises need not only more data processing but also more accurate data processing. Computational algorithms must be improved for accuracy and efficiency. Programming languages and application program generators need improved flexibility and relevance to the applications areas where they are used: banking, insurance, manufacturing, distribution, education, military services, government, etc.

OS IV/F4 is one of the most up-to-date and full-function operating systems ever implemented. It furnishes highly-efficient local and remote batch processing. It offers a time sharing system (TSS) with a maximum set of language and command facilities as well as excellent responsiveness.

OS IV/F4 offers a wide range of compilers for the most common programming languages: Assembler, COBOL, FORTRAN, PL/I, and ALGOL. Some compilers are designed for program-development usage, with special attention to convenient entry, editing, compilation and debugging facilities. Other compilers are designed to operate in a batch environment and produce highly-optimized object programs.

OS IV/F4 provides a full range of service programs: a Linkage Editor, a Loader, a Sort/Merge package, a collection of system utility programs for managing system data sets, and a variety of data set utility programs for such commonplace needs as media transcriptions, listing the contents of tape reels and disk packs, and generating and editing new data sets.

OS IV/F4 furnishes an advanced information management system (AIM) to provide state-of-art data base and data communications facilities. AIM accepts, organizes, stores, and presents information which may be highly structured—accessed in a variety of modes, using several different identifiers and/or aggregated at several levels of detail. AIM furnishes an online communications subsystem so that users of typewriters and display terminals can directly access AIM data bases. AIM facilitates convenient design and implementation of data bases by computer professionals, while providing simple, rapid access to data for non-professional users.

CHAPTER 2
MAJOR FEATURES OF FACOM OS IV/F4

OS IV/F4 furnishes certain features which are unique (among comparably sized operating systems) or at least at the state-of-art in terms of their efficiency, user convenience, or level of design:

- high reliability—in conjunction with M series hardware.
- useful facilities—in terms of ease of operation and installation management.
- efficient usage of system resources—delivering maximum CPU, memory, and I/O resources to user programs, and imposing minimal OS overheads.
- expansibility—without traumatic hardware/software changes.
- convenient interfaces and tools for developing applications programs.
- a state-of-art data base management system.

2.1 RELIABILITY, AVAILABILITY, AND SERVICEABILITY

As the number and variety of applications on a computer increase—and especially when they must run concurrently—reliability, availability and serviceability (RAS) increase in importance. Whereas processing interruptions of an hour were quite acceptable twenty years ago—and interruptions of 5 minutes ten years ago—the need for non-stop operation of hardware/software systems is much more critical nowadays. This need is accentuated by two factors: long continuous updating operations on large, complex data bases; and access to computers by dozens—sometimes hundreds—of online users. To achieve a high level of RAS, OS IV/F4 has been designed jointly with M series hardware components, so that the total hardware/software system operates with maximum reliability. Selected hardware and software elements are devoted to maintaining high availability—for example, machine and channel check handlers (software) and duplicate hardware elements (e.g., disk packs) which can functionally substitute for one another during an emergency.

M series hardware utilizes large scale integration (LSI) technology, which has reduced the total component count considerably from earlier computers of comparable power. A particular innovation of M series is an independent service processor (SVP) dedicated to detecting faults in the M series mainframe. RAS features of OS IV/F4 include the following:

Recovery management support (RMS)
RMS is a collection of software routines which detect, isolate, and diagnose hardware failures of mainframes and peripherals. Whenever a heardware failure occurs, RMS first attempts to recover from it, so as to maintain continuity of OS IV/F4 operations. If recovery is feasible, RMS attempts to isolate the failing device from the remainder of the system and to continue OS IV/F4 operations with reduced hardware resources, i.e., without the failing device. Principal elements of RMS are the following routines:

- Machine check handler (MCH)—recover from and diagnose CPU and main-storage failures.
- Alternate CPU recovery (ACR)—in a multiprocessor configuration, one CPU diagnoses and recovers from hardware failures in the other CPU (or an associated memory module or channel).
- Channel check handler (CCH)—recover from and diagnose I/O channel failures
- Alternate path retry (APR)—when two or more logical paths lead to a device, OS IV/F4 attempts to use alternate paths if it detects a hardware failure on the primary path.
- Missing interruption handler (MIH)—if interruption necessary for continuing I/O operations is "lost" by hardware or OS IV/F4 software, the latter will "time out" this interruption and take remedial measures.
- Dynamic device reconfiguration (DDR)—retry failing I/O operations for a mountable volume on another tape/disk drive.
- Error recovery procedures (ERPs)—analyze, recover, and re-attempt failing I/O operations.
- Logical recording of errors (LOGREC)—record all unrecoverable errors, plus summary informa-

tion on recoverable errors.

Diagnosis and recovery from hardware failures
OS IV/F4 collects failure information via RMS routines which is periodically processed by the logout analysis program (LOA) and formatted and printed by a special system utility program. I/O devices can be tested for sporadic/solid failures by the online test control program (OLTEC) furnished with OS IV/F4.

Diagnosis and recovery from software failures
OS IV/F4 furnishes numerous facilities for debugging user programs and diagnosing residual problems in the OS IV/F4 control program. These include JCL and macro-instruction facilities for displaying the contents of virtual storage (storage dumps), program tracing, and controlled modifications of production software. Debugging and maintenance of the control program are facilitated by a comprehensive diagnostic support system (DSS).

Improved security and privacy protection
The following facilities help installations to control maintenance of data sets and access to confidential data sets, check the validity of user accesses to OS IV/F4 routines and control blocks, and reduce operator errors:
- password protection of data sets and of access to timesharing and remote batch services.
- validation of user authorizations and conformance to local standards, by means of SMF exit routines.
- central storage of attributes and access authorizations for time sharing and remote-batch users.
- restricted inquiries and changes to system operations by auxiliary operator consoles, as selected by each installation.

Multiprocessor configurations
Some models of M series computers can be configured with two central processors connected to a pool of main storage modules and peripherals.

OS IV/F4 fully supports uniprocessor and multiprocessor configurations. Its Alternate CPU recovery feature will use one CPU to diagnose hardware failures of the other CPU and to continue uninterrupted operations whenever possible.

2.2 OPERATOR AND INSTALLATION MANAGEMENT FACILITIES

As computers become larger and increasingly communications oriented, their operation becomes more complex. Operator errors become more costly, and it is helpful for the operating system to furnish commands and other assistance to operators so that they can work more accurately and efficiently.
OS IV/F4 offers the following facilities to console operators:
- Highly automated processing of batch and interactive tasks, requiring minimal operator intervention.
- Coherent, efficient management of data sets and volumes.
- Simple but flexible job scheduling.
- System management facilities (SMF), to permit each installation to measure and tune individual jobs as well as overall utilization of system resources.

Automated processing
Operators must perform many routine but important tasks such as mounting tape reels and disk packs, submitting card decks and paper forms, changing print trains, and controlling the flow of job streams and individual jobs. The following OS IV/F4 facilities assist operators in performing these tasks efficiently with minimum opportunity for errors:
- Coherent and efficient management of job entry and output, via the job entry subsystem (JES).
- Support of multiple operator consoles (MCS) for specialized tasks as well as overall system control.
- Full support of display consoles, including split-screen information displays, program function keyboards, and selector pens.
- Automatic recognition of pre-mounted volumes (AVR) by OS IV/F4, plus efficient requests for setting up volumes as requested by user job control statements.
- Self-service operation for installations with large numbers of users, who can request outputs from OS IV/F4 on demand.

Management of data sets and volumes
A typical OS IV/F4 installation processes thousands of different data sets each week. OS IV/F4 assists installation managers, operators, and users in cataloging and inquiring about these data sets and associated volumes, in particular by the following facilities:
- Centralized management of volumes, data sets, and generations of data sets.
- Data set security via passwords, retention periods, authorization levels for terminal users, etc.
- The new and highly efficient virtual storage access method (VSAM).

Simple, flexible job scheduling
OS IV/F4 processes several batch jobs concurrently, typically reading and writing job streams from several local/remote devices simultaneously. Users are furnished various parameters to influence when and how their jobs are scheduled automatically. Operators can use various commands to control classes of jobs and classes of outputs, so as to meet delivery schedules while at the same time efficiently utilizing the system:
- Adjust selection priorities of each job class, sub-

ject to aging of each job in the input queue.
- Automatic validation of account codes and other job-control parameters by locally-furnished SMF routines.
- Conditional execution of job steps according to codes returned by earlier steps.

System management facilities (SMF)
At a large installation, it is important to measure overall usage of system resources and to report usage summaries periodically to installation management. Also, it is helpful to measure the relative efficiency of large jobs particularly during their development and testing.

OS IV/F4 offers SMF—plus a large number of exit points for installation-furnished routines—to perform the following tasks:
- Collect system performance data.
- Collect resource-usage data for each job, for accounting purposes.
- Intervene in standard OS IV/F4 jobscheduling algorithms, as determined by each installation.

2.3 MANAGEMENT OF SYSTEM RESOURCES

As computers grow larger and more complex, the number of expensive hardware and software resources at typical installations grows. OS IV/F4 offers many fully automatic facilities for tuning the software and operating with maximum efficiency:
- CPU management.
- Main- and virtual-storage management.
- Volume and space management for direct access storage devices.
- Spooling to unit record devices.

CPU Management
OS IV/F4 provides many facilities for efficient CPU management (also called **task management** of which the following two are noteworthy:
- Dispatching of either one or two CPUs, according to whether the configuration is uniprocessor or multiprocessor.
- Automatic raising/lowering of the dispatching priority of each active task, according to whether it has recently been I/O limited or CPU limited. This optional feature is called the Automatic priority group (APG).

Main- and virtual-storage management
OS IV/F4 fully supports the virtual-storage architecture of M series, which permits full utilization of main storage:
- Support of dynamic address translation (DAT) hardware for the CPU.
- Full exploitation of Channel DAT, an M series innovation which significantly reduces CPU overhead for I/O services in a virtual-storage system.
- 16 megabytes of address space for each batch or interactive user, to simplify programming and maintenance of applications.
- Multiple virtual address spaces—each of 16 megabytes—to permit easy design and implementation of new applications.
- Efficient paging algorithms, which move lightly-used pages to/from main storage and page data sets.

Volume and space management for DASDs
Of particular value to OS IV/F4 installations is the new and highly-efficient Virtual storage access method. VSAM provides high processing efficiency, good utilization of DASD space, convenience and efficiency for adding and deleting records, and possibilities for accessing records by several different techniques.

Another representative efficiency of OS IV/F4 DASD management is I/O load balancing, whereby nonspecific requests for space are distributed over lightly-loaded channels and devices so as to level the overall load.

Spooling to unit record devices
Simultaneous peripheral operations online (SPOOL) is a widely-used technique for achieving high throughput from card readers, card punches, and line printers; convenient presentation of system inputs for batch processing; efficient processing of system outputs, likewise; and minimizing overheads in CPU time, main storage, etc. for managing these unit record devices. JES provides a state-of-art spooling capability to OS IV/F4; its counterpart function, Remote entry services, provides the same convenience and efficiency to remote terminals.

2.4 EXPANSIBILITY

Many computer installations continue to grow after installing a major computer, forcing them to increase the amount of main storage, numbers and speeds of major peripheral devices, and equipment needed to support new application areas such as remote entry of transactions. OS IV/F4 offers the following expansibility features:
- number and size of address spaces for users.
- modes of processing jobs.
- communications networks.

Address Spaces
Up to 1536 address spaces—each furnishing 16 megabytes (16,777,216 bytes) of virtual storage—are available in OS IV/F4 for concurrently active batch and interactive tasks. Each address space is sufficiently large to accommodate all pro-

grams and subroutines for any single application.

Processing Modes
Since batch and timesharing facilities of OS IV/F4 use the same compilers, editors, etc., it is easy to move from one mode to the other. Many programs can be developed interactively, then compiled for optimum execution performance in batch mode.

Communications Networks
In a typical communications-oriented configuration, the numbers of lines and terminals are continually changing.

OS IV/F4 facilitates changes and reduces their impact on users by decoupling the overall linkage between application programs, operating system, communications access methods, and communications controllers into distinct corresponding software subsystems:

- applications-oriented subsystems such as RES, TSS, and AIM.
- Virtual Telecommunications Access Method (VTAM).
- Network Control Program (NCP).

Each of these can be independently generated and subsequently changed with minimum impact on the others; their interfaces are carefully defined to be functionally standardized.

2.5 CONVENIENT INTERFACES AND TOOLS FOR APPLICATIONS DEVELOPMENT

Among the more important facilities of OS IV/F4 are a wide variety of language compilers and service programs:

- COBOL—compatible with ANS COBOL (1972 and 1974 versions).
- FORTRAN—superset to H-level FORTRAN and conforming to the ANS and ISO standards.
- PL/I—full language.
- ALGOL—compatible with ISO standard.

With these compilers and the OS IV/F4 Assembler, users have the following facilities:

- Generate reentrant programs
  From these, a user can create executable programs which can be used simultaneously by multiple batch and/or interactive tasks.
- Dynamically link programs to a common repertoire of executable subroutines maintained online in the pagable and fixed link pack areas (PLPA and FLPA). Since object modules are reentrant, dynamic linking is meaningful and helps reduce linkage-editing tasks. It simplifies program preparation considerably.
- Request optimized object modules, for efficient production.
- Request specialized debugging packets and associated diagnostic aids for snapshotting, tracing, and dumping programs.

With the OS IV/F4 Time Sharing System (TSS), users can create COBOL, FORTRAN, and PL/I programs interactively, with special assistance from language prompters and various debugging aids. Since programs and data sets are interchangeable between batch and timesharing modes, user programs can be developed, modified, and tested interactively, then submitted for batch processing using optimized object code. Under TSS, users can rapidly check the syntax of their COBOL or FORTRAN programs with specialized OS IV/F4 Syntax Checkers, then test the programs interactively with first-rate debugging tools.

2.6 ADVANCED INFORMATION MANAGEMENT SYSTEM (AIM)

To manage a complex collection of files—organized by many different attributes, created in different formats, and accessed by various users in widely differing ways—requires enormous amounts of computer hardware, running time, and associated professional manpower. During the past decade, a number of data base management systems have been developed to address this problem. These typically have data communications subsystems to facilitate access by dozens of remote users. Hence, these software systems are often called **data base/data communications** (DB/DC) systems.

AIM is the principal DB/DC system offered with OS IV/F4. It represents the culmination of Fujitsu's long experience in designing, implementing, and enhancing DB/DC systems. It fully supports the major new equipment of M series: larger and higher-performance DASDs, VTAM and NCP, and the newest typewriter and display terminals. Likewise, AIM furnishes interfaces to the most popular languages for developing programs.

AIM is fully modularized — divided into subroutines which can be independently generated, replaced, and aggregated into efficient application-oriented subsystems. AIM can be generated as a small data-communications subsystem dedicated to one application; it can be generated as a large-scale DB/DC system for multiple integrated/distinct data bases accessed by hundreds of terminals; or it can serve intermediate-size applications with equal cost-effectiveness.

Noteworthy are the following features of AIM:

- Online network management facilities.
- Management of diverse data structures.

- Management of data resources.
- Convenient interfaces for application programs.
- Diverse modes of access.
- Independent and private operation by each user or user group.
- Extensive error-handling and recovery facilities.
- Exclusive control over blocks, data sets, segments, etc. during updating activities.

Online network management facilities
AIM provides support for diverse configurations of terminals, modems, multiplexors, concentrators, and communications lines.

Management of diverse data structures
Either data sets or entire data bases can be accessed by a universal READ/WRITE interface within AIM. Hence, programs can be developed for a single data set, then applied to complex data bases without reprogramming. Efficiency of AIM is enhanced by its data base groups (DBGs), exclusive-control attributes, deadlock-resolving logic, and such failure-recovery support as device switching and online error diagnostics.

Management of data resources
As DB/DC systems become larger and more expensive to manage, it is desirable to revise their structures and simplify access to segments and other data aggregates. To this end, AIM furnishes an online Data dictionary/directory System (DD/DS). By entering attributes about communications networks, data bases, and processing programs into his DD/DS, a user can manage his data conveniently by records, segments, or higher levels of aggregation. To tune AIM for better performance or a changed operating environment, the user (or installation manager) need only change the DD/DS, rather than modify programs or job control statements.

Convenient interfaces for applications programs
AIM furnishes efficient compilers to generate AIM-oriented applications programs, which can retrieve, sort, and generate reports from data retrieved from AIM segments in either batch or online mode. Programs can be developed in a special AIM test mode; programmers and other users can be trained to use AIM concurrent with production operations without contaminating permanent data bases. Finally, system functions can be performed—such as testing for overloads and displaying summary performance statistics—with OS IV/F4 support programs such as the Generalized log writer (GL).

Diverse modes of access
AIM supports access modes ranging from simple inquiries to full-fledged online program development and message processing. Each user chooses the mode most appropriate to his application, and all modes are simultaneously available for any AIM installation:
- single-task structure, for debugging or simple online inquiries.
- multi-task structure, for a production system servicing many concurrent requests.
- alternating structure, for servicing small numbers of requests or infrequent but intense bursts of requests.

Independent and private operation by each user or user group
AIM users operate independent of communications networks and remote terminals. Their programs can be designed and implemented independent of the data bases they will ultimately access. The programs are generally independent of the processing mode.

Extensive error handling and recovery facilities
AIM attempts to continue operation even if it encounters nontrivial hardware or software failures, by detecting them as early as possible and making vigorous attempts to recover from them. In particular, AIM attempts diagnosis, recovery, and retry of most local-device or terminal errors, prompting remote users to resubmit unintelligible commands and data. If a user program fails, AIM isolates this program from other users and attempts rapid recovery/retry.

Exclusive control over blocks, data sets, segments, etc.
When two or more tasks require simultaneous access to the same data set, segment, etc., AIM requests exclusive control of individual blocks rather than entire data sets. This approach minimizes delays and deadlocks, thereby improving system throughput without exposing data bases to contamination. In a shared-access configuration, **deadlocks** can occur—two or more tasks seize resources (typically data blocks or DASD access mechanisms) such that neither can proceed until the other releases one or more resources. By timing all resource seizures, AIM can detect when deadlock has occurred, terminate one task, and resume normal operation of the surviving tasks.

CHAPTER 3.
STRUCTURE OF OS IV/F4

The primary components of OS IV/F4 are the control program and various processing programs, the latter selected from libraries provided by Fujitsu, purchased from third parties and/or developed at each installation, as shown in Figs. 3.1, 3.3, and 3.4. The control program manages all hardware and software resources in a configuration, allocates resources to jobs (or transactions) entering the system, and controls how they are processed, as shown in Fig. 3.2.

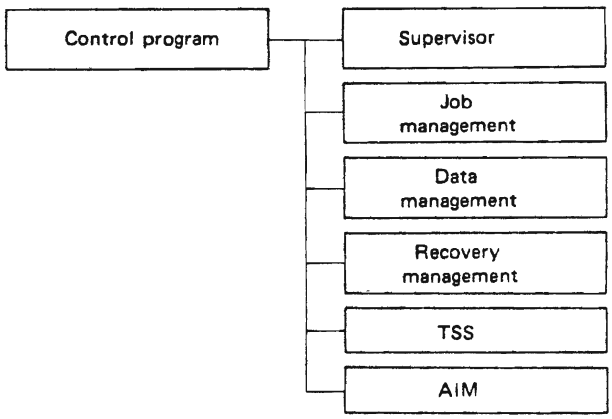


Fig. 3.2 Structure of the control program

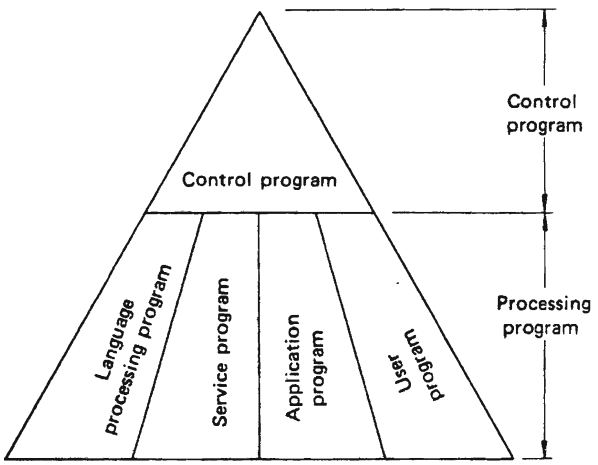


Fig. 3.1 Configuration of OS IV/F4

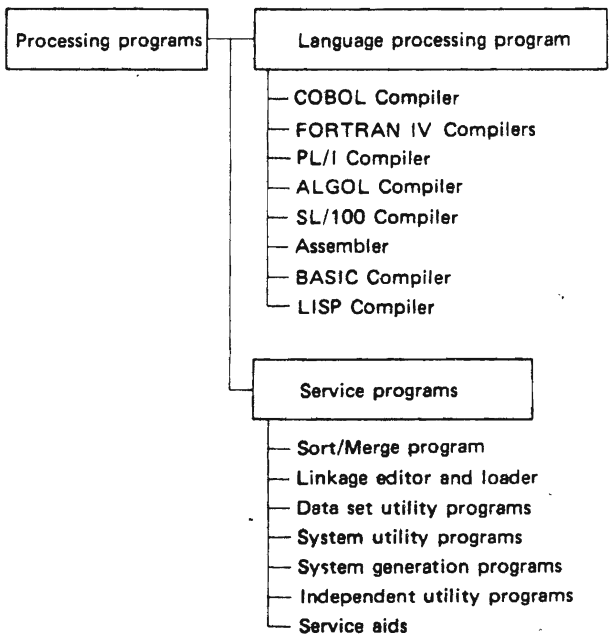


Fig. 3.3 Types of processing programs

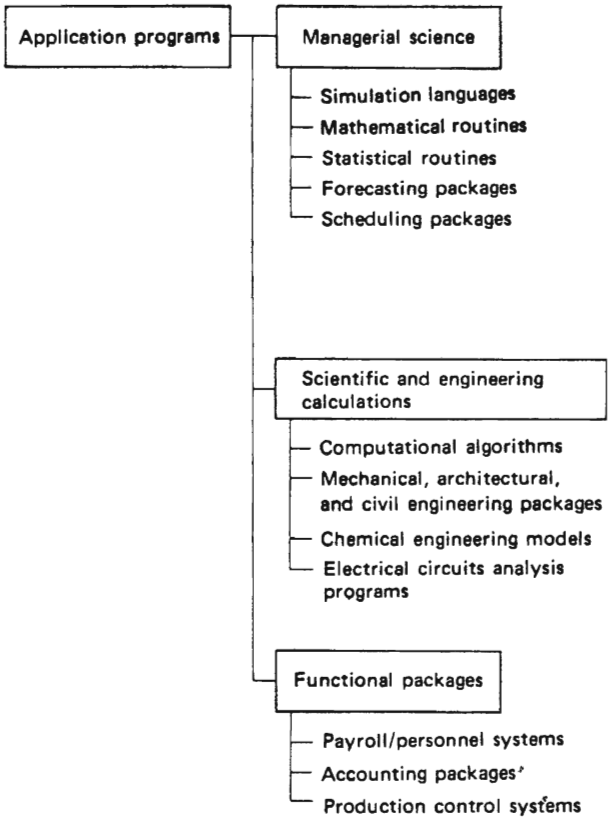


Fig. 3.4 Representative types of application programs

CHAPTER 4
PRINCIPAL COMPONENTS OF OS IV/F4

This chapter presents the functions and inter-relationships of the major components of OS IV/F4. For detailed explanations, the reader should consult corresponding chapters in Parts 2 – of this manual, plus other indicated OS IV/F4 manuals:

- Virtual storage management—Chapters 1 and 8 of Part 2 of this manual, plus the **FACOM OS IV/F4 Job Management Functions and Facilities** and **FACOM OS IV/F4 Supervisor Functions and Facilities.**
- Job management—Chapter 2 of Part 2 of this manual, plus the **FACOM OS IV/F4 Job Management Functions and Facilities.**
- Remote entry services (RES)—Chapter 3 of Part 2 of this manual, plus the **FACOM OS IV/F4 Remote Entry Subsystem Operations Guide** and **FACOM OS IV/F4 RES Terminal Commands Reference Manual.**
- Data management—Chapter 4 of Part 2 of this manual, plus the **FACOM OS IV/F4 Data Management Functions and Facilities.**
- Virtual storage access method (VSAM)—Chapter 5 of Part 2 of this manual, plus the **FACOM OS IV/F4 VSAM Functions and Facilities.**
- Telecommunications management—Chapter 6 of Part 2 of this manual, plus the **FACOM OS IV/F4 VTAM Functions and Facilities** and the **FACOM OS IV/F4 VTAM Generation User's Guide.**
- Reliability, availability, and serviceability (RAS)—Chapter 7 of Part 2 of this manual, plus the **FACOM OS IV/F4 Operator's Guide** and **FACOM OS IV/F4 A Guide to Debugging.**
- Processing programs—Part 3 of this manual, plus the **FACOM OS IV/F4 Applications General Description** and descriptions of particular applications packages.
- Time sharing system (TSS)—Part 4 of this manual, plus the **FACOM OS IV/F4 TSS Commands Reference Manual,** and **FACOM OS IV/F4 TSS Terminal Operator's Guide.**
- Advanced information management system (AIM) – See the **FACOM OS IV/F4 AIM General Description, FACOM OS IV/F4 AIM System Design Guide.**

4.1 VIRTUAL STORAGE MANAGEMENT

OS IV/F4 provides **multiple virtual storages** (also called **address spaces**), one per active batch or interactive user. These provide complete independence of each job from other jobs with respect to the availability of main storage.

4.1.1 Multiple Virtual Storages

OS IV/F4 allocates 16 megabytes (16,777,216 bytes) of virtual storage to each user as his program starts executing. Over 1500 such virtual address spaces can be simultaneously active, although typically 15 batch users and 100 timesharing users will fully load a single large OS IV/F4 system, as shown in Fig. 4.1.

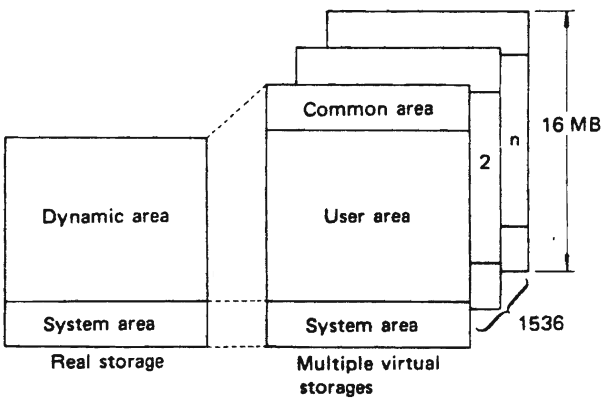


Fig. 4.1 Correspondence between real storage and multiple virtual storages

4.1.2 Page Management

Each installation can allocate up to 16 DASDs for **paging**, a system activity which stores **pages** (blocks

intervals when they are unused and returns these pages to main storage as soon as they are re-referenced. Salient aspects of OS IV/F4 paging are described in the following paragraphs.

Page management algorithm
OS IV/F4 uses a relatively simple LRU (least recently used) algorithm to determine which pages should be **paged out** of main storage (or discarded, when unmodified copies of these pages exist on a paging DASD), according to how recently they have been referenced—those least recently referenced by user programs or the OS IV/F4 control program are removed first.

Page recovery
Often OS IV/F4 will logically remove a particular page from a user's address space without overwriting its **page frame** (corresponding main-storage block of 4K bytes). If the user then references this page, OS IV/F4 is capable of adding the unmodified page back to the user's address space; the corresponding page fault can be satisfied without paging in a fresh copy.

Omission of unnecessary paging
As noted above, if a page is to be removed from main storage, and if this page corresponds to an unmodified copy on DASD, the OS IV/F4 Paging Supervisor discards this page rather than writing it out unnecessarily.

Slot sorting
Since pages have fixed lengths (4K bytes), the number held on each DASD track (of a page data set) is pre-determined. OS IV/F4 periodically sorts all page requests outstanding for a particular paging device and satisfies them in **slot sequence** (relative positions on the current cylinder) rather than **FIFO sequence** (first in, first out) or any other arbitrary sequence. This technique maximizes the average transfer rate of pages to/from DASDs, which alleviates paging bottlenecks.

Paging to different types of DASDs
An installation can allocate devices with different sizes and speeds to page data sets, for example, a drum (high transfer rate, small capacity) and several disk drives (medium transfer rates, large capacities). The OS IV/F4 Paging Supervisor will utilize the drum for a small number of frequently-referenced pages, such as those of the Pagable link pack area (PLPA), which are shared by all users. The Paging Supervisor will store infrequently-referenced pages on disk drives. This overall approach maximizes the cost-efficiency of paging to different DASDs.

Swapping
Timesharing users often remain inactive for several seconds — even minutes. The OS IV/F4 Paging Supervisor detects such inactivity and removes all of 4K bytes) of each program on DASDs during pages corresponding to these users from main storage (**swapping** their address spaces) until these users resume usage of the CPU.

Fixed pages
For tasks requiring fast response times, such as inquiry applications, users can request that some/all pages be **fixed** in real storage, i.e., ineligible for paging to DASDs. They can specify ADDRSPC=REAL on corresponding JCL statements—requesting that all of their pages be fixed in real storage for the duration of the job or job step —or they can issue PGFIX (Page fix) macro instructions to insure that selected pages are fixed for specific durations. Later (or at the end of the job step), they can release these pages with PGFREE macro instructions.

Automatic adjustment of paging rates
If many users compete for main storage by simultaneously executing programs requiring many active pages, paging rates often rise rapidly. This phenomenon characterizes most virtual storage systems, including OS IV/F4. If the paging rate rises too high, the system is said to be **thrashing** and can perform little useful work. To relieve thrashing as soon as it occurs, OS IV/F4 automatically **suspends** one or more low-priority tasks. Suspension consists of swapping out active pages for these tasks for a prolonged period of time—possibly minutes—until the paging rate subsides and stabilizes. Suspended tasks are resumed, one at a time, when the paging rate is satisfactorily low.

4.1.3 Channel Dynamic Address Translation (Channel DAT)

Just as the CPU can dynamically translate virtual addresses to real-storage addresses, Fujitsu M series I/O channels can also translate virtual addresses automatically; this is a unique capability of M series computers, giving them a significant performance advantage over prior systems. With Channel DAT, addresses in channel command words (CCWs) and associated operand/data areas may refer to virtual storage. OS IV/F4 fully supports this hardware feature.

4.2 JOB MANAGEMENT

OS IV/F4 Job Management is a collection of routines which accept batch jobs from numerous input devices, queue them for execution, select and control their execution, and transcribe their outputs to appropriate printers, card punches, magnetic tapes, etc.

4.2.1 Job Entry Subsystem (JES)

JES is a function within OS IV/F4 Job Management which reads jobs, writes their outputs to appropriate devices, and operates unit record devices (card readers, card punches, line printers, etc.) at maximum speed. Salient features of OS IV/F4 JES are as follows:

Centralized management of buffers
A single pool of buffers is used for readers, writers, and spool devices. This approach reduces the aggregate real-storage requirements for JES, particularly since buffers are allocated and formatted to keep page-fault interruptions to a minimum.

Efficient spooling
Each spool data set is divided into a number of small equal-size units called **logical cylinders.** JES allocates spool storage to JES readers and OS IV/F4 initiators in integral logical cylinders for each user. Logical cylinders are allocated dynamically, based on which empty logical cylinder is nearest the current position of the spool-device access mechanism. This technique minimizes delays due to moving these access mechanisms. Often JES can allocate an empty logical cylinder from the current physical cylinder, i.e., corresponding to the current position of the access mechanism.

Also, JES utilizes spool data sets on several different devices if the installation has generated multiple data sets; this improves the probability of allocating a logical cylinder with little/no delay.

Simple operating procedures
The JCL statements required for a JES reader or writer are usually prepared by an installation and stored in the system procedure library (SYS1.PROCLIB). This approach facilities simple operator commands for starting readers and writers, since most parameters can be predetermined. Operators can override default parameters by specifying various parameters with their START commands, which furnishes great operational flexibility.

4.2.2 Multiple Console Support (MCS)

OS IV/F4 utilizes one **main console** to control the entire configuration, plus up to 31 **auxiliary consoles** to perform specialized functions and back up one another and the main console. Consoles can be keyboard/printers, displays, line printers, and other appropriate devices.

Functional consoles
Auxiliary consoles can be dedicated to such functions as managing tape reels (displaying which reels are mounted, to be mounted, finished processing and to be kept/scratched, etc.), managing disk packs (mount and dismount messages), servicing line printers (requests for chain/train changes, forms changes, verification of forms positioning, etc.), and other specialized activities. This MCS facility permits large installations to create functional work areas and other management efficiencies.

Display consoles
Based on CRT displays, these consoles permit operators to interact efficiently and accurately with the OS IV/F4 Master scheduler. Optionally, an installation can provide program function keys (PFKs) and a selector pen to further simplify entry of commands and responses. The overall internal status of OS IV/F4 — or of selected activities within OS IV/F4, such as particular readers, initiators, writers, TSS, or executing jobs — can be displayed on command. Certain dynamic status indicators — such as which jobs are executing under various initiators — can be automatically displayed in one area of the display, changing whenever their status changes or at periodic intervals.

Hardcopy log
An OS IV/F4 installation can define a **hardcopy log** to record all operator commands, corresponding system actions, and system-issued operator messages on one chronological listing. The hardcopy log facilitates operator interventions for emergencies and quick, reliable restarts as necessary.

4.2.3 Efficiency Enhancements

OS IV/F4 contains several state-of-art facilities for improving system throughput, simplifying entry of batch jobs, and providing job turnaround as expected by users.

Priority aging
In general, each initiator selects jobs according to class (CLASS parameter) and selection priority (PRTY parameter). However, this algorithm tends to keep low-priority jobs on input queues for prolonged periods, especially if an installation is processing a heavy load of high-priority jobs. **Priority aging** is an optional OS IV/F4 feature which permits each initiator to periodically increment the selection priority of jobs in certain classes. Low-priority jobs can thereby rise gradually in priority until they are eligible for selection.
OS IV/F4 furnishes several parameters to control priority aging, designating which initiators, which job classes, at what time intervals, and to what upper limits age increments can be made.

I/O load balancing
As OS IV/F4 satisfies nonspecific volume requests, it selects DASDs (and associated channel paths) which are relatively lightly loaded. Selections are

based on statistics maintained internally by the OS IV/F4 I/O Supervisor, which show how many users have allocated data sets to each DASD and aggregate recent activity, in terms of how many I/O requests were issued to each device during the previous n seconds. I/O load balancing thus levels DASD activity across all devices and channel paths for all batch and time sharing users.

Execution batch processing

For each normal job, the OS IV/F4 job initiator performs several non-trivial tasks such as allocating I/O devices, allocating volumes and data sets, and interpreting and processing its JCL statements. The **execution batch facility** relieves this overhead considerably for small standardized user jobs such as compilations, compile-go test jobs, and routine production jobs. This facility processes all jobs of a certain pre-specified class as a single SYSIN stream in one job step, although the resulting SYSOUT data sets are segregated by user and returned to local/remote destinations just as if they had been fully processed by the OS IV/F4 job initiator. In addition to reducing the pre-job overhead for initiation functions, the execution batch facility permits users to furnish streamlined JCL; typically, a JOB statement can be immediately followed by SYSIN data, since EXEC and DD statements are pre-defined for this class of jobs by a single cataloged procedure.

4.2.4 Installation-management enhancements

In addition to the internal efficiency enhancements described in the preceding section, OS IV/F4 provides many facilities to assist programmers, operators, and installation managers to use the system more conveniently, such as the following representative features.

Demand output

Typically, users of a large computer center deliver card decks and other inputs to a counter, where they are received and logged by I/O-control personnel. These personnel then submit the inputs to the computer, store the source materials in bins, and await processing of these jobs. When OS IV/F4 has finished printing and punching job outputs, the I/O-control personnel identify each user's output and store it in a bin (often the same as for the corresponding source decks), where the user can subsequently claim it.

This set of procedures and manual handling has several drawbacks. First, the number of I/O-control personnel is typically rather large. Second, the user must interact with these personnel both when submitting his job decks and when picking up his outputs; this imposes unnecessary delays and inefficiencies. Finally, the overall turnaround time per job is lengthened as compared to the demand-output procedure, where a large percentage of users can—if they wish—submit their source decks directly to a local card reader. Each user claims his outputs by submitting a single JES statement to this card reader subsequent to entering his card decks; this /*OUTPUT statement identifies him and requests OS IV/F4 to immediately print (and/or punch) all outputs waiting for him on the demand output SYSOUT queue.

In summary, **demand output** is a self-service facility which can be conveniently used by a large percentage of local job submitters. Its primary advantages are reduced efforts by I/O-control personnel in handling system inputs and outputs and faster turnaround for job submitters, who handle their own card decks and retrieve their printed/punched outputs on demand.

Anticipatory volume setups

Earlier operating systems often issued volume-mount messages on demand; a job would commence executing before the OS issued messages to the operator requesting that he mount various tape reels and/or disk packs. An alternative (and supplementary) approach to demand mounting is anticipatory setup of volumes, which is supported by OS IV/F4 as follows:

As JES reads source decks from various devices, it scans JOB and SETUP control statements of the following format:

```
//job-name  JOB    account-number,
                   programmer-name,
                   CLASS=...
/*SETUP      PAYMST(FP),TIMES,NEWPAY
```

The contents of each SETUP statement are immediately displayed on the hardcopy log device, and OS IV/F4 puts the corresponding job into the input hold queue; it cannot be selected for execution until it is explicitly released by the operator.

The operator can retrieve all tape reels and disk packs required for this job by inspecting the hardcopy log. He can optionally premount some/all packs and labelled reels, since the **automatic volume recognition** (AVR) feature of OS IV/F4 will recognize these volumes and record the fact that they are mounted and ready. The operator then releases the job, which will issue demand mount messages only for volumes which are not already mounted. Hence, it is entirely feasible for the operator to premount all volumes needed by small jobs, so that they experience no delays after being released for execution.

The advantages of this approach are that operators can retrieve several volumes at one time from the tape/disk library and bring them to the volume-setup area in an orderly, preplanned procedure; and that after OS IV/F4 initiates a job, the initiator (and resources it controls, such as virtual storage,

devices, and data sets) is delayed minimally for volume-mounting activities. The overall effect of anticipatory setups is to smooth operations and to improve processing efficiency for setup jobs.

System management facilities (SMF)
SMF is an OS IV/F4 function for collecting summary statistics on system performance, such as aggregate usage of CPU time, channels and devices, main and virtual storage, plus such system events as initiator start/stop times, reader and writer start/stop times, volume-mountings, etc. SMF also will optionally collect and capture on the **SMF data set** records of resources used by individual jobs and job steps, so that installation managers can scrutinize long-running jobs for possible inefficiencies. SMF facilitates installation accounting for computer usage, since it can generate a comprehensive journal of which jobs used which hardware/software resources. Periodically, the console operator transcribes data from the SMF data set to a permanent archive, using a special OS IV/F4 utility program. This archieve can be processed and summarized by installation-developed report programs to develop whatever management reports are desired.

SMF furnishes many exit points during the JES Reader, Job Initiator, Step Initiator, Terminator, and JES Writer routines where each installation can insert locally-written routines for the following fuctions:

- validate various parameters on user JCL statements.
- impose installation-defined limitations on CPU times, SYSOUT quantities, etc.
- capture additional data on jobs and their resource usage.

4.3 REMOTE ENTRY SERVICES (RES)

RES is a facility for remote entry of jobs to OS IV/F4, furnishing an interface to users and operators of remote terminals which is essentially identical to the JES interface for local users. RES provides readers and writers similar to those of JES, and it merges remotely-submitted jobs onto the same input queue. RES provides operator-communication facilities for remote operators to send messages to central-site operators and to one another. Under some restrictions, RES operators can manage their input streams just like central operators; they can start and stop their terminals, communications to the central site, and associated input readers and output writers. They can display the status of jobs entered through their terminals and outputs awaiting return to their terminals.

Jobs can be eentered remotely via RES and printed/punched at the central site via JES, if requested by the user (or if the job are rerouted by a local/remote console operator). Likewise, jobs can be entered via JES and routed to a remote terminal. Finally, jobs can be entered through on terminal and their outputs selectively routed to other terminals.

Starting a RES session
The remote operator starts a session by establishing a communications link to the central site, then issuing a LOGON command. Authorization to submit jobs in various classes and priority levels is verified by RES, utilizing attributes for terminal users stored in the SYS1.UADS data set at the central site. Broadcast messages accumulated for this terminal at the central site are displayed to the remote operator when he logs onto RES.

Job flow
Each job is read from the remote terminal, transmitted block by block over the communications link, and received at the central site through a communications control processor (CCP) and the virtual telecommunications access method (VTAM) into RES, which stores the image of this job on the spool data set. RES enters a control record into the system job queue (SYS1.SYSJOBQE data set), merged with control records for jobs submitted from other terminals and—via JES—through local input devices.

Thereafter, remotely-submitted jobs are queued and processed just like locally-submitted jobs. Their outputs normally return to the terminal from which they were submitted, although—as indicated above—they can be optionally rerouted. The remote operator can use his console (or card reader, if the terminal has no console) to inquire about the overall OS IV/F4 operational status, jobs input from his terminal, or the queue of output data sets awaiting return to his terminal. The remote operator can respond to messages and queries issued by executing programs previously submitted from his terminal; he can cancel jobs during input, execution, or after their outputs have been generated. He can use WRITER commands to control outputs flowing to his terminal; change forms, request multiple copies, forward space the SYSOUT stream, repeat earlier portions of a SYSOUT data set, and other useful operator functions.

Messages to/from the central site
At the central site, the broadcast data set (SYS1.BRODCAST) contains messages for all remote terminals in the network. Some messages are issued by central-site operators, others by remote operators. Since terminals are not necessarily online when messages are submitted, the latter are queued on disk until the receiving terminals log on. Messages may be broadcast to all terminals or restricted to a single terminal.

4.4 DATA MANAGEMENT

OS IV/F4 data management facilities utilize the M series virtual-storage and channel-DAT features to provide state-of-art facilities and unmatched performance. Of particular note are OS IV/F4 capabilities to perform chained scheduling into virtual storage, support paper-tape equipment comprehensively, and read several sequential data sets in parallel—the **parallel GET** facility. These and other feature improve system efficiency considerably over prior systems.

Buffering options
The user can select any of the following buffer techniques, as appropriate to his application:

- Queued sequential access method (QSAM)
  Simple, exchange, or dynamic buffering.
- Basic access methods (BSAM, BPAM, and BDAM)
  Simple or dynamic buffering.

BDAM Features
These include DASD address feedback (current and/or next address after each READ/WRITE operation), extended searches (for particular records, over multiple tracks), and exclusive control of individual blocks within sharable data sets, in addition to exclusive control of entire data sets.

Space management
Improved allocation/deallocation algorithms have reduced the CPU overhead for space management by 20—30% compared to prior systems.

Data set security features
In addition to control passwords used by installation managers, OS IV/F4 offers secondary passwords which can be defined and presented by users with their data set requests. OS IV/F4 checks both passwords (as appropriate) and denies access to persons unable to furnish correct passwords.

Chained scheduling
Rather than issuing a separate channel program for each **I/O request** (EXCP macro instruction or equivalent higher-language verb), OS IV/F4 optionally permits users to dynamically chain requests together. The OS IV/F4 I/O Supervisor performs the chaining and monitors its successful performance. **Chained scheduling** serves to reduce the CPU overhead for issuing I/O hardware requests rapidly and improves the throughput of corresponding devices, since they transmit data continuously through the channel so long as their chains continue.

Parallel GET
This OS IV/F4 feature permits an assembler-language user to issue read requests to several data sets concurrently, accepting corresponding data blocks in any sequence. This facility permits better CPU utilization, since the job can proceed after any one of the I/O operations completes rather than requiring these operations to complete serially and synchronously.

4.5 VIRTUAL STORAGE ACCESS METHOD (VSAM)

VSAM is a relatively new access method and associated data set organization, developed to supersede the indexed sequential Access Method (ISAM) and—in some cases—specialized uses of the sequential and direct access methods (SAM and DAM). VSAM data sets may only be created on DASDs, in one of the two following formats: **key sequenced** which are similar to indexed sequential data sets in format and usage; and **entry sequenced,** which are similar to sequential data sets in these respects.

Every VSAM data set is cataloged into the VSAM catalog, which is a mandatory component of an OS IV/F4 system and serves as a high-level catalog manager for non-VSAM data sets as well. For VSAM data sets, the VSAM catalog contains numerous format and usage attributes; in this respect, it differs considerably from non-VSAM catalog entries, which record for each data set only the device types and serial numbers of volumes on which it resides.

Access method services (AMS) is a comprehensive service program designed to handle VSAM data sets, display their contents, and catalog and uncatalog non-VSAM data sets.

The following aspects of VSAM are particularly noteworthy:

High processing efficiency
Skip sequential access to portions of VSAM data sets is facilitated by a hierarchical indexing structure. This reduces the need for sequential searching, yet it provides the efficiencies of sequential processing once desired records have been located.

Indexes and data blocks can be retained in virtual storage indefinitely, so long as they are frequently referenced. This eliminates the ISAM clumsiness in requiring fixed allocations of real storage for buffers and various indexes.

On the indexing track of each cylinder, indexes can be optionally duplicated several times so as to reduce rotational latency when they are sought by VSAM.

During key-sequenced access to a VSAM data set, several consecutive records can be updated in one virtual storage block prior to updating its DASD image, in contrast to keyed accesses to ISAM data sets (where each record updating requires rewriting the corresponding block). Also, several VSAM records can be inserted onto a track in one opera-

tion; VSAM reads the image of the entire track, inserts the new records into the virtual storage buffer, and rewrites the entire track with one I/O request.

Within most VSAM indexes, keys can be compressed by removing unnecessary leading and trailing characters. Key compression serves to reduce their DASD storage requirement and to increase the logical content of each block of keys read into main storage.

Creators of VSAM data sets can preplan expansion space; OS IV/F4 will insert free space periodically while writing a VSAM data set—as directed by user parameters—which will help reduce record movements and track reformatting when new records are subsequently added to the data set.

Easy management and control
Since the VSAM catalog contains most attributes for all VSAM data sets, installations can more easily manage data centrally and consistently. User JCL statements for VSAM data sets are typically much simpler than for other data set organizations. Also, usage and efficiency statistics are more easily and reliably collected by VSAM.

Since records are not managed by physical addresses but rather by their relative byte addresses (RBAs) from the start of a VSAM data set, device independence is much more useful and meaningful than for other data set organizations.

Protection and maintenance of data sets
Four levels of passwords can be assigned to VSAM data sets, according to various authorizations to read, update, add, or access the data. Automatic journaling of transactions can be optionally requested.

4.6 DATA COMMUNICATIONS

OS IV/F4 furnishes a virtual telecommunications access method (VTAM) to control all access to terminals, whether used for remote job entry, time sharing, inquiry applications, or other online functions. VTAM controls not only terminals connected via communication lines but also local character displays connected directly via channels. VTAM uses a new type of programmable communications control processor (CCP), which operates under its internal program called the network control program (NCP). Part of the data communications function previously performed by the host processor is performed in OS IV/F4 by the NCP. This architecture distributes processing tasks to two or more specialized computers, achieving higher host-processor efficiency, faster response times, and greater overall cost-effectiveness.

4.6.1 Virtual Telecommunications Access Method (VTAM)

VTAM permits application programs to communicate with terminals without any consideration of intervening control units, communication lines and moderns, or the CCP. The main functions of VTAM are to allocate communications resources, establish communication links to remote terminals, and transfer records to/from terminals.

Pools of communications resources for multiple applications
VTAM manages pools of network resources: control units, lines, and terminals. VTAM permits diverse application programs to share these network resources for several different functions simultaneously. VTAM also permits a single terminal to communicate with different application programs and online facilities, as chosen by the terminal user.

Establishing communications links
VTAM issues I/O requests—in conjunction with the NCP program—which establish network paths for communication between application programs and terminals. Application programs may request connection to any appropriate terminal. Requests for communications links can also be made from the terminals.

Data transfer
After VTAM has connected an application program to a terminal, the application program exchanges data with the terminal. VTAM furnishes macro instructions corresponding to various types of terminals. Since NCP performs most of the transmission control functions, application programs may use these macro instructions (or their higher-level language equivalents) depending on the types of lines and terminals. Users present EBCDIC-coded records to VTAM, ignoring any consideration of how these data are translated into transmission codes. On the other hand, records presented by terminals to central site application programs are converted to EBCDIC by the NCP; therefore, no translation or modification need be performed by user programs—for example, eliminating line control characters.

4.6.2 Network Control Program (NCP)

A program operating in a CCP is called a **network control program** (NCP); its principal function is to transmit data received from the host processor (via VTAM) to terminals, and vice versa. NCP recognizes transmission control characters, assembles and disassembles characters for controlling line timeouts, and records and diagnoses errors.

The principal advantages of NCP−CCP architecture, compared with prior architectures for communications control, are as follows:

- Since complicated line control characters, line control procedures, error recovery processing, etc. are all performed by the NCP inside the CCP, the host-processor load is reduced, resulting in higher efficiency for applications programs.
- Terminals and lines with different line control procedures can be easily expanded without modifying existing applications.

4.7 RELIABILITY, AVAILABILITY AND SERVICEABILITY (RAS)

The RAS facilities of OS IV/F4 help prevent and/or recover from hardware failures. RAS also helps diagnose certain software failures, described under "Dynamic Support System" and "Generalized Trace Facility" below. RAS comprises a collection of hardware diagnostic facilities and OS IV/F4 software routines which operate jointly to produce high levels of system availability and quick recovery from failures.

4.7.1 System Recovery

If a hardware failure occures in an M series configuration, certain recovery actions are attempted automatically, such as error checking and correction (ECC). In case hardware components alone can not recover from the failure, OS IV/F4 recovery management support (RMS) software is invoked to attempt recovery. This multistage recovery strategy has proven quite successful and cost-effective for achieving high system availability.

Recovery by RMS

- CPU and main-storage failures
  The machine check handler (MCH) analyzes CPU and main-storage failures; it tries to exactly restore the machine status preceding these failures and reattempts corresponding instructions.
- Recovery by Means of an Alternate CPU
  In case a hardware failure occurs in one CPU of a multiprocessor system, the alternate CPU recovery (ACR) facility of OS IV/F4 attempts to recover and continue system operations.
- Channel failures
  The OS IV/F4 Channel Check Handler (CCH) analyzes channel failures and performs recovery processing when appropriate.
- Retry by alternate channel paths
  For I/O devices which access main storage through two or more channel paths, the OS IV/F4 alternate path retry facility will reattempt failing operations on alternate paths.
- Timing out missed I/O interruptions
  After each I/O request is issued, the OS IV/F4 missing interruption handler (MIH) tests for completion of the request at certain intervals; if the hardware interruption is "lost," MIH will intervene to permit processing to continue.
- I/OHardware failure analysis
  The OS IV/F4 error recovery procedures (ERPs) analyze failures during I/O operations and perform appropriate recovery processing.

4.7.2 System Restoration

OS IV/F4 provides system restoration functions to detect hardware failures, quickly diagnose their causes, and thereby reduce preventive-maintenance and emergency-repair intervals to a minimum.

Diagnosis by the M series service processor (SVP)
Each M series configuration contains an independent SVP, which ordinarily can continue operation even if the central CPU or main storage has partially/totally failed. The SVP diagnoses failures quickly using prestored diagnostics, permitting rapid resumption of normal operations.

Detailed status of important hardware components is continually displayed on the SVP console, which reduces the time needed for maintenance and recovery episodes. It also helps maintenance personnel to diagnose hardware failures quickly.

Prevention of failures and improved diagnostic facilities
For any large computer configuration, it is necessary to continually record the system status, particularly minor/major failure episodes for hardware and systems software. Such a record permits maintenance personnel to perform anticipatory maintenance of marginally-operative hardware components, as well as to quickly and successfully perform emergency maintenance when a critical hardware component malfunctions chronically.

RMS records information about solid/intermittent hardware failures into the LOGREC data set, which is a permanent DASD data set. Serious failures are also logged onto the system hardcopy console and−if attached−a display console. RMS collects complete information about location and probable cause of each failure by interrogating various hardware registers and main-storage logout areas. Periodically, a maintenance engineer can process the LOGREC data set with a special service aid which sorts, summarizes and prints reports about all nontrivial errors during the preceding interval.

Besides RMS, OS IV/F4 furnishes an **online test**

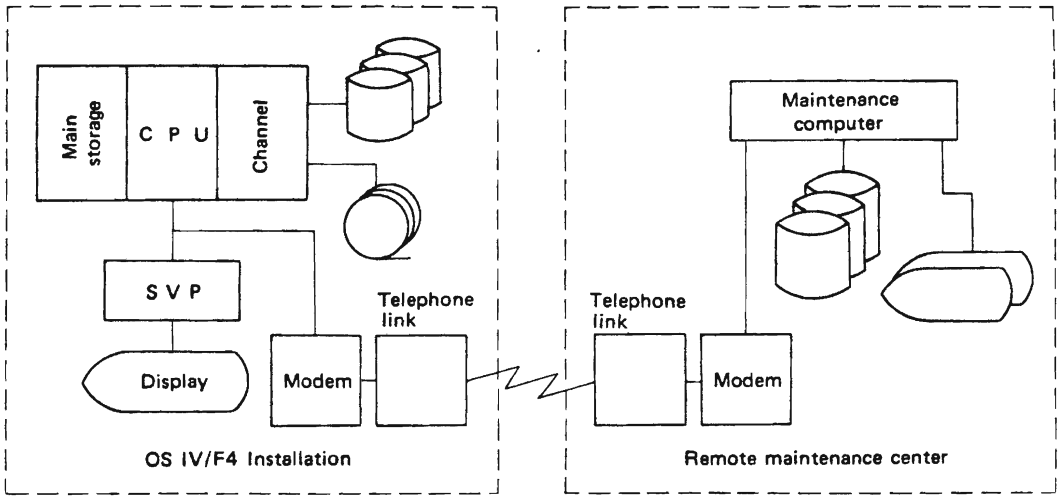


Fig. 4.2 Remote maintenance

control program (OLTEC) which performs diagnostic tests on any designated I/O control unit or device. OLTEC can analyze marginal or solid failures in any M series peripheral device, whether it is logically online to OS IV/F4 or offline—the only requirement for OLTEC testing is that the device be physically online to the central configuration. OLTEC can also be invoked by the console operator to verify that some/all peripherals are correctly functioning, as a routine verification of hardware status. OLTEC can operate either concurrent with production operations or stand-alone.

For diagnosing and correcting residual failures in systems software—the OS IV/F4 control program, compilers, service aids, etc.—the dynamic support system (DSS) is particularly efficient and accurate. The installation system programmer can ask DSS to suspend system operation at a pre-designated address and collect various data, specified in the DSS command language. When appropriate, the programmer can even use DSS to apply and test changes to systems software.

The generalized trace facility (GTF) collects data on performance and status of systems software; it can also collect similar data on selected user programs. Summarized data can be printed periodically to help diagnose software problems and thus improve maintenance procedures and overall system reliability. OS IV/F4 offers several service aids to display detailed and/or summarized trace data.

Remote maintenance of M series computers
Any maintenance procedure which can be performed locally on an M series configuration can be optionally controlled from a remote maintenance center operated by Fujitsu Limited. The operator dials into the remote maintenance center, using one communications link attached to the service processor. The remote maintenance center also furnishes technical assistance via voice telephone to a local maintenance engineer, and it can thereby furnish rapid, high-quality and consistent maintenance to remote configurations. OS IV/F4 routines for remote maintenance are called maintenance assistance by remote telecommunications (MART); they protect the integrity and privacy of user data sets fully.

4.8 SUPERVISOR

The supervisor provides fundamental control and support functions for the entire configuration. Highlights of the OS IV/F4 Supervisor are the following features.

4.8.1 Multiprocessing Support

Prior Fujitsu computers offered multiprocessing support, such as the FACOM 230-55, 230-58, 230-60 and 230-75 systems. The state-of-art multiprocessing options for OS IV/F4 are derived from experience with these systems and advanced features in M series hardware, of which the following are representative:

Local locks
A lock is a control block which may be set, reset, and tested by several unrelated tasks in order to access a serially-usable resource in an orderly and non-hazardous sequence. OS IV/F4 contains many **local locks** and a few **global locks**; the former are used within one address space, the latter are available to all tasks in a configuration, whether this configuration contains one or two CPUs. By increasing the number of locks, and by narrowing the scope of

resources controlled by each lock, OS IV/F4 increases system throughput even if two CPUs are being utilized to access numerous serially-usable resources.

Minimum disabled state
A CPU is **disabled** for interruptions when the OS IV/F4 supervisor determines that a brief computation must be completed immediately and without interruptions. The OSIV/F4 Supervisor has an improved design in this respect, compared to prior systems; most OS IV/F4 tasks run enabled for interruptions at all times, a few tasks run disabled for minimum intervals. This approach serves to maximize responsiveness of the system to I/O interruptions and other performance critical events and to minimize the likelihood of deadlock events (neither CPU can proceed without release of a resource held by the other).

4.8.2 Automatic Priority Group (APG)

APG is a feature of the OS IV/F4 Task Supervisor which modifies the dispatching priority of certain user tasks periodically to reflect whether each task was relatively CPU-limited or I/O-limited during the preceding time interval. APG raises the priorities of tasks which received little CPU service during the previous interval, lowering the priorities of tasks which received relatively heavy CPU service. The effect of APG is to raise the overall performance of I/O equipment on the system without lowering the CPU utilization; in turn, this improves overall system throughput and hardware utilization. APG can be explicitly/implicitly requested by a user for each job step; it is the default method for determining dispatching priorities for batch jobs.

4.9 PROCESSING PROGRAMS

Under OS IV/F4, **processing programs** comprise language processors (compilers and the assembler), service programs, and application programs. The first two categories are discussed briefly below; additional details may be found in Part III of this manual and in the **FACOM OS IV/F4 Applications General Description manual.**

The OS IV/F4 language processors are enumerated in Table 4.1 and have the following highlight features.

Conversational facilities
For several languages, OS IV/F4 furnishes special syntax-checking facilities and /or prompters and/or interactive debugging facilites. These tools permit programmers to create, test, and execute their programs interactively with great convenience and cost-effectiveness.

Optimization options
For several languages, OS IV/F4 offers optimizing options with their compilers, which generate programs which are smaller than usual and/or execute faster.

Reentrant object programs
At the user's option, compiler for the COBOL, FORTRAN, and PL/I languages can generate reentrant object programs, which are useful for multitasking applications and the system link pack.

Dynamic link structure
The dynamic link structure is a novel OS IV/F4 facility for managing object modules, as an alternative to the traditional management of object modules and load modules. Dynamic linking facilitates maintenance of rapidly-changing programs.

Debugging tools
OS IV/F4 furnishes a wide variety of powerful debugging tools, some of which are specified by users in their source programs, others selected when they compile these programs. In several languages, tracing, snapshotting, and dumping facilities are easily specified.

Table 4.1 OS IV/F4 Language processors

Processor	Features/functions
ANS COBOL	Most features of the 1974 ANS standard.
FORTRAN IV (GE)	Meets the ANS standard. Used for developing programs; contains many debugging features. Compiler is reentrant.
FORTRAN IV (HE)	Meets the ANS standard. Used for generating optimized programs.
ALGOL	Meets the 1960 ISO standard.
PL/I	Full implementation.
SL/100	Software implementation language developed by Fujitsu. High-level control statements – IF, DO, GO-TO, etc. – are added to the assembler language.
Assembler	Machine-instruction operation codes, macro instructions for sequences of instructions, and other typical facilities.
BASIC	Dartmouth University BASIC plus character string manipulation, formatted print-outs, linkage with FORTRAN programs, etc. Runs under TSS.
LISP	Based on LISP 1.5. Runs batch or TSS. Suitable for string processing.

The OS IV/F4 **service programs** comprise the sort/merge program, the linkage editor, the loader, a utility program for dumping and restoring volumes,

and various utility programs for generating, copying, deleting, moving, comparing, and displaying data sets and their catalogs.

Table 4.2 Service programs

Program	Features/functions
Sort/merge	Work files on tape or DASD. Sort/Merge program is reentrant.
Linkage editor	Combines object programs and/or load modules to create new load modules. Optionally, load modules can be (a) reentrant, (b) structured into overlays, or (c) structured in various other ways.
Loader	Combines object programs and/or load modules into an executable program in main storage. Less CPU and DASD overhead than the linkage editor.
System utility programs	Copy or move contents of one volume to another; initialize volumes; list system control information; etc.
Data set utility programs	Process sequential, partitioned, or system data sets to perform copying, comparing, displaying, dumping, restoring, and other functions. Handle records or (in some cases) fields within records.
Independent utility programs	Initialize DASD volumes; dump/restore DASD volumes to magnetic tape reels or other DASD volumes.
Service aids	Detect, dump, restore, summarize, report, etc. data accumulated in the LOGREC, SMF, and other special-purpose data sets.
System generation	Selects and combines modules, assembles parameter tables, and prepares a customized version of OS IV/F4.

4.10 TIME SHARING SYSTEM (TSS)

OS IV/F4 offers a responsive, reliable, high-performance TSS, which is easy to use for a wide variety of applications. TSS can be simultaneously accessed by many users, each using TSS facilities as if he had exclusive control of a powerful interactive computer. Outstanding features of OS IV/F4 are cited below.

Conversational entry of batch jobs
Jobs are entered from a keyboard terminal, using full TSS facilities for checking and editing source statements. After a job has been entered, part/all of it can be compiled for interactive testing. Thereafter, the job can be submitted for ordinary batch processing by JES and an OS IV/F4 initiator, using special TSS commands.

Compatibility with batch mode
All language processors, data-management access methods, data sets, and data bases used in OS IV/F4 batch processing can be utilized under TSS control. OS IV/F4 Job Control Language (JCL) can be entered by TSS users. Programs and data sets created under TSS can be executed in batch mode, and vice versa.

Improved processing efficiency
Jobs are swapped into/out of main storage according to activity of corresponding terminal users and their priorities for CPU control relative to one another and to batch users. CPU time for TSS users is sliced into relatively short intervals—typically 0.1—1.0 seconds—which are automatically allocated to active users by the TSS Supervisor. No TSS job can execute for longer than one time slice without yielding control to the TSS supervisor, which then decides whether to let this job continue, to switch control to another TSS user, or to yield control to a batch job or other non-TSS task.

Security features
Since TSS is typically used by many terminals simultaneously, it must verify the authority of users to access and/or change programs and data sets. Authorizations are verified by user entered passwords as they log onto TSS, corroborated with their names and other attributes stored in the attributes data set (SYS1.UADS). During execution of his programs, each user has sole possession of a 16-megabytes address space; he can neither access the address spaces of other users nor be accessed by them. Finally, a user can define multiple levels of usage and change access protections by issuing PROTECT commands.

Multiple command processors
Six different command processors operate concurrently in TSS, so as to achieve very quick response times and high system efficiency. **Session control commands** start and stop each session. **System control commands** are utilized by console operators and installation managers. Users issue **data control commands** to generate and edit their data sets, **compiler invocation commands** to call various compilers (including the assembler), **program control commands,** and **batch interface commands.**

Prompters
Prompters are offered with certain language processors (COBOL, FORTRAN, and PL/I compilers, plus the assembler) and the linkage editor and loader. These facilities issue prompting messages to terminal users when the latter are allocating system resources to their programs, such as data sets and devices. Prompting helps reduce programming errors due to inadvertent omission of necessary parameters.

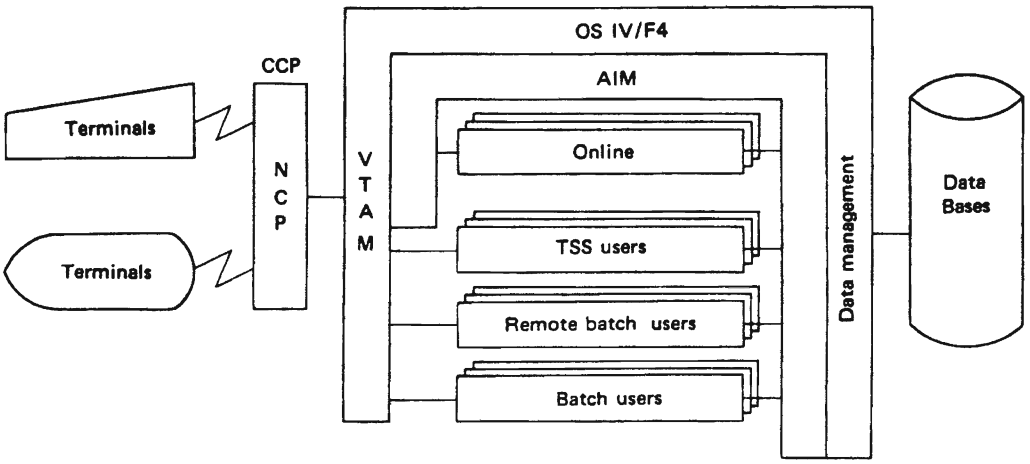


Fig. 4.3 Location of AIM within an OS IV/F4 configuration

Debugging aids
TSS offers **syntax checkers** for the FORTRAN and PL/I languages, which check the syntax of each source statement as it is entered. Whenever the terminal user submits an incorrectly-structured source statement, the syntax checker immediately issues a diagnostic message to his terminal, which permits him to resubmit a corrected statement at once. After he has entered his entire program, he typically requests compilation to locate any residual errors not detected by the syntax checkers; the checkers do not validate inter-statement syntax and semantics of a program.

During execution of a COBOL, FORTRAN, PL/I or Assembler-language program under TSS, the terminal user can request various debugging aids, such as displays of selected statement numbers/labels and values of selected variables as his program executes.

4.11 ADVANCED INFORMATION MANAGER (AIM)

4.11.1 Overview

AIM is the principal data base/data communications (DB/DC) subsystem of OS IV/F4. Fig. 4.3 shows how AIM fits into the hardware/software configuration of a typical OS IV/F4 installation with remote users. The online data base component of AIM is accessible to local/remote batch users. Also, AIM permits several different users (or software subsystems) to concurrently access the same data bases.

4.11.2 Architecture of AIM

AIM comprises a large number of different software functions, each relatively independent of the others:
- data base management.
- program management.
- message management.
- operational management.
- support management.
- languages management.

Each installation can generate AIM with functions relevant to its applications and modes of usage. Each user selects only the subset of AIM functions—implicitly or explicitly—needed for his particular application, processing scale, and mode of access.

4.11.3 Major Components of AIM

Data Base Management
The software component managing a data base in an operating system is called a **data base management system** (DBMS). The OS IV/F4 DBMS manages ordinary data sets and also **data bases,** which are special aggregates of data sets using a consistent and integrated approach. Hence, all permanent data sets on DASD are potentially controlled by the DBMS, whether created by batch, TSS, or other online jobs.

Also, the DBMS supports **shared DASD** (one or more DASDs accessed dynamically by two or more independent configurations) and concurrent access by two or more independent tasks in a single system. The OS IV/F4 DBMS provides exclusive control of physical blocks to users processing shared DASD or concurrently-accessed data bases. This exclusive control is transparent to users.

If a deadlock situation or system failure occurs, the OS IV/F4 DBMS provides facilities for automatically recovering permanent data sets and transaction files in usable form.

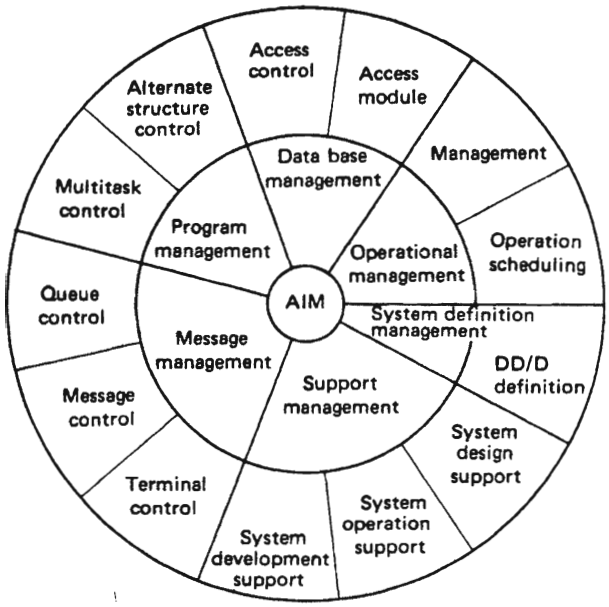


Fig. 4.4 Functional structure of AIM

In addition to conventional sequential and direct organizations, DBMS provides list and ring organizations for data basis; its repertorie of organizations is quite broad.

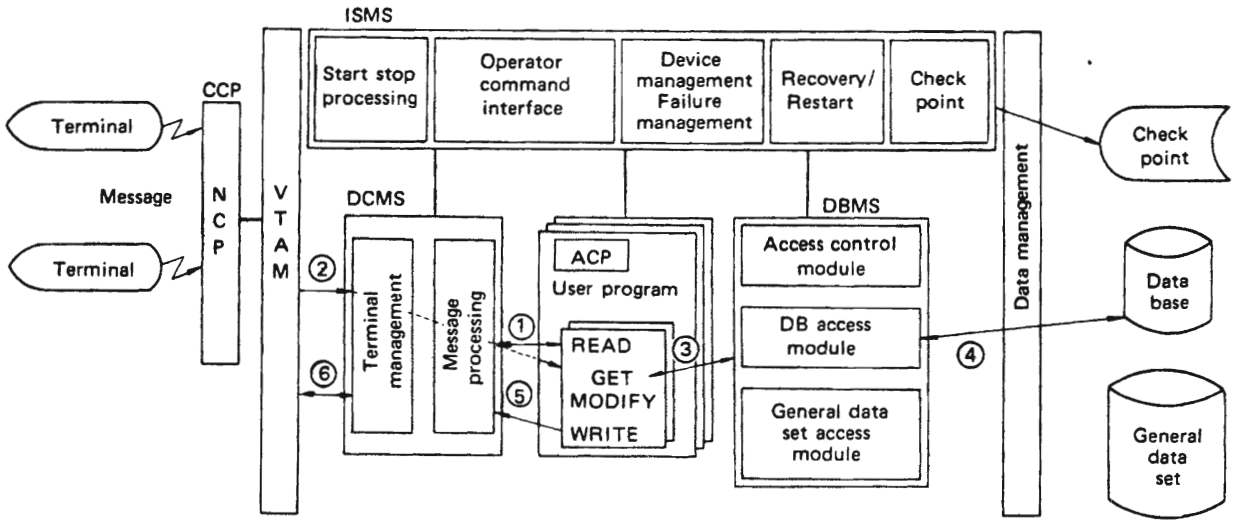
Program management

OS IV/F4 AIM contains an **application control program** (ACP) to manage user programs, which can become quite complex in the AIM environment with respect to resource management, failure recovery, and other online aspects. ACP controls initiation, termination, and failure recovery independently for each user program.

ACP permits each user to initiate his programs much as he requests batch jobs; he need not change his program structures, in general. Several ACPs can operate in a single system, each serving a particular application; if one fails for any reason, the others continue uninterrupted.

Message management

The AIM component which manages **messages** (data and control statements flowing between terminals and the central site) is the **data communication management subsystem** (DCMS). Utilizing the NCP and VTAM, the OS IV/F4 DCMS deter-



① User program asks DCMS to read message.

② LOGON command from terminal connects it to DCMS. Thereafter messages from the terminal flow to DCMS, which edits and translates them, then presents them to the requested programs.

③ User program asks DBMS to process data base.

④ DBMS access module reads records via OS IV/F4 Data Management and transmits them to the requestor. This processing is controlled by the Access Control Module and optionally journals and/or exclusively controls records.

⑤ DCMS edits and translates messages when requested by the user program; when terminal is ready, DCMS asks VTAM to transmit the messages.

⑥ VTAM transmits messages to the terminal via NCP.

Fig. 4.5 Execution flow of the AIM system

mines which messages go to each terminal and the central site, furnishing the same interfaces to user programs as for normal data sets, including the same higher-language interfaces (COBOL, PL/I, etc.). Hence, users can write programs without explicit consideration of AIM or terminals it controls. The DCMS controls all message queues on a unified basis.

Operational management
The AIM component for assisting operators is called the **integrity and schedule management subsystem** (ISMS), which provides for starting AIM operations, restarting after failures, stopping AIM, and responding to operator commands. ISMS attempts to automate most scheduling and operational decisions, using a **data dictionary/directory** (DD/D) data set to keep records of system resources and operations. By changing DD/D, ISMS or a user can modify resources or operating modes without any changes to application programs.

System definition management
AIM provides a **dictionary and directory management subsystem** (DDMS) to furnish generation and maitenance facilities for AIM software.

Support Management
AIM provides support utilities to assist installation managers in designing, generating, operating, and maintaining system and user programs. Support utilities provide various facilities for testing new/revised user programs against AIM data bases and such tools as a simulator and a performance data logger/analyzer for installation managers to model and evaluate versions of AIM.

4.11.4 Execution Flow

Communication between terminals and the central site is depicted in Fig. 4.5 and explained in the following section.

PART 2
CONTROL PROGRAM

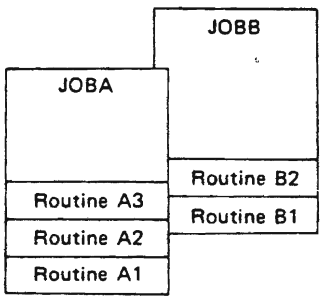
CHAPTER 1
VIRTUAL STORAGE

1.1 BACKGROUND OF VIRTUAL STORAGE SYSTEMS

Virtual storage is simply an address range that can far exceed the actual range of addresses in **real storage** (or **main storage**). To the programmer, virtual storage appears as real storage; therefore, a programmer is able to write programs for the capacity of virtual storage, and the frustration of using limited real storage is greatly diminished.

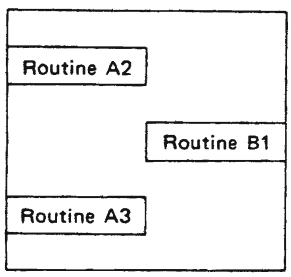
In OS IV/F4 virtual storage is an address range of 16,777,216 bytes (**16 megabytes**). Each user—whether executing a batch job, interactive job, or system component—receives his own copy of virtual storage from OS IV/F4 minus space used for certain system functions.

Programs are actually stored in auxiliary storage called **external page storage**, which is divided into 4K blocks called **slots**; similarly, programs themselves are divided into 4K blocks called **pages**, and



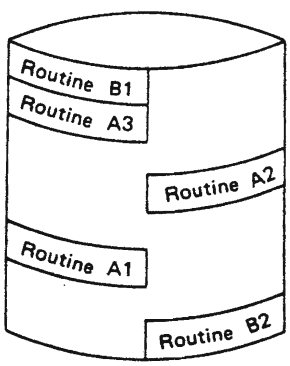
JOBA and JOBB in virtual storage

JOBA and JOBB have private address spaces. To each user, it appears that his job exists in contiguous real storage.



JOBA and JOBB in real storage

Pages containing currently-referenced instructions and data for JOBA and JOBB are in real storage.



JOBA and JOBB in external page storage

Complete images of JOBA and JOBB are retained on auxiliary storage. Instructions and data are transferred to real storage as required for execution.

Fig. 1.1 Relationship between virtual storage, real storage, and external page storage

real storage is divided into 4K blocks called **frames.** The system transfers pages of programs from external page storage to real storage as required during execution, automatically translating virtual storage addresses to actual addresses in real storage. The pages do not ordinarily exist continuously in real storage. This paging activity is transparent to the user. Fig. 1.1 illustrates private address spaces in OS IV/F4 and the relationship between virtual storage, external page storage, and real storage.

This virtual storage design provides more efficient **multiprogramming** (concurrent execution of several large jobs) and a more diverse mix of interactive and batch jobs. Also, since each job is isolated in a private address space, interregion virtual storage fragmentation is eliminated and protection features are extended.

1.2 THE OS IV/F4 VIRTUAL STORAGE ARCHITECTURE

1.2.1 Overview

Virtual storage concepts

In any system with virtual storage, the address space available to programs is limited by the addressing scheme of the central processor rather than the amount of real storage available in the configuration. For example, every M series CPU uses a 24-bit binary address scheme, so an address space as large as 16,777,216 bytes can be supported.

With this design, a virtual storage system can support an address space large than the actual amount of real storage available. To accomplish this, the OS IV/F4 control program stores the contents of virtual storage—instructions and data—onto direct access storage; it brings instructions and data into real storage (from direct access storage) only when required by executing programs. Likewise, the control program returns altered instructions and data to direct access storage when the real storage they occupy is needed and they are no longer being used. Thus, at any time, real storage contains only a portion of the contents of virtual storage.

Virtual storage in OS IV/F4

Virtual storage is divided into address spaces for the control program and address spaces for user programs. Each user job (and some system components) receives its own private user address space. That is, OS IV/F4 supports **multiple address spaces,** as shown in Fig. 1.2.

The control program creates private address spaces for the following users and system components:

- Each batch job scheduled by an initiator.
- Each logged-on time-sharing job.
- The master scheduler.
- The job entry subsystem (JES).
- The virtual telecommunications access method (VTAM).
- Every program initiated by a START command.

1.2.2 Virtual Storage Layout

Although each user job is given its own private address space, it does not have control over all of it—each address space is divided into the system

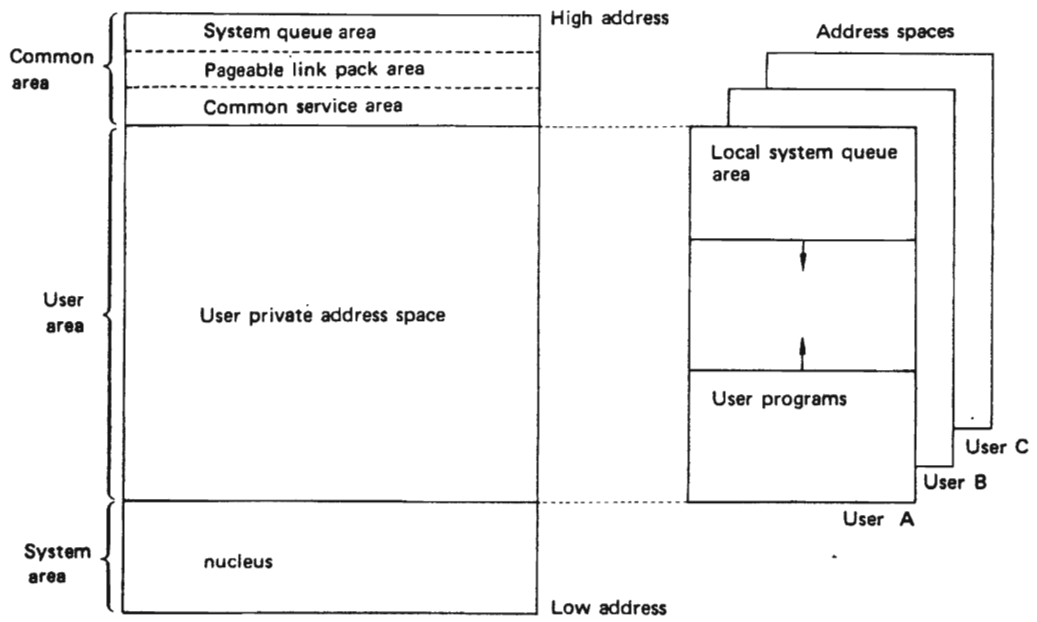


Fig. 1.2 Virtual storage layout

area, the user area, and the common area (see Fig. 1.2).

The **system area** contains the nucleus, which is fixed in real storage that is mapped into the low addresses of each user's address space. The **common area** corresponds to the highest addresses of virtual storage. The common area contains the **system queue area** (SQA), the **pagable link pack area** (PLPA), the **modified LPA** (MLPA), the **pagable BLDL list** (PBLDL), and the **common service area** (CSA). The SQA contains tables and queues relating to the entire system. The three LPAs contain SVC routines, access methods and other read-only system programs, and any reentrant (read-only) user programs selected by the installation that can be shared among users of the system. The CSA contains data for communications among the private user address spaces.

Each user's private address space begins immediately after the nucleus and extends up to the common area. Thus, all users have the same amount of private address space. Fig. 1.2 shows the structure of the user address space in virtual storage. Space is assigned to user programs from the low address up. Space is assigned from the high address down for the **local system queue area** (LSQA) which contains tables and queues associated with the user's job and address space. The remainder of the private address space is available for its user, with space being allocated from the low address up.

Some user programs must remain in real storage during execution. These programs are assigned discontiguous real-storage pages, just like other programs in virtual storage. Hence, their pages are not written to DASDs during execution. (Each real-storage execution is specified for the duration of one job step.) Since M series CPUs and channels utilize dynamic address translation, **real-storage programs** can execute at full speed without any page-fault interruptions. However, their address spaces are considerably smaller than 16 megabytes, the standard size for pagable programs, since they must have a real-storage page frame for each program page.

1.2.3 Storage Organization

For ease in storage management, virtual storage, real storage, and direct access storage containing virtual-storage images are divided into contiguous fixed length sections of equal size.

Virtual storage is divided into 64K byte segments. (A maximum virtual storage of 16,777,216 bytes, therefore, contains 256 segments.) Each segment of virtual storage is divided into 4K-byte virtual storage pages; thus, each segment contains 16 pages.

Real storage is divided into 4K-byte page frames. Hence, a page frame is a block of real storage that

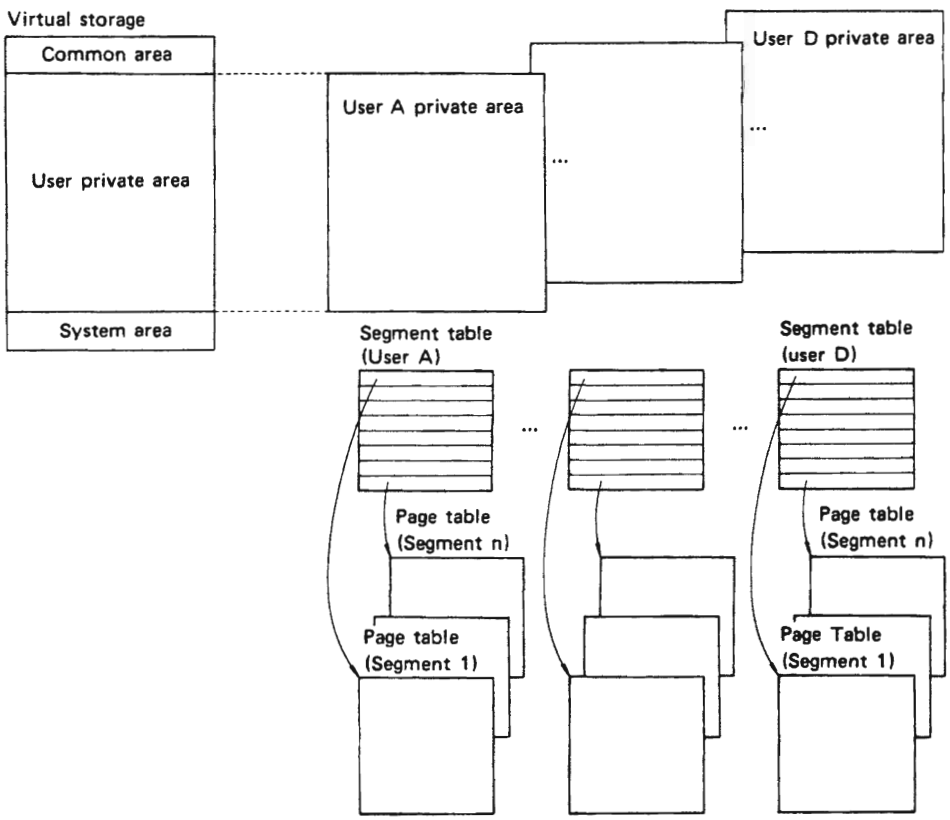


Fig. 1.3 Segment and page tables

can contain one page at a time.

The direct access storage used to contain virtual storage contents is called **external page storage.** External page storage is divided into physical records called slots. A slot is 4K bytes long; therefore, each slot can contain one page at a time.

In summary, a page of data or instructions is assigned a virtual storage address. The page occupies a slot when it is in external page storage, a frame when it is in real storage.

Address translation

When coding a program the user refers to data and instructions by names or labels without knowing their physical addresses. In a virtual storage system, the control program assigns to each name or label a virtual storage address that can be used to locate the data or instruction. By comparison, real storage addresses are actual physical locations in processor storage where data and instructions can be placed for processing by the CPU.

A mechanism is required to associate virtual storage addresses of data and instructions with their actual locations in real storage. Transformation of a virtual storage address to its real storage address is **address translation.** In M series computers, the **dynamic address translation** (DAT) hardware feature in the CPU and channels performs address translation.

To translate the addresses, DAT uses tables in real storage. These tables, which are maintained by the control program, are the segment table and a number of page tables. One segment table and a corresponding set of page tables exist for each address space in the system, as shown in Fig. 1.3.

A **segment table** contains one entry for each segment in the address space that the table describes. A segment table entry defines the number of pages allocated in the segment and points to the real storage location of the page table for the segment.

There is one **page table** for each segment in the address space. A page table contains one entry for each page in the associated segment. It indicates which pages are currently in real storage and the real stoorage locations of these pages. As pages are transferred between real and external page storage, the control program changes the corresponding page table entries.

DAT translates any virtual storage addresses referenced by an instruction during its execution. Translation occurs after the 24-bit effective virtual storage address has been computed, as usual, by adding base, displacement, and any index values together. The format of the effective virtual storage address is included in Fig. 1.4.

The translation process is shown in Fig. 1.4. First, DAT obtains the address of the appropriate segment table from a system control register. To this segment

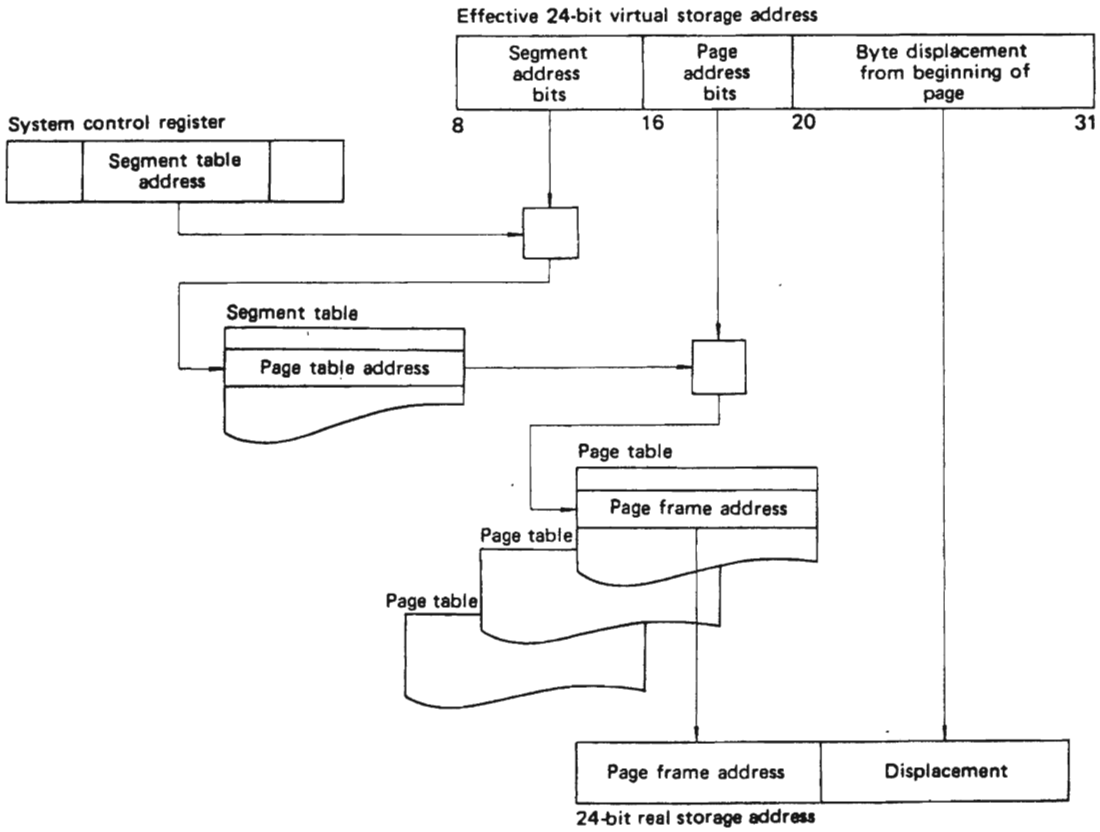


Fig. 1.4 Dynamic address translation procedure

table address, DAT adds the segment address bits (from the effective virtual storage address), to obtain the segment table entry. Next, DAT obtains the page table address from the segment table entry and adds the page address bits to it in order to obtain the page table entry. Finally, DAT forms the 24-bit real storage address by appending the displacement (from the effective virtual storage address) to the page frame address in the page table entry.

To reduce the amount of time required for address translation, DAT retains up to 128 previously-translated addresses in a **translation lookaside buffer** (TLB). Prior to performing a translation using segment and page tables, DAT searches the TLB for the required translated address.

Paging

A program interruption occurs during address translation if DAT attempts to translate a virtual storage address to a real storage address and the required page is not in real storage. This interruption, called a **page translation exception** or **page fault**, alerts the control program that the page must be loaded from external page storage into a page frame of real storage.

Data and instructions are transferred between external page storage and real storage as needed, page by page. This activity is called **paging**. The control program routine responsible for paging is **real storage management**.

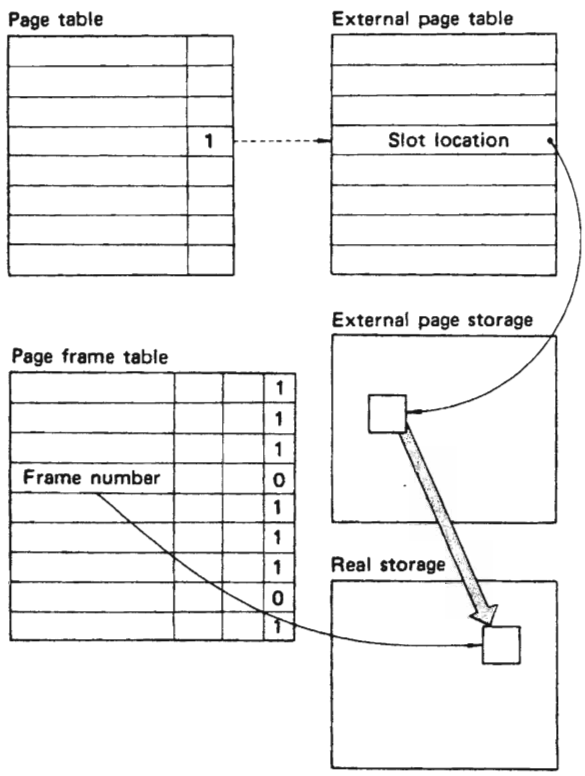


Fig. 1.5 Page-in process

The transfer of a page into real storage is a **page-in**, as shown in Fig. 1.5. When a requested page is not in real storage (indicated by a bit in its page table entry), real storage management examines the corresponding entry in an external page table. (One external page table corresponds to each page table in the system.) The external page table entry gives the slot location for the page.

Next, storage management selects a frame in real storage to hold the required page. To do so, it refers to the **page frame table**, that indicates which frames are allocated. Storage management finds an available frame and brings in the required page from its slot in external page storage. To complete the page-in process, storage management updates the appropriate pageframe-table entry and page-table entry.

To keep a supply of frames available for page-in, the OS IV/F4 control program removes pages from real storage that have not been recently referenced. Prior to removing a page from a frame, the control program determines whether its contents were modified during processing. If so, storage management performs a page-out; otherwise, an exact copy of the page already exists in external page storage. A **page-out** copies the modified page from its realstorage frame to a slot in external page storage; the slot need not be the one that contains the old version of the page. Storage management need only update the external page table entry to designate the new slot.

For various reasons, certain pages should not be paged out of real storage. For example, pages that contain I/O buffers must remain in real storage while the buffers are being referenced during an I/O operation. A page that cannot be paged out is called a **fixed page**.

Pages that are fixed for the duration of a job or job step are **long-term fixed**. For example, pages that contain certain control blocks related to a job are long-term fixed for the duration of the job. Pages that must be fixed for only a portion of the time they are in real storage are **short-term fixed**. For example, a page containing an I/O buffer is fixed prior to the start of the I/O operation; after the I/O operation is completed, the page is unfixed and is eligible for page-out. Only the control program can selectively fix user pages. For instance, the control program can short-term fix user I/O area pages.

The user can fix pages, but not selectively, by coding ADDRSPC=REAL on his JOB or EXEC statements. Each such job step is allocated a collection of real-storage page frames, which the DAT and Channel-DAT features make logically contiguous to each other and to the nucleus. Since these programs are not paged, they do not occupy external page storage. The entire program is loaded into real storage when it is initiated, and all pages are fixed.

Swapping
If an interactive task remains in Wait state for a long interval—for example, while a terminal user is deciding what calculation to perform next—the OS IV/F4 Supervisor will page out all pages of his address space (that is, those unshared with other users). This technique is called **program swapping** (or merely **swapping**). It is no more than anticipatory paging out of all pages held by a user who is expected to be inactive for several seconds or minutes.

The same technique is used when a initiated batch job must await operator invention, for example when an executing program has issued a Write to Operator with Reply (WTOR) macro instruction, COBOL ACCEPT instruction, etc. Since OS IV/F4 knows that the executing program cannot proceed until the console operator replies, and since several seconds will typically elapse until the operator enters the desired response, the OS IV/F4 Supervisor swaps out this user's address space. Similarly, if a volume must be mounted at the beginning of a job step or—as requested by a DEFER option in a JCL UNIT parameter—in the middle of a step, OS IV/F4 will swap out this address space as soon as it has issued the volume-mounting command.

Swapping immediately furnishes a number of page frames for allocation to other jobs. Hence, successful swapping permits other active jobs to run with fewer page faults; it even permits OS IV/F4 to start additional batch jobs.

Levelling paging activity
In OS IV/F4, page frames are not mapped one to one onto slots, nor are the pages of each active address space rigidly allocated to slots. Hence, the OS IV/F4 Supervisor can store a page into any available slot when page-out becomes necessary. Since the choice of slots can be delayed until the last instant, the OS IV/F4 Supervisor can select a lightly-loaded channel or paging device each time.

Slot sorting
The OS IV/F4 Paging Supervisor issues I/O requests to paging devices whenever pages must be transferred to/from main storage. The channel programs associated with these requests are dynamically modified by the Supervisor so long as additional pages must be read/written to corresponding devices. Hence, slots on these devices can be chosen at the last instant. Whenever several input and/or output requests for pages are outstanding for the current cylinder on a paging device, OS IV/F4 satisfies them by their physical sequence on the DASD, rather than by their order of issuance or another queuing algorithm. ("Physical sequence" is in terms of their angular displacements from the current position of the DASD read/write mechanism.) This **slot sorting** technique maximizes the average transfer rate of pages to/from main storage, which keeps paging bottlenecks to a minimum.

Preventing paging overloads
No matter how efficiently paging is managed, the Supervisor can only transfer a limited number of pages per second. If executing programs (including timesharing and system tasks) request pages faster than this rate—summed over all paging devices—their throughput declines. For interactive tasks, paging overloads can cause drastically longer response times at terminals. This condition is often called **thrashing**.

To prevent thrashing—or to correct it after it begins—the OS IV/F4 Supervisor will swap out one or more low-priority batch jobs. By thus suspending their execution, OS IV/F4 releases their page frames for use by other tasks. Also, the CPU time and channel capacity they would have used can be released to other jobs. If necessary, OS IV/F4 will suspend higher-priority batch jobs in order to reduce thrashing. When thrashing has subsided sufficiently, the OS IV/F4 Supervisor will swap in one suspended task after another, in order to keep the paging rate acceptably low.

Paging hierarchies
If an installation assigns two or more different DASDs for paging, OS IV/F4 will use each as appropriate to its speed and capacity. For example, if an installation allocates one F6625 Drum and two F478B Disk drives for paging, the paging supervisor will utilize the drum fully prior to writing any pages onto the disk drives. Thereafter, the Supervisor will write frequently-referenced pages (e.g., from the Pagable link pack area, which is shared by all users) onto the drum and infrequently-referenced pages onto the disk drives. Since the drum is much faster than the disk drives but has much less storage capacity, this strategy maximizes the overall paging rate.

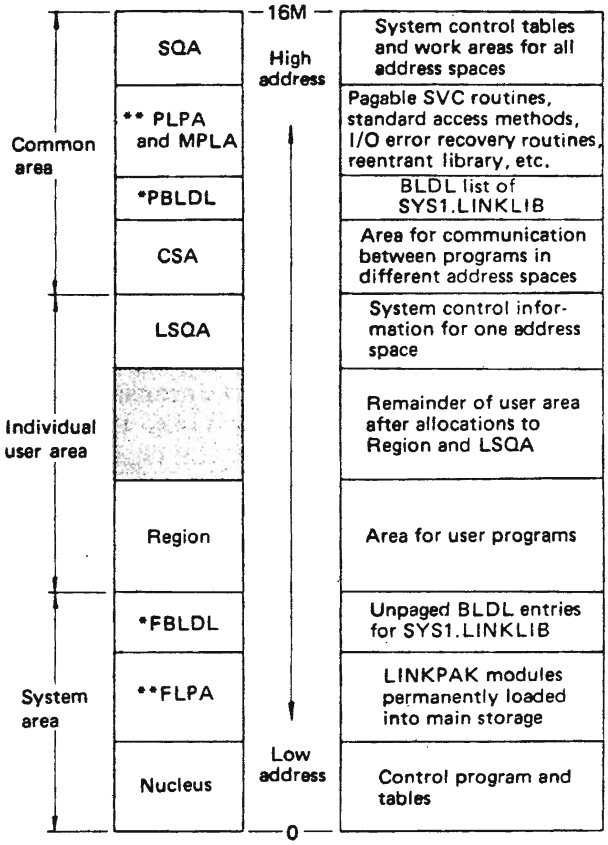
Real-storage regions
For time-dependent programs and other tasks requiring ultra-fast response times to I/O requests, users can request **real-storage regions** by coding ADDRSPC=REAL on their JOB or EXEC statements. All pages for such jobs (or job steps) are fixed in main storage throughout execution. Unlike several prior hardware systems and corresponding operating systems, OS IV/F4 utilizes discontiguous page frames to define a real-storage region; the combination of CPU-DAT and Channel-DAT features permits all instructions, CCWs, and data areas to be in virtual storage. The only differences between real-storage reegions and other virtual-storage address spaces are that the former are not paged out or limited in size, since each of their pages requires one main-storage page frame throughout its job step. Paging overhead is completely eliminated for real-storage regions, but typically only a few (if any)

should be allocated at one time since they cause drastic depletion of the pool of page frames available to other jobs and the OS IV/F4 control program.

Macro instructions for fixing/freeing pages
For some Assembler language programs, it is useful to deliberately fix certain pages in main storage for certain intervals—not necessarily throughout an entire job step. For example, a program may use a scatter-storage (hashing) algorithm to sort data or perform another computational task that requires a large address space briefly. In this case, the program can optimize its efficiency by reducing/eliminating paging of certain program or data areas. The program requests that these pages be fixed by issuing PGFIX macro instructions; later, it can free these pages for paging by issuing PGFREE macro instructions. PGFIX and PGFREE macro instructions can be issued only by previously-authorized programs since they can—if misused—seriously degrade paging performance and system throughput.

1.2.4 Structure of OS IV/F4 Address Spaces

Each address space contains common elements, which are shared among all address spaces (including those for interactive and system tasks). These elements are the system area and the common area. The remainder of the address space contains pages which are uniquely accessible by this address space so long as it executes. Fig. 1.6 outlines a typical address space, Figs. 1.7 to 1.10 provide details on various shared areas in this address space.



* FBLDL and PBLDL are defined exclusive of each other within a system. Selection of which to use can be made at the time of IPL.

** Ditto for FLPA, PLPA, and MLPA

Fig. 1.6 Typical OS IV/F4 Address space

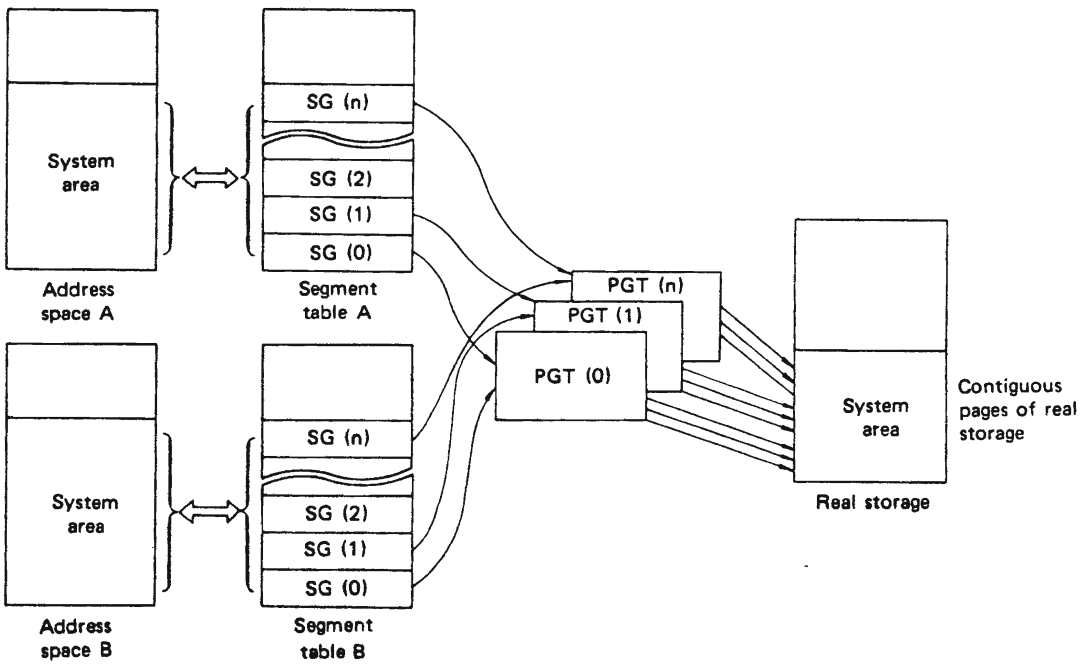


Fig. 1.7 Address mapping for the system area

System Area

The system area comprises the system nucleus and, optionally, a **fixed link pack area** (FLPA) and **fixed BLDL table** (FBLDL). The system area is identical for all address spaces with respect to size and contents. It is contained in contiguous page frames at the lowest addresses in real storage. The system area is not paged, and its storage addresses are not translated into virtual addresses, as illustrated in Fig. 1.7.

The **nucleus** contains basic control-program routines which are loaded when OS IV/F4 is initially started (IPL). The FLPA contains reentrant programs and read-only tables intensively and simultaneously used by many tasks; they are a subset of the total link pack area (LPA), comprising reentrant load modules which are heaviest used:

- some/all pagable SVC routines.
- some/all standard access method-modules.
- I/O error recovery routines.
- Reentrant libraries used by language processors.
- User-generated reentrant load modules.

The FLPA can be included or excluded each time OS IV/F4 is loaded.

The FBLDL is the image of part/all of the directory for the system link library (SYS1.LINKLIB), which contains the OS IV/F4 Assember, Linkage editor, Loader, various service aids, and user-written programs which are accessed frequently. Each time OS IV/F4 is loaded, the operator can select whether to include/exclude an FBLDL area, which serves to speed up retrieval of LINKLIB members named in FBLDL. In the common area, a pagable BLDL table can optionally be created which is complementary to the FBLDL. Neither, either, or both can be selected by the operator, although he typically accepts default values for these tables set during OS IV/F4 system generation.

Private user area

Each private user area comprises a **local system queue area** (LSQA) and a region. The LSQA contains control information specific to this address space such as its segment table, part/all of its page table, and various control blocks for its tasks, as shown in Fig. 1.8.

An LSQA is allocated for each address space—an integral number of segments of 64K bytes each—when the corresponding initiator is started by the operator.

The **region** is the area in each address space where programs are loaded and executed, plus associated workspaces. Its size can be optionally limited by the REGION parameter on a JOB or EXEC statement. If the user specifies ADDRSPC=REAL on his JOB or EXEC statement, he must furnish a REGION parameter unless a default value is assigned by the associated initiator procedure, by an SMF exit routine, etc. As discussed in the previous section, ADDRSPC=REAL selects a real-storage region, used primarily for time dependent jobs and others with critical response time requirements.

If the user specifies ADDRSPC=VIRT (the OS IV/F4 default value), his region is allocated pagable storage in integral segments (64K blocks).

Common area

Like the system area, the common area is shared among all address spaces; unlike the system area, it is pagable. It contains the system queue area (SQA), pagable and modified link pack areas (PLPA and MLPA), pagable BLDL list (PBLDL), and a common service area. One set of page tables exists for the common area, shared by all address spaces just as they share the unique set of page tables for the system area; this structure is indicated in Fig. 1.9.

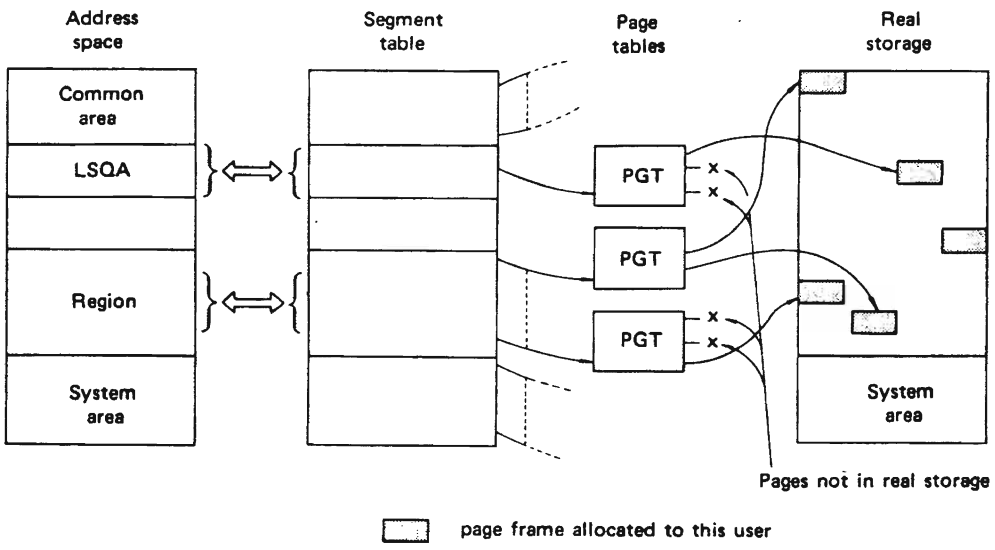


Fig. 1.8 Address mapping for a private user area

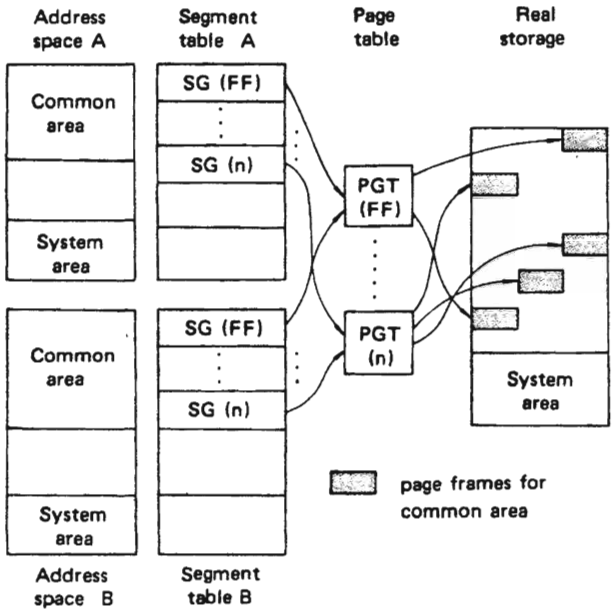


Fig. 1.9 Address structure for the common area

System queue area (SQA)
The SQA contains control blocks and tables for the entire system, rather than for individual address spaces. When OS IV/F4 is reloaded (IPL), the Supervisor allocates segments (blocks of 64K bytes) to SQA, as pre-determined during system generation and optionally modified by the console operator. SQA segments are allocated downward from the top of each address space (location 16,777,215 for all virtual-storage regions). Just as it handles the LSQA, the Supervisor allocates page frames to these segments only as corresponding addresses are referenced by active tasks.

Pagable and modified link pack areas (PLPA and MLPA)
Reentrant load modules used by all address spaces are retrieved into these LPAs from the system link pack library (SYS1.LPALIB):

- Pagable SVC routines.
- Standard access methods.
- I/O error recovery routines.
- Reentrant libraries used by language processors.
- User-generated reentrant load modules.

Designated load modules from SYS1.LPALIB may optionally be loaded into the FLPA; all others are loaded into the PLPA, as selected at IPL time.

Pagable bLDL list (PBLDL)
Just as for the Fixed BLDL table, the PBLDL is created when OS IV/F4 is reloaded, as an image of part/all of the directory for the system link library (SYS1.LINKLIB). Some (or no) LINKLIB directory entries may be in FBLDL, some (or no) directory entries in the PBLDL, and the remaining entries are retrieved from DASD as needed.

Common service area (CSA)
The CSA is used by address spaces to communicate with one another; this includes such systems tasks as JES and VTAM as well as user address spaces. CSA is allocated in integral segments (blocks of 64K bytes) when OS IV/F4 is reloaded. Address spaces allocate and release space in CSA by means of system macro instructions. Page frames are allocated only as necessary; the CSA is fully pagable.

1.2.5 Processing Jobs in Virtual Storage

This section explains how OS IV/F4 manages address spaces and jobs prior to/during job processing.

Prior to loading a user program
The console operator starts an initiator with a START command, to which OS IV/F4 responds by creating a new address space together with its LSQA. A page frame is allocated within the new LSQA to hold the segment table and first page table for this address space. Other necessary control blocks are then created in the LSQA, as shown in Fig. 1.11.

Prior to processing a job, the initiator reserves sufficient external slots to contain all of its pages, based on explicit REGION parameters of its JOB or EXEC statements or the default region size for this initiator. The region is allocated virtual addresses beginning just above the highest address of the system area.

The programs named in EXEC statements must all be members of **program libraries,** partitioned data sets in a special format created by the linkage editor. The directory of each library contains the names and sizes of all member programs. The Supervisor first reads the directory, allocates sufficient virtual storage to hold the requested program, and loads the program into this storage area. If sufficient page frames exist, the entire program can be loaded into real storage; otherwise, page-out activity begins while the program is being loaded, as shown in Fig. 1.12.

Page management during execution
After a program has been successfully loaded, it begins execution. Under OS IV/F4, most installations operate several initiators plus TSS, VTAM , AIM, etc. Each initiator may receive virtual-storage requests exceeding the number of allocatable page frames; the total of all outstanding requests (and consequent allocations) of virtual storage typically exceeds the total real storage of the system by a large factor, forcing the OS IV/F4 Supervisor to continually page out inactive pages from one or more address spaces.

Corresponding to each page of each address space is a **page table entry**, one of whose flag bits is the

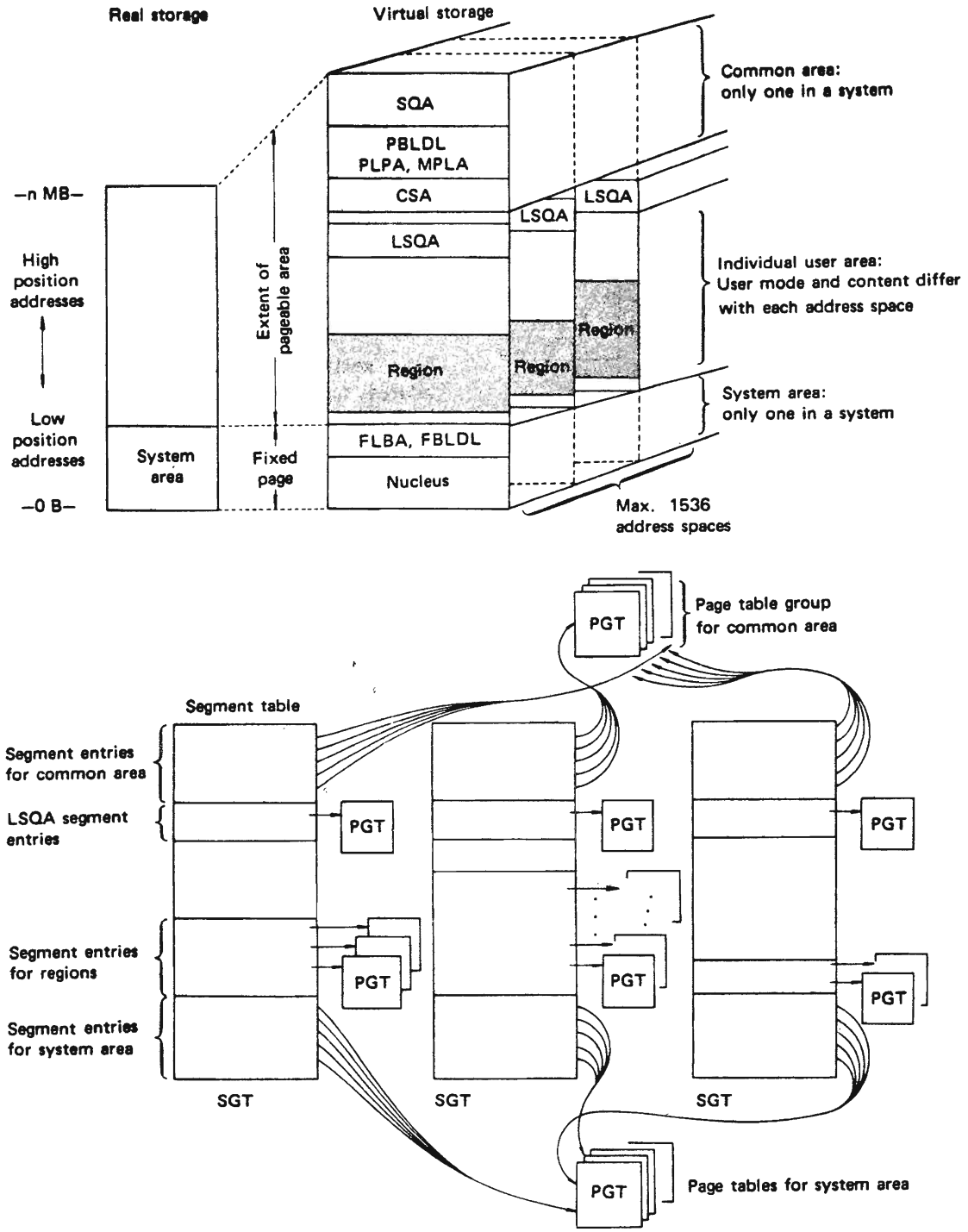


Fig. 1.10 Overall addressing of an address space

invalid bit. Whenever the CPU references a virtual storage address, DAT hardware automatically references the corresponding page table entry and tests its invalid bit. If on, there is no page frame currently allocated to this page, and the DAT hardware automatically generates a **page fault interruption** (or **page fault**).

Whenever a page fault occurs, the OS IV/F4 Supervisor gains control and places the corresponding task into wait state until the page satisfying this request is brought into a main-storage page frame. If an unallocated page frame already exists—in the pool of such frames controlled by the supervisor—the requested page is read into this frame,

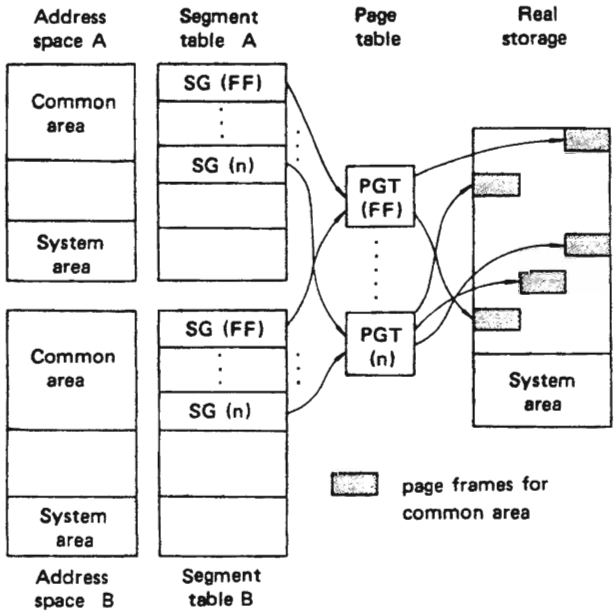


Fig. 1.9 Address structure for the common area

System queue area (SQA)
The SQA contains control blocks and tables for the entire system, rather than for individual address spaces. When OS IV/F4 is reloaded (IPL), the Supervisor allocates segments (blocks of 64K bytes) to SQA, as pre-determined during system generation and optionally modified by the console operator. SQA segments are allocated downward from the top of each address space (location 16,777,215 for all virtual-storage regions). Just as it handles the LSQA, the Supervisor allocates page frames to these segments only as corresponding addresses are referenced by active tasks.

Pagable and modified link pack areas (PLPA and MLPA)
Reentrant load modules used by all address spaces are retrieved into these LPAs from the system link pack library (SYS1.LPALIB):

- Pagable SVC routines.
- Standard access methods.
- I/O error recovery routines.
- Reentrant libraries used by language processors.
- User-generated reentrant load modules.

Designated load modules from SYS1.LPALIB may optionally be loaded into the FLPA; all others are loaded into the PLPA, as selected at IPL time.

Pagable bLDL list (PBLDL)
Just as for the Fixed BLDL table, the PBLDL is created when OS IV/F4 is reloaded, as an image of part/all of the directory for the system link library (SYS1.LINKLIB). Some (or no) LINKLIB directory entries may be in FBLDL, some (or no) directory entries in the PBLDL, and the remaining entries are retrieved from DASD as needed.

Common service area (CSA)
The CSA is used by address spaces to communicate with one another; this includes such systems tasks as JES and VTAM as well as user address spaces. CSA is allocated in integral segments (blocks of 64K bytes) when OS IV/F4 is reloaded. Address spaces allocate and release space in CSA by means of system macro instructions. Page frames are allocated only as necessary; the CSA is fully pagable.

1.2.5 Processing Jobs in Virtual Storage

This section explains how OS IV/F4 manages address spaces and jobs prior to/during job processing.

Prior to loading a user program
The console operator starts an initiator with a START command, to which OS IV/F4 responds by creating a new address space together with its LSQA. A page frame is allocated within the new LSQA to hold the segment table and first page table for this address space. Other necessary control blocks are then created in the LSQA, as shown in Fig. 1.11.

Prior to processing a job, the initiator reserves sufficient external slots to contain all of its pages, based on explicit REGION parameters of its JOB or EXEC statements or the default region size for this initiator. The region is allocated virtual addresses beginning just above the highest address of the system area.

The programs named in EXEC statements must all be members of **program libraries,** partitioned data sets in a special format created by the linkage editor. The directory of each library contains the names and sizes of all member programs. The Supervisor first reads the directory, allocates sufficient virtual storage to hold the requested program, and loads the program into this storage area. If sufficient page frames exist, the entire program can be loaded into real storage; otherwise, page-out activity begins while the program is being loaded, as shown in Fig. 1.12.

Page management during execution
After a program has been successfully loaded, it begins execution. Under OS IV/F4, most installations operate several initiators plus TSS, VTAM , AIM, etc. Each initiator may receive virtual-storage requests exceeding the number of allocatable page frames; the total of all outstanding requests (and consequent allocations) of virtual storage typically exceeds the total real storage of the system by a large factor, forcing the OS IV/F4 Supervisor to continually page out inactive pages from one or more address spaces.

Corresponding to each page of each address space is a **page table entry**, one of whose flag bits is the

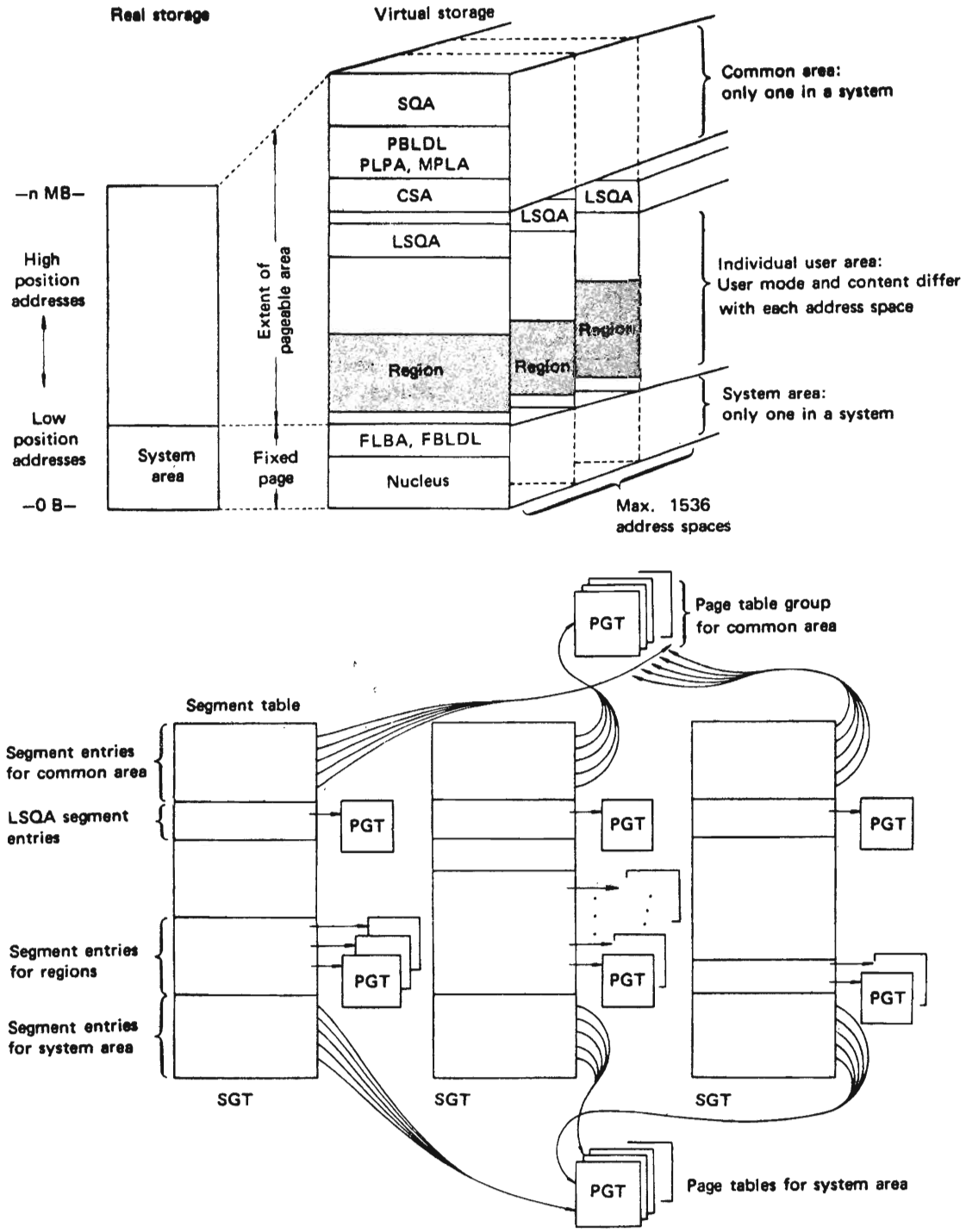


Fig. 1.10 Overall addressing of an address space

invalid bit. Whenever the CPU references a virtual storage address, DAT hardware automatically references the corresponding page table entry and tests its invalid bit. If on, there is no page frame currently allocated to this page, and the DAT hardware automatically generates a **page fault interruption** (or **page fault**).

Whenever a page fault occurs, the OS IV/F4 Supervisor gains control and places the corresponding task into wait state until the page satisfying this request is brought into a main-storage page frame. If an unallocated page frame already exists—in the pool of such frames controlled by the supervisor—the requested page is read into this frame,

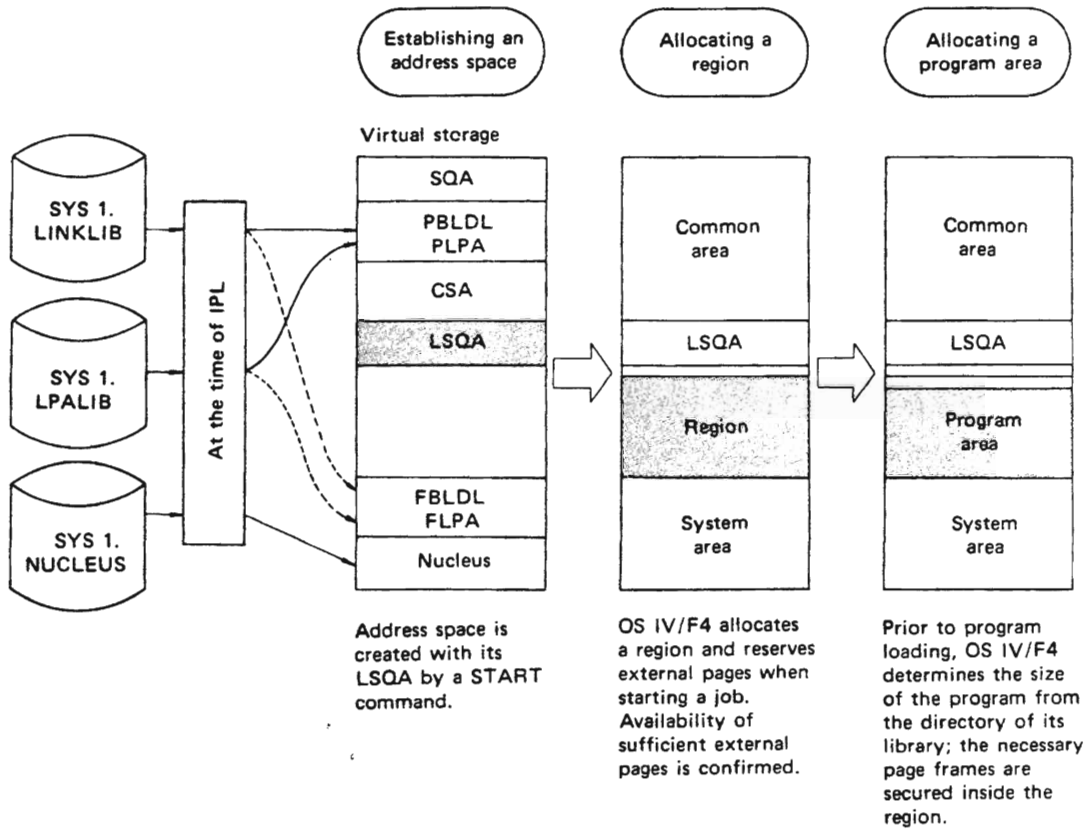


Fig. 1.11 Creating a new address space

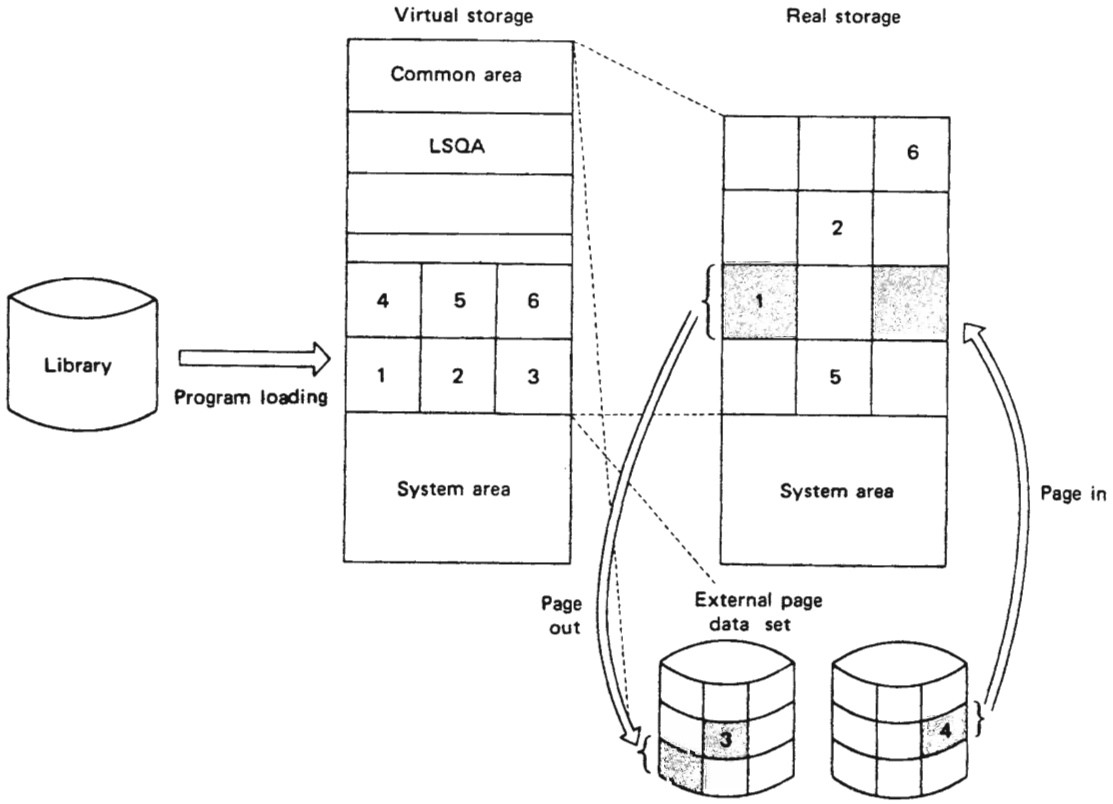


Fig. 1.12 Loading and execution of a program

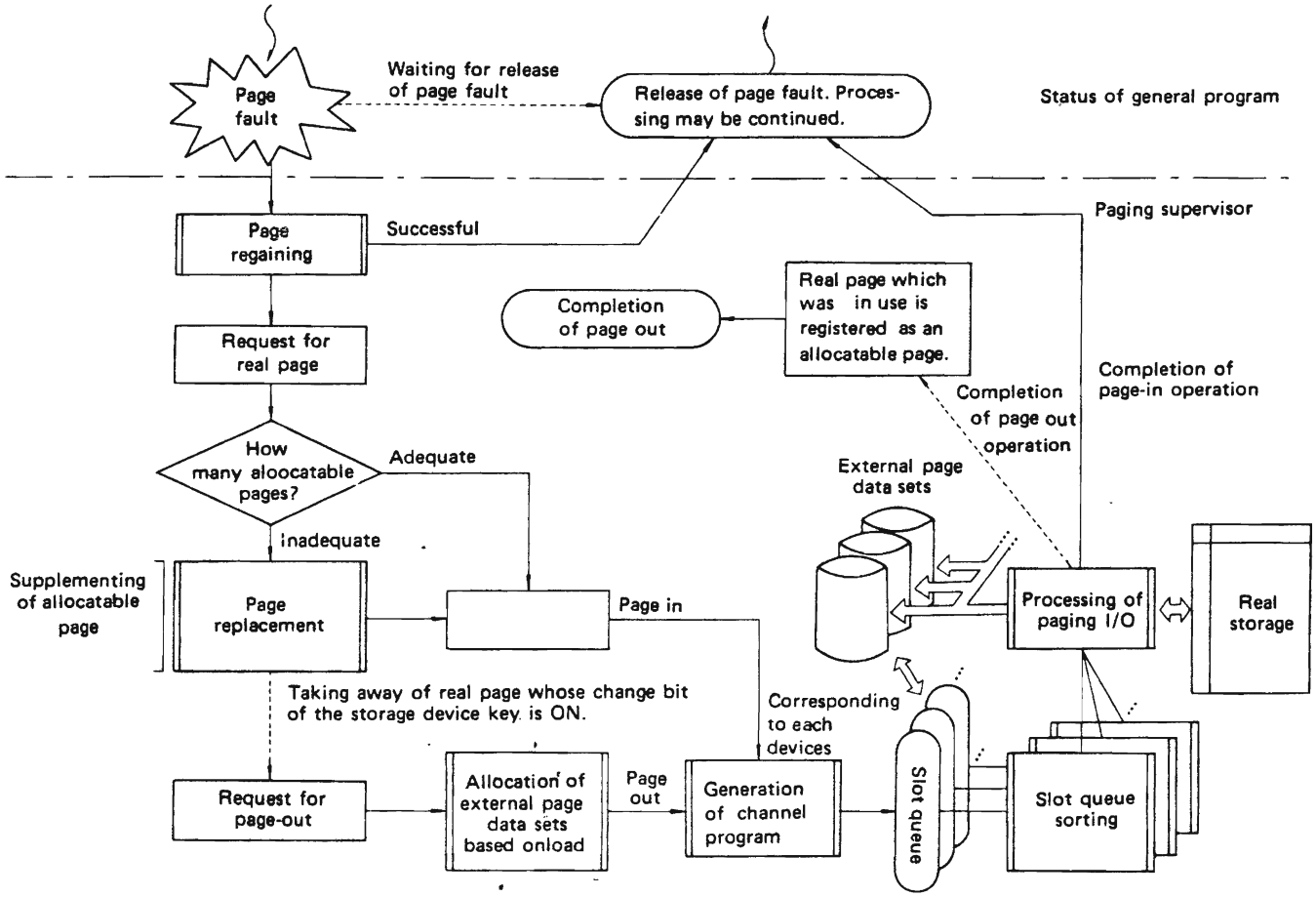


Fig. 1.13 General flow of paging process

whereupon the supervisor inserts the address of this page frame into the page table entry for this address space. Also, the invalid bit for this entry is turned off. The supervisor then returns to the interrupted program, which reattempts the instruction causing the page fault. This sequence is shown in Fig. 1.13.

Should no unallocated page frame be available, the OS IV/F4 Supervisor must create one or more empty page frames. (It will typically attempt to reclaim several frames at once, so as to meet future page frame needs by this task and others. The reader should review the preceding subsection on "Swapping" for a discussion of how/why multiple frames are released.) If an in-use page has not been referenced for several seconds, it is eligible for page-out. The OS IV/F4 Supervisor selects one or more such pages and examines **reference bits** and **change bits** in their corresponding page table entries. If any page has its reference bit off, this page has not been recently referenced and is a candidate for page-out. If it also has its change bit off, the page has not been altered by a CPU or channel since it was last loaded into a main-storage page frame. An identical copy of this page already exists on a paging data set, and the main-storage copy need not be paged out. In other words, the current main-storage copy can be **discarded** (its frame allocated to another page); when AND if this page is again referenced, the identical copy can be retrieved from the page data set.

When the reference bit for a page is off but its change bit is on, this page has been previously altered but not recently referenced. If this page is selected for paging-out, the OS IV/F4 Supervisor must copy it to an empty slot, then add its page frame to the pool of unallocated frames.

In this way, OS IV/F4 uses a combination of discarding unchanged pages and paging out changed pages to create empty page frames. When a job step completes, all of its page frames are released to the pool by the OS IV/F4 Step Terminator. This overall technique is depicted in Fig. 1.13.

1.3 CHANNEL DYNAMIC ADDRESS TRANSLATION

Channel DAT is a feature of M series computers which considerably advances the state of art for virtual storage systems. OS IV/F4 fully supports this feature, which facilitates allocation of real-storage regions (described in Section 1.2.2) to discontiguous

page frames. In prior virtual storage operating systems, real-storage regions often required contiguous page frames with "virtual addresses" identical to real-storage addresses, so that channels without DAT capability could read/write into main storage without suffering page faults.

Virtual-storage addresses for channel programs

A **channel program** comprises one or more **channel command words** (CCWs) chained together. A channel program is analogous to a CPU program, as shown in Fig. 1.14.

Just as a CPU accesses virtual storage addresses and dynamically translates them into real storage addresses, an M series channel accesses CCWs in virtual storage, whose operand and data addresses are also in virtual storage, and dynamically translates all of these addresses into real storage addresses.

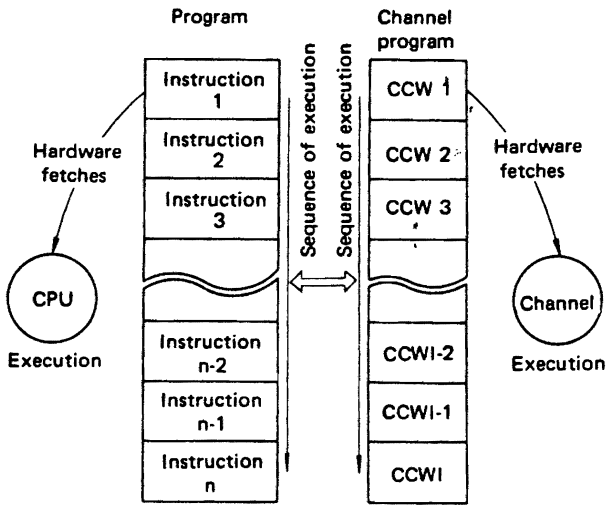


Fig. 1.14 Analogies between CPU program and channel program

Problems of non-DAT channels

In prior virtual-storage hardware, channels often lacked DAT features; hence, their CCWs had to be addressed directly in real storage, and their operand and data addresses also had to reference real storage. Furthermore, these real storage addresses often had to be allocated to corresponding tasks for the duration of a job step (or interactive session), since channels could not sense when "page frames" were "reallocated" to other tasks. CPUs could sense such reallocations, but channels could not.

Problems in retrieving CCWs

If a channel program were created in virtual storage by a CPU, it might straddle two or more page frames. If such frames were reallocated by the control program, a non-DAT channel would be unaware of this reallocation and would try to retrieve CCWs from an unknown area, causing an undiagnosable software error.

A technique often used by recent operating systems is to fix all page frames containing channel programs (or parts of channel programs) so that their contents are guaranteed. This preempts a substantial number of page frames merely to hold channel programs.

Problems in referencing addresses within CCWs

Operand and data addresses within CCWs refer to various page frames. Non-DAT channels are unable to detect when these frames are reallocated to other pages, and they would read and write unknown operands and data if these frames were reallocated. Hence, operating systems supporting non-DAT channels have typically been forced to fix all pages containing operands and data referenced by CCWs for the duration of a job step — or at least, for the duration of an entire I/O request.

Problems in stradding page frames

A further complication for non-DAT channels is that initial addresses within CCWs do not indicate all possible page frames into which reading AND writing may take place; a channel may reference two or more pages when executing a particular CCW. Hence, a virtual storage operating system supporting non-DAT channels must investigate all channel programs to determine how many page frames are accessed by each CCW.

Typical solutions to problems with non-DAT channels

Operating systems supporting channels without DAT capability typically "solve" the above problems as follows:

- Copy all channel programs from virtual storage to a page-fixed area controlled by the operating system.
- Calculate real-storage addresses for all CCWs and their operands.
- Rewrite CCWs as necessary, so that their data areas are contiguous and in page-fixed storage.
- Drastically curtail opportunities for chained scheduling and other performance-enhancing techniques for modifying channel programs after they have been issued by users.

The Channel DAT Solution

M series channels accept a special CCW command code called transfer virtual and lock (TVL). When issued by the OS IV/F4 I/O supervisor, a TVL command causes the channel to interpret subsequent CCWs as having virtual addresses. The Supervisor typically issues a TVL command in the system nucleus, pointing to the user's channel program in his virtual storage address space. The channel util-

izes the segment table corresponding to this address space; OS IV/F4 names this segment table in the TVL command.

Channel DAT permits CCWs, operands, and data addresses to be in virtual storage. Furthermore, pages containing these CCWs, operands, and data need not be fixed into main-storage frames throughout entire job steps or interactive sessions. The OS IV/F4 I/O Supervisor short-term fixes channel program pages at the begining of each I/O operation. It also fixes operand areas referenced by the channel program: control operands and data areas. Hence, no page-fault interruptions occur during channel operation. After each I/O operation completes, the I/O supervisor unfixes these pages. This decreases supervisor overhead for I/O operations, and yet allows compatibility with other similar operating systems.

CHAPTER 2
JOB MANAGEMENT

2.1 OVERVIEW

OS IV/F4 Job Management performs the following services to user jobs:

- Job scheduling
  OS IV/F4 schedules and controls job flow including initiation, execution, and termination of all batch jobs.
- Master scheduling
  OS IV/F4 controls operator consoles and receives inputs from operators at any time.
- System management facilities (SMF)
  OS IV/F4 provides exit routines and other exits at several points during job initiation, step initiation, step execution, termination, etc. where each installation may gather statistics about the performance of the entire system (or of particular jobs).

In addition, OS IV/F4 provides a job entry subsystem (JES) and remote entry services (RES), which manage entry and return of jobs from local and remote unit record devices, respectively.

JES is described in Section 2.2 of this chapter, RES in Chapter 3.

2.1.1 Jobs and Job Steps

The basic batch-processing unit in OS IV/F4 is a **job**, a connected sequence of processing tasks using a collection of permanent and temporary data sets. Each job comprises one or more **job steps**, single executions using one set of I/O devices and associated data sets. The distinction between "tasks," "job steps," "jobs," and "sequences of related jobs" may seem relatively arbitrary to the user, but each of these terms is uniquely and formally defined in OS IV/F4, just as for most other modern operating systems. These entities and their relationships to one another are carefully defined in the present section.

2.1.2 Job Flow

Input
To bring jobs into OS IV/F4, the console operator must start one or more JES readers. (Users cannot directly start JES readers.) Jobs can be read from local card readers, magnetic-tape or DASD drives, or—via RES—from remote terminals. JES temporarily stores the **image** of each job—the collection of card images—onto the SYS1.SYSPOOL data set. Job control (JCL) statements are not interpreted and processed at this time. After JES has successfully read an entire job and transcribed it onto the **spool data set** (SYS1.SYSPOOL), JES enters a record for this job into the system job queue (SYS1.SYSJOBQE data set). At this time, the job is sorted onto a selection queue according to its CLASS and PRTY parameters, which take the values

A, B, C, . . . , O

and 0, 1, ... , 13, respectively.

Job initiation
The console operator normally starts one or more job initiators to process batch jobs from each class (as designated by CLASS parameters on users' JOB statements). To each initiator corresponds one or more classes, sequenced in priority order. As OSIV/F4 completes processing a job, the corresponding initiator picks the next job from the highest-priority non-empty queue of unprocessed jobs. The initiator interprets JCL statements for the selected job, allocates system resources to it, and yields control to this job. For temporary storage of interpreted and processed JCL, the initiator uses a **scheduler work area data set** (SWADS); one SWADS corresponds to each active initiator.

Job execution
Each active initiator processes exactly one user job in one virtual address space of 16 million bytes. Thus, the number of concurrently executing jobs is at most equal to the number of initiators. JES performs all transcriptions of system input (SYSIN) and system output (SYSOUT) data sets to unit-

record and other low-speed devices; this process is usually called **spooling** of SYSIN and SYSOUT.

Step termination
After the first step of a job completes, the OS IV/F4 step-terminator routine automatically gains control to perform post-processing of data sets and recording of SMF data. If the job contains additional steps, the initiator regains control and commences execution of the second, third, etc. steps. At the conclusion of each step, the step terminator regains control until no more steps remain in this job. The step terminator decides whether any job steps should be skipped, based on user-furnished conditions (COND parameters) for these steps.

Job Termination
After all steps have been processed (or skipped. as appropriate), OS IV/F4 terminates the job. All system outputs (SYSOUT data sets) are written onto the spool data set, to await transcription by JES/RES to their ultimate output devices, typically a line printer and/or a card punch. SYSOUT is distributed into classes according to user-furnished SYSOUT parameters for certain temporary data sets.

Disposition of job outputs
The JES writer processes SYSOUT data sets after the corresponding job completes, according to which SYSOUT classes they were directed and the user-furnished PRTY parameter on the JOB statement. The operator manages each JES writer according to the type of device it services, user-furnishes SYSOUT controls, forms controls, print trains, etc.

Monitoring job execution and collecting usage data

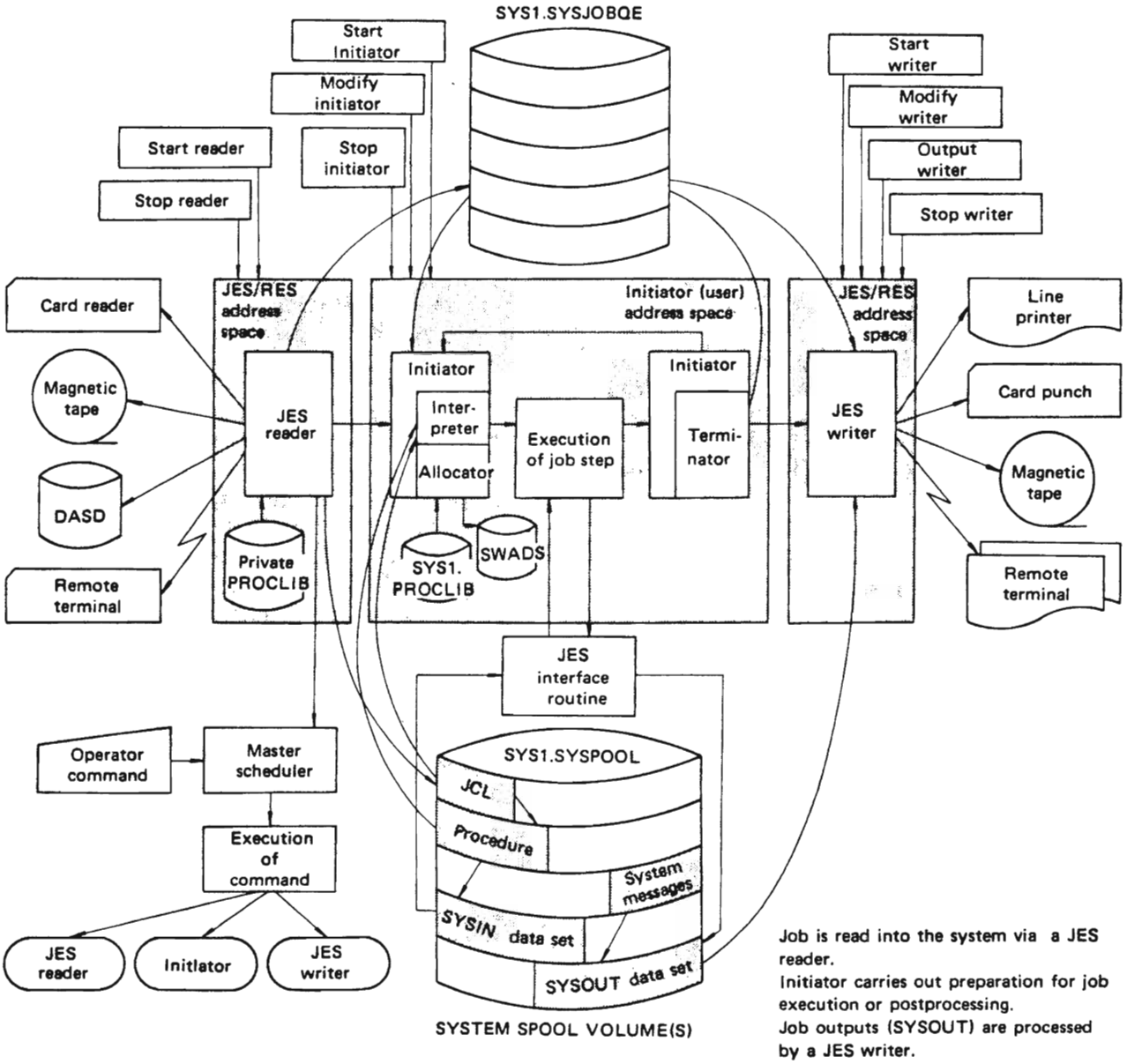


Fig. 2.1 Outline of Job execution

SMF provides to each installation a flexible set of entry points where locally-written routines to validate, summarize, and evaluate resource usage data – CPU time, channel programs, virtual-storage pages, etc. – can be inserted. The routines can monitor activity and efficiency of individual jobs as well as aggregate system performance. If a particular user job reaches a pre-selected CPU time limit, a special installation-supplied routine gains control via SMF; this routine can – for example – terminate the job immediately, request a decision by the console operator, or change the charging rate for CPU time for this job. With SMF, each OS IV/F4 installation can collect perjob data needed for accounting or charge-back purposes, volume-usage data, dataset usage data, and the like.

Operator commands
All commands entered from the operator consoles are accepted and processed by the OS IV/F4 Master Scheduler. Other operator commands can be prepared by users and entered before/with their JCL statements.

2.1.3 Components of Job Control

The three major components of the OS IV/F4 job control function are job scheduling, master scheduling, and the system management facilities (SMF), as shown in Fig. 2.2.

2.2 Job Entry Subsystem (JES)

JES controls all original inputs and final outputs for each batch job submitted through OS IV/F4. This section describes the functions performed by JES, plus a brief outline of how JES operates internally. For details on the input-reader and output-writer components of JES, the reader should study Sections 2.3 and 2.7, respectively. The present section describes the following aspects of JES:

- Overview.
- Structure of the spool volume.
- Optimization of spooling performance.
- Contents and space control of spool volumes.
- Interfaces between JES and user programs.
- JES parameters.

2.2.1 Overview

The job entry subsystem provided with OS IV/F4 is JES, which serves as the point of entry for all jobs and the function which produces all hardcopy job output. To accomplish these functions, JES controls local job-entry and output devices. A complementary OS IV/F4 facility, remote entry services (RES), furnishes comparable facilities for remote batch terminals. A special job entry source, the internal reader facility, allows OS IV/F4 users to submit system jobs: started tasks and time-sharing LOGONs. Tape and disk input are also supported

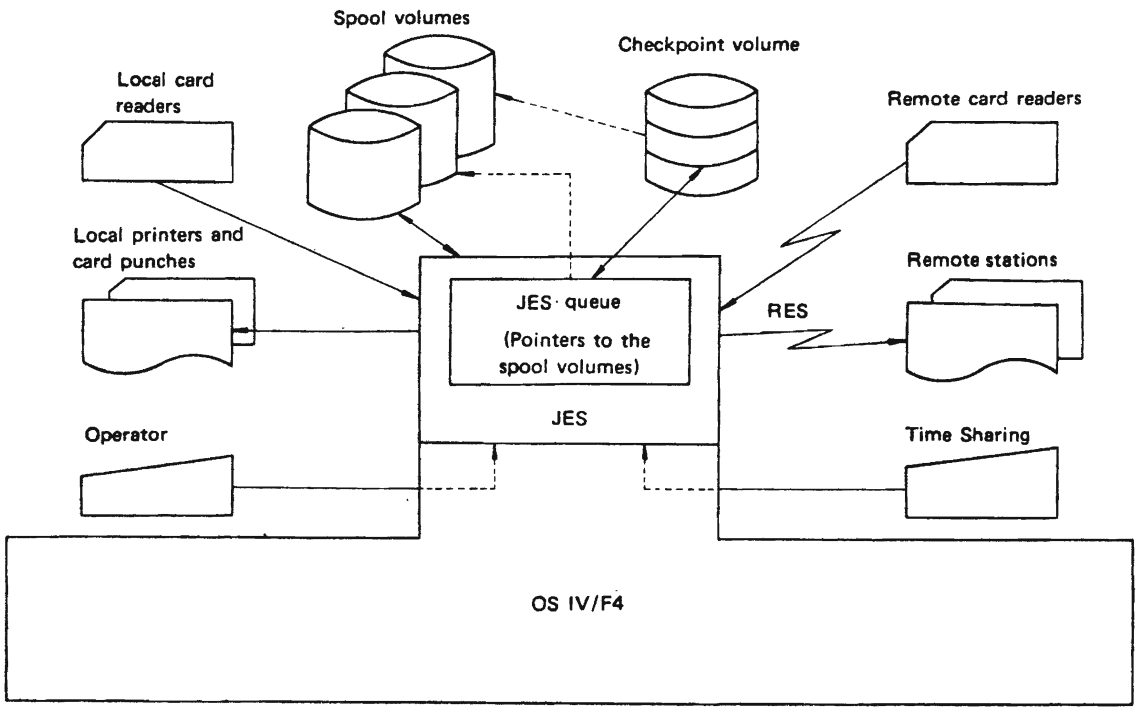


Fig. 2.2 JES I/O Relationships

through the internal reader facility. See Fig. 2.2 for input/output relationships to the job entry subsystem and OS IV/F4.

While each job is in OS IV/F4, the JES job queue residing in pagable storage maintains a record for the job. Job-related system records plus records related to job input and output are maintained on external spool volumes.

The system programmer during JES generation and initialization—plus the operator during JES processing—define and control the configuration of entry sources and output destinations. JES provides centralized control of job input, queuing, and output, such that all jobs are controlled in the same manner whether submitted from local or RJE (remote job entry) devices, or through the Internal Reader facility.

Fig. 2.3 outlines this section of the manual. As suggested in this figure, jobs that are batched for execution (**execution batch facility**) do not go through the same conversion and execution process as other jobs. Other functions described below are as follows:

- Configurations—configurations of local and RJE devices, generation of JES, and specification of the internal reader facility and spool volumes.
- Starting and stopping the job entry subsystem—starting the default (system-generated) subsystem, and initializing JES automatically via data sets containing initialization parameters.
- Controlling job submission and queueing—how to submit a job to JES, the internal reader facility, the RDR and RDRT procedures, the role of job classes and priorities in job queuing, priority aging, and placing of jobs in HOLD status.
- Controlling conversion and execution — JCL conversion, scanning the accout number field of the JOB card, defining a procedure library for the job, specifying converter parameters, command authority and recognition for JES and OS IV/F4, control of initiators, and job monitoring.
- The execution batch facility—establishment of the facility and writing an execution batch monitor.
- Controlling output and output devices—how output is queued and by what function, data set enqueuing, device selection, separator pages and separator cards, overflow, output routing, the external writer and the XWTR procedure.

Parameters necessary for controlling various JES functions are described in two manuals: **FACOM OS IV/F4 System Programmer's Guide** and **FACOM OS IV/F4 System Generation User's Guide**. These manuals contain detailed descriptions of the implementation of each parameter. When groups of parameters are described in this section, the reader is referred to the system generation manual or the initialization manual for implementation details.

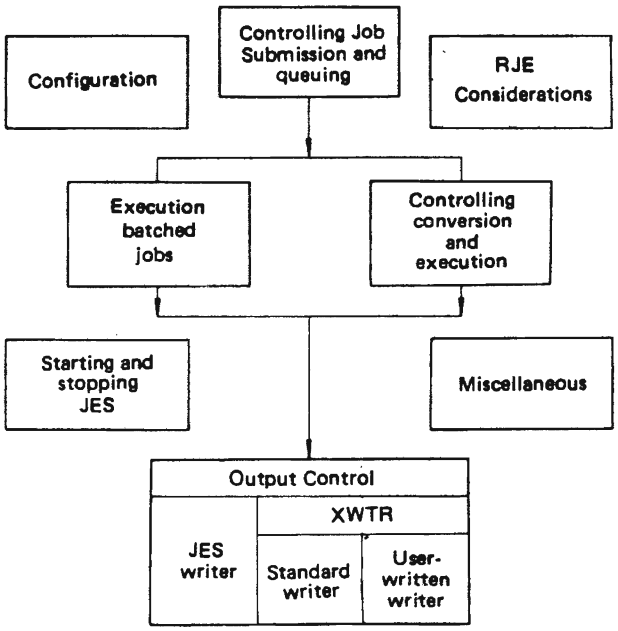


Fig. 2.3 Topics described under JES

During JES generation and initialization, the system programmer can specify the configuration of JES local devices, the JES internal reader facility, and the JES spool volumes.

JES is created by a process called JESGEN. The system generation process is designed so that the operator can generate JES while stage II of SYSGEN is in progress. Following JESGEN the operator must issue a START command START JESBLD in order to linkage edit JES into SYS1.LINKLIB and SYS1.LPAIB.

Local devices refer to card readers, printers, and card punches in an OS IV/F4 system for reading jobs and writing output.

During JES generation, the system programmer specifies the number of readers and writers to be controlled by JES via the &NUMRDRS and &NUMWTRS parameters. It is not possible to assign additional devices to JES without regenerating the system.

The system programmer can also specify JES processing parameters for each device and indicate whether a device is to be considered active or inactive upon completion of JES initialization. An active (for example, a "hot reader") device is dynamically allocated during JES initialization, and processing on that device begins as soon as work is available. An inactive device must be activated by the operator via a JES START command.

During JES initialization, if the system programmer does not identify as many devices as were specified during JES generation, JES selects devices and dynamically allocates them. Devices are selected according to lowest device address for each type of device (reader, printer, punch), until the number

specified during JES generation is obtained or no devices of that type remain. For a device to be selected, it must be physically attached to the system. For devices not identified to JES during JES initialization, default paremeters established during JES generation and initialization are used.

During JES processing, devices can be activated via the JES START command and deactivated via the JES STOP command resulting in their dynamic allocation or deallocation.

Role of JES

JES reads two kinds of source statements: JCL statements (including certain operator commands which may be optionally submitted by users) and system input (SYSIN) statements. Collectively, a sequence of jobs—each comprising one or more JCL statements plus associated SYSIN data sets—is called a **job stream** or **input stream.**

JES processes most output data sets from a job except those which are retained as permanent data sets on magnetic tape or DASD devices. JES also processes essentially all diagnostic messages from language processors, utility programs, and application programs. These outputs are written onto line printers, card punches, and other output devices intermixed in a precise and pre-defined sequence, so that they can be easily retrieved and interpreted by users.

The basic objective of JES is to handle system inputs and outputs rapidly and efficiently. The technique of spooling SYSIN and SYSOUT from disk to lower-speed devices has proven most efficient among all alternative techniques utilized and evaluated for prior operating systems. Spooling is described in the following paragraphs.

System input

JES reads several input streams—typically 3-10—simultaneously and stores the JCL and SYSIN data sets in a unique data set (named "SYS1.SYSPOOL") which is permanently allocated to one or more DASDs at each installation. Relatively small installations can use part of one disk drive for SYS1.SYSPOOL; intermediate size installations may require an entire disk drive, and large installations may need two or more drives for spooling functions.

Storage of input streams on DASD is only temporary; later, jobs in these streams are presented to OS IV/F4 initiators one at a time, in priority sequence, where they are processed just as if they were read directly from local card readers, etc.

System output

Likewise, JES transcribes output data sets from SYS1.SYSPOOL to one or more line printers, card punches, or other output devices, as indicated by user JCL. Here too, storage on DASD is only temporary for these data sets.

Optimal speed and efficiency

Although JES requires additional main storage, DASD space, and CPU overhead, its performance is generally superior to other job-entry designs; hence, JES is a mandatory service for each batch job entered or returned to the user by OS IV/F4.

Hardware/software components of JES

JES is a collection of OS IV/F4 system routines—for controlling associated DASD data sets and low-speed devices, using a substantial address space of its own — which are divided into two major services:

- JEPS — job entry peripherals service.
- JECS — job entry central service.

JEPS operates all low-speed peripherals, whereas JECS controls JES overall and allocates and manages space within the SYS1.SYSPOOL data set. The latter functions include buffer management and all JES interfaces to application programs; these interfaces are identical among all compilers, assemblers, utility programs, and user programs.

Contribution to system performance

The following six aspects of system performance are particularly enhanced by the OS IV/F4 JES:

Unit-Record Device Speeds

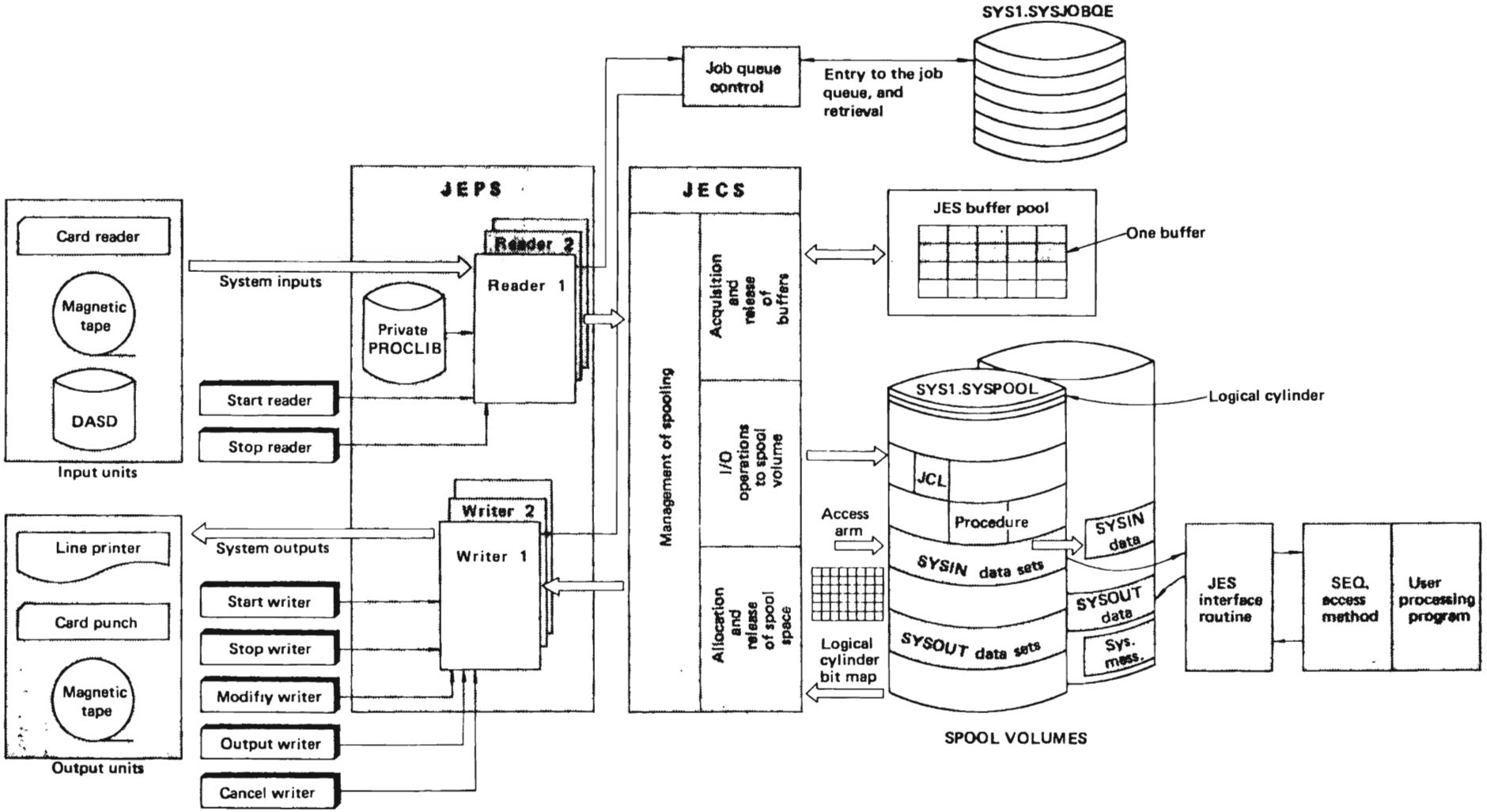
JES creates and uses specialized channel programs, which readANDwrite multiple records with a single EXCP macro instruction. This technique permits these devices to operate close to their top speeds, and it also reduces their per-record overhead on the main CPU. These channel programs are efficient in their usage of multiplexer subchannels and also in reducing virtual-address manipulations by hardware/software to a minimum.

Unit-record utilization

Since each card reader, card punch, line printer, etc. can be used for multiple unrelated jobs, OS IV/F4 users need not be assigned specific devices. Essentially any OS IV/F4 job can be submitted to the system through any appropriate device, with comparable flexibility for receiving SYSOUT data sets. Even if consecutive jobs are submitted on behalf of different users in a continuous stream, JES operates associated card readers, printers, etc. at full speed, creating and maintaining internal job delimiters.

Simplified job scheduling

All JCL and source data sets for a job can be submitted at one time, without concern for when and how long each job step will run. Hence, card readers, card punches, and line printers are committed to particular users only for as long as transcriptions to them require; they need not be committed during the execution of corresponding job steps, which can vary emormously.



COMMENTS: JES performs spooling for system input/output.
Readers and writers are reentrant modules to minimize their total need for virtual storage.
Prior to writing data to spool volume, equalization of load and optimization of access are taken into consideration.

KEY:
JEPS: Job Entry Peripheral Service
JECS: Job Entry Central Service
Command name — Shows operator command

Fig. 2.4 Configuration of JES

Flexible operation of JES readers and writers
Since reading source jobs and processing SYSOUT data sets are chronologically decoupled from job processing, a minimum number of readers can be started by the console operator even if the number of initiators is relatively large. Likewise, the number of output writers can be raised or lowered according to the available numbers of card punches, line printers, etc., without considering either the number of input readers or the number of active or inactive job initiators.

Minimal number of main-storage page frames
JES operates almost entirely out of virtualstorage pages which need not be fixed into realstorage page frames. Since readers and writers are re-entrant, only one copy of their program logic need be furnished in JES, plus associated I/O buffers and work areas. Furthermore, these reentrant pages need not be paged out; they are automatically discarded by the OS IV/F4 Paging Supervisor during periods of light activity, since they can be retrieved intact from their DASD library.

Installation-defined configuration
Each installation can define one or more JES configurations appropriate to its collection of unit-record equipment, remote terminals, usage intensity throughout the typical work week, size and speed of DASD devices for spooling, etc. During each reloading of OS IV/F4 (IPL), the console operator can either accept system-generation default values for JES or override them selectively.

2.2.2 Structure of the Spool Volume

Four elements of the spool volume are defined in the following section and diagrammed in Fig. 2.5 :
- Spool volume
- Spool data set
- logical cylinder
- initialization of a spool volume

JES uses the SYS1.SYSPOOL data set on each spool volume to store all job input, job output, JES control blocks, and system data such as the job journal. Spool volumes are identified to JES by their volume serial numbers. A six-character name identifying the primary spool volume is specified in the &SPOOL parameter during JES generation. A &SPOOL parameter can be used during JES initialization to override the JES generation parameter. The primary spool volume must exist during JES initialization.

Each volume with a volume serial number named in the SPOLVOL parameter is considered a spool volume by JES and is searched for a SYS1.SYSPOOL data set. The maximum number of spool volumes is also specified during JES generation by the SPOLVOL parameter.

The system programmer also specifies how tracks of the volumes are allocated and subdivided into physical records by the ALOCUNIT and BUFSIZE parameters. These parameters can be specified only during JES generation.

JES also requires one system checkpoint data set on a direct access volume to store a copy of the JES queue and other information needed for warm start. This data set must be on the primary spool volume. See **FACOM OS IV/F4 System Generation User's Guide** for a description of how to allocate this data set and SYS1.SYSPOOL data sets.

Spool volume
Each volume containing part/all of the SYS1.SYSPOOL data set is called a **spool volume.** Up to 10 DASD devices can be online with spool volumes, whose mount status must be permanently resident, i.e., never removed during processing. It is desirable to keep other high-activity data sets off spool volumes to avoid excessive contention for associated channels, control units, and access mechanisms.

Spool data set
All SYSIN and SYSOUT data sets for all jobs are written into and out of the single SYS1.SYSPOOL data set, which may possibly extend over two or more volumes. To the user's application program, however, it appears that each SYSIN or SYSOUT data set is directed to an independent DASD.

Logical cylinder
OS IV/F4 allocates space within the SYS1.SYSPOOL data set by logical cylinders rather than by blocks, tracks, physical cylinders, etc. Each **logical cylinder** is the same size and comprises one or more complete tracks, depending on the track length for the corresponding DADS and user-selected system generation parameters. The default capacity of an OS IV/F4 logical cylinder is approximately 40K bytes, equal to three tracks of a F478B or F479B Disk Drive or three tracks of a F6625A Drum.

The number of tracks per logical cylinder can be changed during system generation or after any "cold start" reloading of OS IV/F4 ("cold-start IPL").

If most jobs at a particular installation submit small SYSIN decks (500 card images = 40K bytes, for example), most space of SYSIN logical cylinders may be wasted. However, logical cylinders are also used to temporarily store SYSOUT data sets—approximately 500 print lines can be contained in each 40K logical cylinder. Hence, wastage of logical cylinders on input must be traded off at each installation against the CPU overhead for allocating and managing a larger number of smaller logical cylinders, e.g., with only one or two tracks apiece.

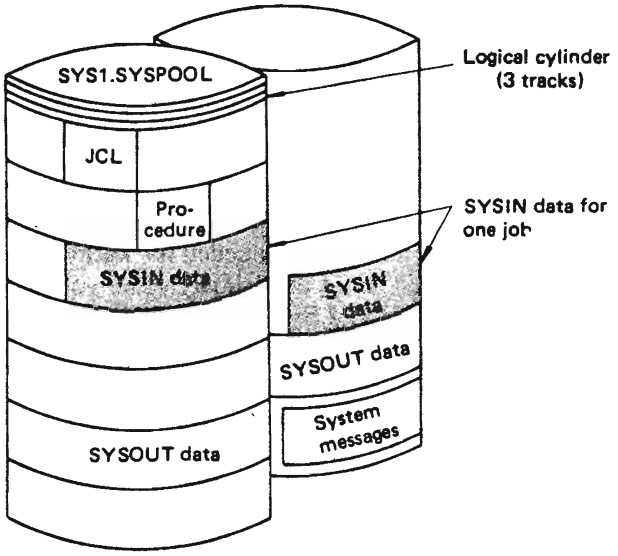


Fig. 2.5 Structure of spool volume

Spool volume initialization
Each logical cylinder on a spool volume is formatted into one or more fixed-length blocks; the block size is constant across all spool volumes at a particular installation, irrespective of device type. The default block size is selected during system generation, subject to overriding by the console operator during a cold-start IPL. The OS IV/F4 default value is 880 bytes.

During a cold start, all spool volumes are reformatted to the indicated block size, and the logical cylinder maps are zeroed. During a warm start, OS IV/F4 determines the block size from the previously-formatted spool volumes, as described in Section 2.11.3.

2.2.3 Spooling Performance Optimization

As JES reads and writes SYSIN and SYSOUT data, JCL statements, etc. to spool volumes, it employs several techniques for balancing this I/O load across channels and spool volumes.

Spool space allocation
To allocate fresh logical cylinders — either for newly-read SYSIN data or for newly-created SYSOUT data — JES utilizes a special algorithm to allocate an empty cylinder near the current position of an available spool volume access mechanism. One

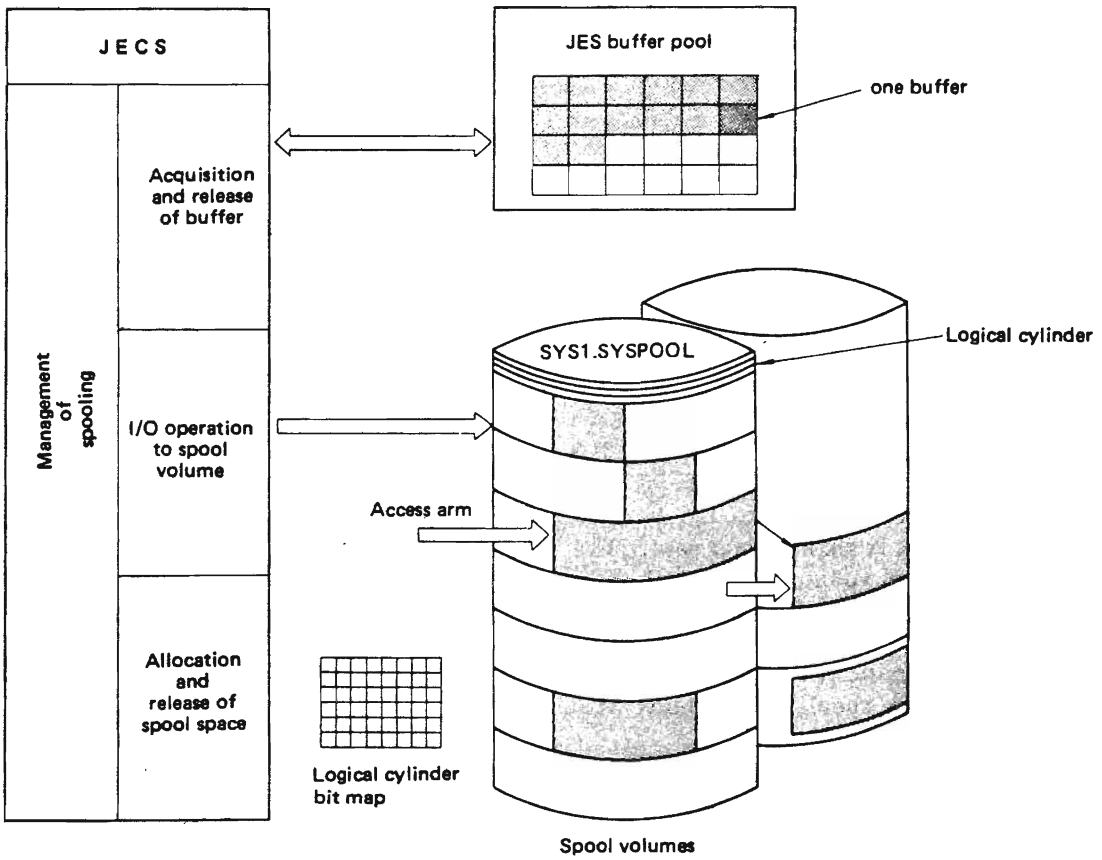


Fig. 2.6 Optimization of spooling

bit of the logical cylinder bit map corresponds to each logical cylinder on each spool volume, as shown in Fig. 2.6. This bit is turned on whenever the corresponding logical cylinder is allocated for storing SYSIN/SYSOUT; it is turned off when this logical cylinder is released to the unallocated pool, after its contents have appropriately processed.

Thus, JES uses essentially no references to spool volume VTOCs or other tables which are relatively cumbersome to search.

I/O Load Balancing
When a fresh logical cylinder must be allocated, JES selects a spool volume which is relatively lightly loaded at this instant. (If there is only one spool volume at this installation, the choice is trivial.) This choice is based on the average rate of accesses in the past few minutes to each spool volume.

Having selected a spool volume, the empty logical cylinder closest to the current access-mechanism position is allocated for the new SYSIN/SYSOUT data. In this way, a single SYSIN/SYSOUT data set may be distributed over several different volumes, in several discontiguous physical cylinders on each volume.

JES buffer pool
All main-storage buffers in the JES address space are centrally controlled, whether they are used for input card images (input card reader or communications line), disk spool buffers, operator messages, or output print/punch images. These buffers comprise the **JES buffer pool**; a sufficient number must be allocated at IPL time (defaulting to a system-generation parameter) so that no device need be idled due to exhaustion of the pool. Only buffers actively in use are page-fixed in real storage; hence, it costs the installation no additional real storage to allocate a generously large pool.

When JES needs a fresh buffer, it attempts to allocate one recently returned to the pool, since this buffer is unlikely to have been paged out by OS IV/F4.

2.2.4 Control and Space Management of Spool Volumes

Contents of spool volumes
JES has total control over the contents of spool data sets; no other system or user routine accesses these data sets directly. The following data is spooled.

- **Data from JES input readers**
  this includes all JCL and SYSIN data sets.
- **Data from Executing Programs**
  this includes all SYSOUT data sets.
- **Data from OS IV/F4 Routines**
  this includes all system messages created during JCL interpretation, program execution, step and job terminations, plus the system log.

Spool space conservation
JES truncates trailing blanks from each JCL, SYSIN, or SYSOUT record (or system message) prior to writing this record into the spool data set. This effectively converts all fixed-length records to variable-length, although JES reconstructs the original format prior to presenting these records to application programs or JES output writers.

Monitoring spool capacity
During routine operation of OS IV/F4, the storage capacity of spool data sets may become nearly exhausted. Hence, JES contains an installation-defined **threshold percentage** (default of 80%); when exceeded, the console operator is warned that spool capacity is nearly reached. Any active JES input readers are automatically stopped, so as to avoid accumulating additional SYSIN data.

The console operator should attempt to reduce the amount of spooled SYSIN and SYSOUT data by starting as many JES writers as possible and halting several initiators, so that new loads of SYSOUT records are not created on the spool volumes.

When the spool-volume utilization has dropped at least 10% below the threshold value, OS IV/F4 notifies the operator that sufficient spool capacity now exists to resume normal operation of input readers and initiators.

If spool volume utilization continues to grow, OS IV/F4 notifies the operator each time it rises another 5% above the threshold value. If spool capacity is entirely exhausted, executing jobs must be aborted, since there is no room for their SYSOUT data; this is a most undesirable situation and should be avoided at any cost.

SYSOUT limitation
Most installations impose limits on the amount of SYSOUT data written by each job. For example, if an undebugged program enters an endless loop writing SYSOUT records, it can quickly exhaust spool capacity. A default limitation SYSOUT for each job can be defined by each installation. A user can impose a SYSOUT limitation on a particular data set by furnishing an OUTLIM parameter on the corresponding DD statement.

2.2.5 Interfaces Between JES and User Programs

As noted earlier, user programs cannot read/write spool data sets directly; to read SYSIN data or write SYSOUT records, they issue requests to JES, which performs the actual input/output operations. Hence, user address spaces must communicate with the JES address space. SYSIN and SYSOUT data are handled as independent sequential data sets, subject to certain limitations described in Sections 2.3 and 2.7. As

a job step terminates – normally or abnormally – any associated SYSIN data sets are automatically deleted by JES from the spool volumes; likewise, after a SYSOUT data set has been completely printed, punched, etc., JES returns all logical cylinders containing this data set to the pool of unallocated logical cylinders.

2.2.6 JES Parameters

Most parameters have default values, which each installation sets while generating JES. The system parameter library SYS1.PARMLIB contains a JESPARA member, which holds these default values. Whenever the system is reloaded (IPL), the console operator has the opportunity to change these parameters selectively. Alternatively, permanent changes can be made to the JESPARA member with the JSEUPDTE utility program.

At system generation or thereafter, an installation can set the following JES parameters:

- number of buffers in the JES pool.
- maximum number of input readers.
- maximum number of output writers.
- maximum number of unit records read/written by each JES channel program, i.e., string of chained channel command words.
- volume serial number(s) of spool volume(s).
- spool capacity threshold.
- default limitation on SYSOUT records per job.
- size of a logical cylinder.

2.3 SYSTEM INPUT

Jobs are submitted through the job entry subsystem and queued in priority order. The system programmer can use various parameters and facilities to control input streams, to control the specification of job classes and priorities for jobs, to hold or release jobs, to set the default class and/or priority for a job, and to change these specifications by altering entries in the JES job control table, using the JOB statement SMF exit routine.

Jobs are submitted to JES in three ways:

- through card readers allocated to JES,
- through RJE devices allocated to JES, via remote entry services,
- through a JES internal reader facility.

The following section describes major aspects of JES readers:

- flow of control
- starting and stopping a reader
- reading methodology
- transcription to the input queue
- command statements
- reader procedures.

2.3.1 Flow of Control

JES reads JCL statements and SYSIN data sets for each job. Both types of inputs are stored on spool volumes, as described in Section 2.2. After an entire job has been successfully read, JES writes an entry into the system job queue to indicate the job's class, priority, size, and spool-volumes location. Optionally, the user may have designated the job to be held on queue until explicitly released by the console operator or to be merely checked for correct JCL syntax, rather than actually executed.

Optionally, the user can enter certain operator-command statements with his jobs either prior to his JOB statement or imbedded within his JCL statements.

2.3.2 Starting and Stopping a Reader

To start a JES reader, the operator designates the name of the associated **reader procedure** in a START command. This procedure is a collection of control statements previously entered as a member in the **system procedure library** (SYS1.PROCLIB). The reader procedure may optionally specify the device address for a card reader, magnetic tape drive, or DASD; alternatively, the operator can enter this address with the START command. The operator can start as many readers as have been generated for this OS IV/F4 system. Certain readers can be automatically started by OS IV/F4 immediately after the system has been reloaded (IPL). In this case, corresponding START commands must be included with the system-startup member of SYS1.PARMLIB.

To stop a JES reader, the console operator enters an appropriate STOP command. In the case of a magnetic tape or DASD drive, JES automatically stops the reader when it encounters an end of file (EOF) record. In the case of a card reader, the JES reader enters Wait State when the input hopper is empty rather than automatically stopping. If additional jobs are subsequently placed in the hopper and the device readied, JES resumes input reading; this facility is called the **hot reader** function and is commonly used by OS IV/F4 installations.

2.3.3 Reading Methodology

This section describes the JES access method, how JCL statements are read and partially processed, and how cataloged procedures are merged with JCL statements.

JES access method (JAM)
JAM is a special access method for reading several punched cards with one channel program, to achieve higher device efficiency and reduced system overhead. JAM may only be used for EBCDIC-punched

cards, not column binary cards.

If an input stream is read from magnetic tape or DASD, JES uses QSAM rather than JAM.

Reading and processing of JCL statements

JCL statements are not fully processed until the corresponding job is initiated by OS IV/F4. JES cannot fully check JCL while reading source statements, since this would cause substantial degradation of reading speed. However, certain parameters of the JOB statement are needed for scheduling each job; JES scans, validates, and enters these into the control block for this job. For omitted parameters, JES furnishes default values specified by the installation in the associated reader procedure.

Merging cataloged procedures

Most batch jobs use one or more cataloged procedures to simplify their JCL statements. If the operator has allocated SYS1.PROCLIB to the reader, JES does not copy procedures into the JCL stream at reader time, deferring this activity until the job has been initiated. If the operator has allocated a locally-developed procedure library to this reader, procedures are copied into the JCL stream at reader time.

2.3.4 Transcription to the Input Queue

All jobs are stored on spool volumes by JES while awaiting execution. A control block for each job is stored in the SYS1.SYSJOBQE data set.

Jobs are classified according to their class (CLASS parameter of JOB statement), selection priority (PRTY parameter), and various other attributes.

Job classes are designated by the letters A - O, fifteen altogether. Each installation determines the meaning of its own job classes according to such attributes as:

- heavy CPU usage,
- light CPU usage but heavy I/O usage,
- trival length,
- urgency, etc.

If the CLASS parameter is omitted, the corresponding reader procedure assigns a default job class to this job.

The JES queue

A job received by JES is enqueued in priority order on the JES queue residing in the JES address space. A job is considered received by JES when it has been totally read in and its control block placed on the SYSJOBQE data set.

The queue entry for each job contains its name, priority, a flag to indicate whether the job is held, pointers to JES control blocks on the spool volumes, and the **JES process** (JCL conversion, execution, output processing, purge) for which the job is next eligible. Jobs are selected in priority order for each JES process. Logically, the overall job queue comprises subqueues for each JES process, plus 16 subqueues (one for each class) for executing jobs.

A job which is held is not removed from the queue; instead it is made ineligible to be selected for any JES processing. A job can be held at any time. If an executing job is held by the console operator, it is not eligible for output processing until released. A job may be held by name, by class, or as a consequence of all jobs being held.

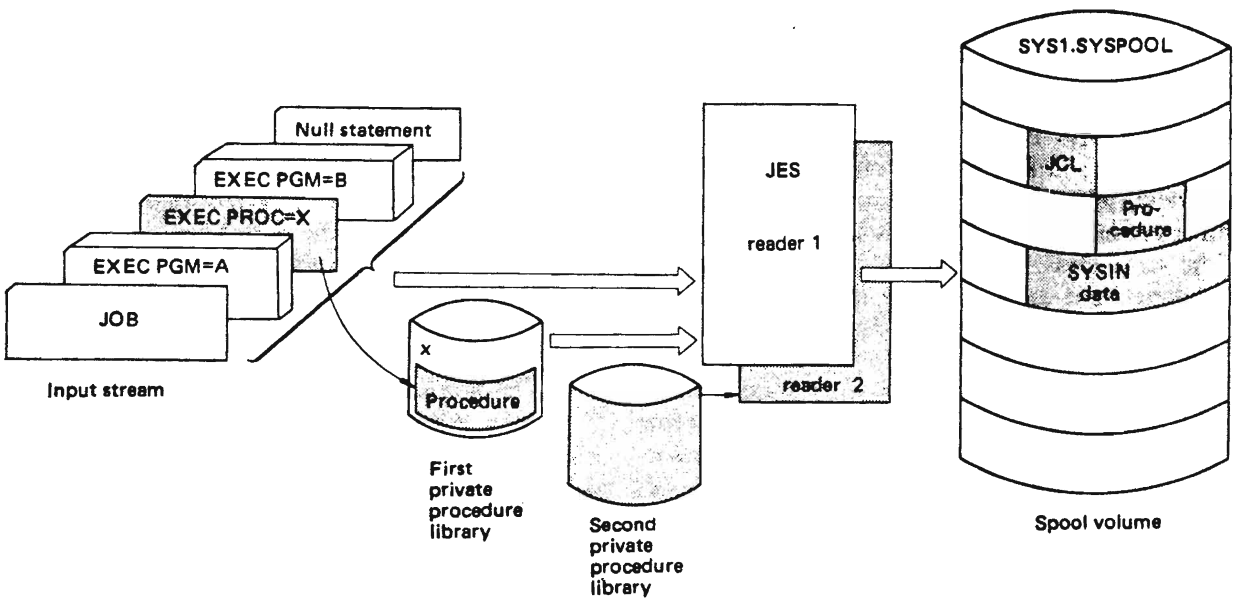


Fig. 2.7 Reader and procedure library

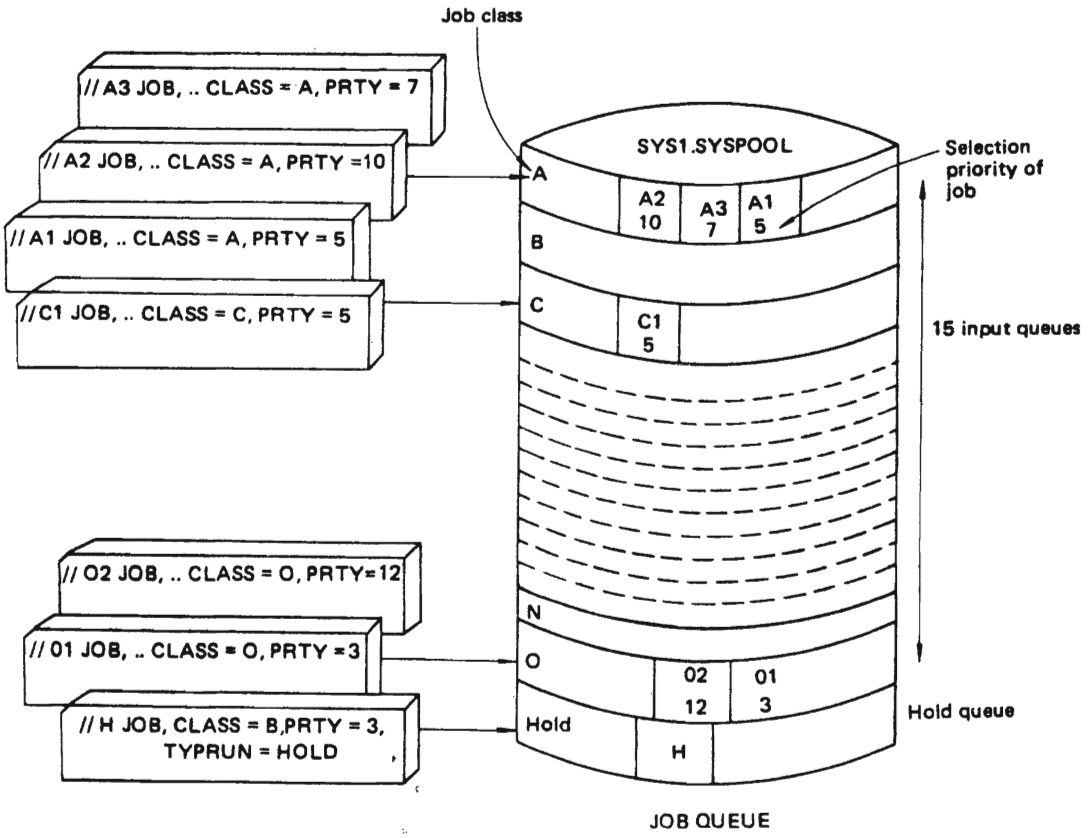


Fig. 2.8 Enqueuing jobs

Job class

There are classes of jobs possible under JES. One class is used by the system: CRJE for time-sharing LOGONS. The other 15 classes, (A–O) are for normal jobs and can be used to help control the job mix.

The job class is specified on the JOB statement (CLASS parameter). If not specified, JES assigns it a default class corresponding to the device through which the job is entered. All jobs entered through the internal reader facility are considered to be entered through a device described by the INTRDR parameter.

There are no absolute rules for assigning job classes, and some experimentation is necessary. Generally, jobs of similar characteristics and with similar processing requirements should be assigned to the same class. For example, if several jobs are time-dependent and must execute in nonpagable dynamic storage, it may not be desirable to tie up all of nonpagable dynamic storage by running these jobs concurrently. These jobs may all be assigned to class B (or C or D – class names have no inherent meaning); then, if only one initiator is started that can handle class B jobs, there will never be more than one of these jobs executing at once.

Suppose the following assignments are made:

Class B = jobs that are time-dependent.
Class C = jobs with high CPU requirements.
Class D = jobs with high I/O requirements.

The system programmer can specify initiator parameters such as:

I1 CLASS=BCD
I2 CLASS=CDB
I3 CLASS=DCB

If the three initiators are processing jobs with the same priority and all necessary resources (for example, I/O devices and data sets) are available, then three jobs – one from each of the three different classes – run concurrently. If a job within one of the classes has higher priority than others in its class, it will be initiated first.

During JES initialization, the system programmer can assign job classes according to running time, memory requirements, file resources, and urgency of jobs in each class. Characteristics of each class can include some/all of the following:

- JCL conversion parameters.
- Whether a JES job log is to be produced for this class. The JES job log is a list of all messages and replies issued by – or on behalf of – a job.
- Whether a system journal is to be saved for this job. If it is not saved, the overhead is avoided but the job may not be automatically restarted in case of job failure or system restart.
- Whether this class is reserved for the execution batch scheduling facility. (See Section 2.4.4)
- Whether output is suppressed for jobs in this class (e.g., started tasks).
- Define the procedure library (PROCnn).
- SMF options.

- Whether the job is held.
- Whether JCL statements are to be converted but not executed.

The system programmer should assign separate job classes to jobs having distinct execution characteristics such as:

- ratio of CPU to I/O processing
- use of special devices
- number of devices used
- use of real storage

Within classes, jobs are selected for initiation according to their **selection priorities** designated by the PRTY parameters of their JOB statements, ranging from 0−13. If several jobs have the same CLASS and PRTY parameters, OS IV/F4 selects them according to time of entry. If the PRTY parameter is omitted from a JOB statement, JES assigns a default value from the corresponding reader procedure.

If the user codes TYPRUN=HOLD on his JOB statement, the job is automatically held by OS IV/F4 until released explicitly by the console operator. When he issues a RELEASE command for this job, it reenters the normal queue for its job class.

If the user codes TYPRUN=SCAN, OS IV/F4 marely scans the syntax of his JCL statements and does not execute any job steps. Scanned jobs flow immediately from JES through the OS IV/F4 JCL scanner/interpreter and back to JES for SYSOUT processing; hence, they provide a convenient and inexpensive way to validate JCL for long production runs.

2.3.5 Command Statements

Most commands are entered by console operators with typewriter or CRT devices. However, batch jobs may contain the following operator commands:

CANCEL	MOUNT	START
DISPLAY	RELEASE	STOP
HOLD	REPLY	UNLOAD
LOG	RESET	VARY
MODIFY	SET	WRITELOG

Interpretation of commands depends on their position within an input stream, as shown in Fig. 2.9. If commands appear prior to a JOB statement but following the delimiter of a previous job, they are interpreted and executed at JES reader time, i.e., as soon as read. If commands follow a JOB statement, their interpretation and execution are deferred until this job is initiated.

The JES reader procedure determines whether each in-stream command should be executed, displayed and then executed, displayed and then executed only after confirmation by the console operator, or ignored altogether.

2.3.6 Reader Procedures

JES starts an input reader in response to a START command specifying a reader procedure. Standard reader procedures are developed by each installation and entered into the system procedure library (SYS1.PROCLIB). For a card reader, the standard procedure name is RDR; for magnetic tape or DASD, the standard name is RDRT.

The internal reader facility

An internal reader jobstream is identified to JES by the fact that an output data set specifying a special user writer (INTRDR) has been allocated dynamically, or via SYSOUT=(x,INTRDR) coded on a DD card. JES recognizes such data sets and places them in the input stream, thus allowing jobs and system tasks to enter jobs into the input stream.

A job entered through an internal reader begins with a //JOB statement and ends with the next // JOB statement, or /*EOF statement, or by closing the internal reader data set. Abnormal closing or closing after a WRITE error causees deletion of the last job. A/*DEL statement may be used to explicity delete the last job.

The class to which the internal reader data set is allocated, e.g., class X if SYSOUT=(X,INTRDR),becomes the MSGCLASS for the submitted job unless the JOB statement contains a MSGCLASS parameter. If the internal reader data set is dynamically allocated without a specific class, the MSGCLASS of the submitting job or TSS user becomes the default. Two exceptions to this are time-sharing LOGONs and started tasks. If specified for the internal reader allocation, the DEST parameter becomes the default SYSOUT destination for all jobs submitted via that internal reader.

JES can accept multiple jobs simultaneously via the internal reader facility. OS IV/F4 uses it to pass started tasks, TSS LOGON, and TSS background jobs to JES. Also, jobstreams can be read from tape and disk (any QSAM-supported device) and submitted through the internal reader via the RDR procedure, and any job executing in OS IV/F4 can use the internal reader facility to pass a job-stream to JES.

Although the internal reader facility appears to JES logically as multiple input devices (maximum set by the &NUMINRS parameter during JES generation), the facility is controlled as one entity. The number (&NUMINRS) of internal readers is the number of jobs that can be received simultaneously through this facility.

Characteristics of the facility are specified during JES initialization as subparameters of the INTRDR parameter.

The RDR procedure

The OS IV/F4 procedure for using the internal

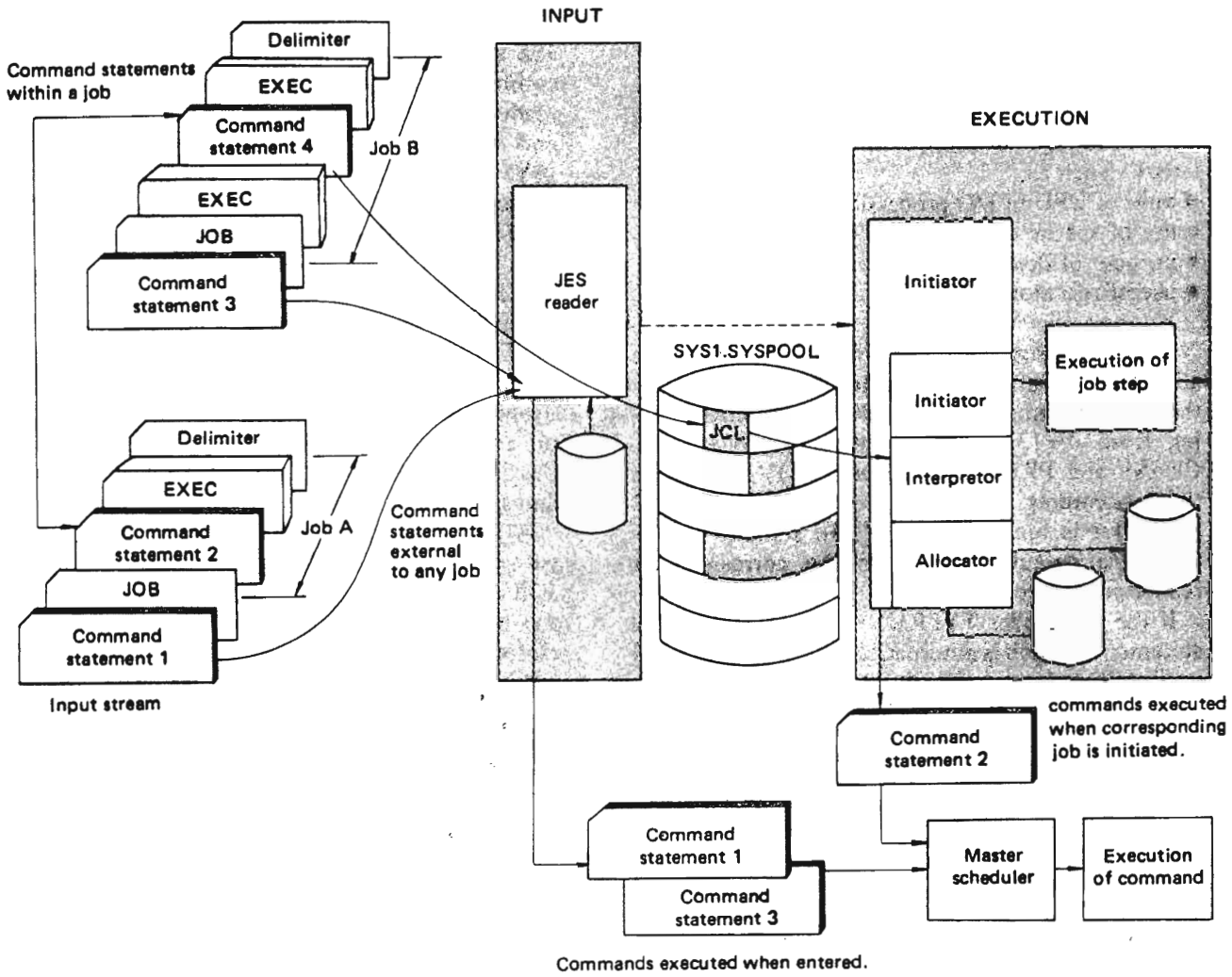


Fig. 2.9 Execution of command statements

reader facility to read jobs from tape or disk is named KDJRDRI. The starter system provides the RDR procedure in SYS1.PROCLIB to allow the operator to start the reader (see Fig. 2.10). Basically the same procedure can be used to read a jobstream from any QSAM-supported device. The operator uses the RDR procedure as follows:

```
//KDJPROC   EXEC  PGM=KDJVMA,
                  PARM='00600
//                300005010E00011A'
//KDJRDER   DD    UNIT=F610-2,
//                LABEL=(,NL),
//                VOLUME=SER=SYSIN,
//                DISP=OLD,
//                DCB=(RELFM=F,
//                LRECL=80)
```

Fig. 2.10 The RDR procedure

- To read a jobstream from the second file of a tape named JOBTAP on device 180:

```
START   RDR,180,JOBTAP,LABEL=2,
        DSN=JOBS
```

- To read a jobstream from a cataloged library of jobs:

```
START   RDR, F478B,
        DSN=PRODUCTN(PAYROLL)
```

- To read a jobstream starting with a specific job on a tape named JOBTAP, the operator must submit a job to JES:

```
//READJOBx  JOB     .....
//         EXEC  RDR
//RDER     DD    DSN=JOBS,
//               VOL=SER=JOBTAP,
//               UNIT=TAPE,
//               DISP=OLD
//SYSIN    DD    *
//         EDIT  START=JOBx
/$               UNIT=
```

The system programmer can define internal readers on EXEC statements in such a manner that

they are started conditionally. This allows the formation of a set of dependent jobs that can execute without operator intervention. For example:

- To submit Jobs B and C if the first steps of Job A complete successfully.

```
//JOBA      JOB  .....
//STEP1     EXEC ....
//          ...  ...
//          ...
//          ...
//          ...
//STEP5     EXEC RDR,
//               COND=(8,LE)
//JSERDER   DD   DSN=JOBS(JOBB),
//               DISP=SHR
//          DD   DSN=JOBS(JOBC),
//               DISP=SHR
```

- To submit Job B if Job a terminates normally, Job C if it terminates abnormally, and Job D in either case.

```
//JOBA      JOB  .....
//STEP1     EXEC .....
//          ...
//          ...
//          ...
//STEPN     EXEC RDR
//JSERDER   DD   DSN=JOBS(JOBB),
//               DISP=SHR
//STEPN1    EXEC RDR,COND=ONLY
//JSERDER   DD   DSN=JOBS(JOBC),
//               DISP=SHR
//STEPN2    EXEC RDR,COND=EVEN
//JSEFDER   DD   DSN=JOBS(JOBD),
//               DISP=SHR
```

Installation-written procedures and programs can further exploit the intemal reader facility to select particular jobs, to generate special job streams, and to allow operator submission of production jobstreams.

2.4 JOB INITIATION

This section describes how jobs are selected for execution and processed by OS IV/F4. Allocation of resources is described in Section 2.5, execution of jobs in Section 2.6.

2.4.1 Overview

How jobs awaiting execution are controlled by OS IV/F4 is shown in Fig. 2.11.

As described in Section 2.3, batch jobs are entered and enqueued for execution by class, priority, and entry time. The SYS1.SYSJOBQE data set contains, for each job, all attributes necessary to schedule it effectively.

Each OS IV/F4 job initiator can process from one to eight job classes. For example, Initiator N might process classes C, D, H, B, A, in selection order. If the following jobs were awaiting execution, with indicated classes, priorities, and entry times,

Name	Class	Priority	Entrytime
JOE	D	3	01:00
SAM	A	4	01:30
TOM	C	2	01:30
HARRY	D	1	02:00
BILL	D	12	02:00
DICK	D	3	02:30

then their order of selection by Initiator N would be as follows:

TOM, BILL, JOE, DICK, HARRY and SAM.

To prevent low-priority jobs from remaining enqueued indefinitely, OS IV/F4 provides a **priority aging** facility, whereby the selection priority of each job increases one level at fixed time intervals. After a substantial time on queue, even a low-priority job will gain a sufficiently high selection priority to bypass a higher-priority job in the same class.

The H,L, and T subparameters of the PRICOND JES-generation parameter specify, respectively, a limit above which there is no incrementing, a limit below which there is no incrementing, and an integer representing the number of times that the priority is incremented in a 24-hour period—subject to the upper limit. The default of zero for the T subparameter specifies no priority aging.

The T subparameter specifies whether the feature is used and, if so, how many times in a 24 hour period the priority is incremented. For example, PRICOND=(T=48) specifies a priority increment of one unit every 30 minutes. The H subparameter specifies the upper limit; a priority lower than the value of the L subparameter specifies that the job is not subject to priority aging.

Within each OS IV/F4 job initiator are the following components: **JCL interpreter,** resources allocator, job terminator, and others. The JCL interpreter converts JCL statements into control blocks after fully validating them against one another and against installation standards. These control blocks are written into the **scheduler work area data set** (SWADS) for this initiator.

The **resources allocator** allocates main storage, I/O devices, data sets, and other resources to the job and its component steps. If dedicated data sets are available for this initiator, the allocator will attempt to pair them with user-JCL requests for dedicated data sets, which serves to reduce the aggregate overhead for allocating/de-allocating temporary data sets.

The **step terminator** reclaims all resources allocated to each job step and issues appropriate system messages and return codes. The terminator decides

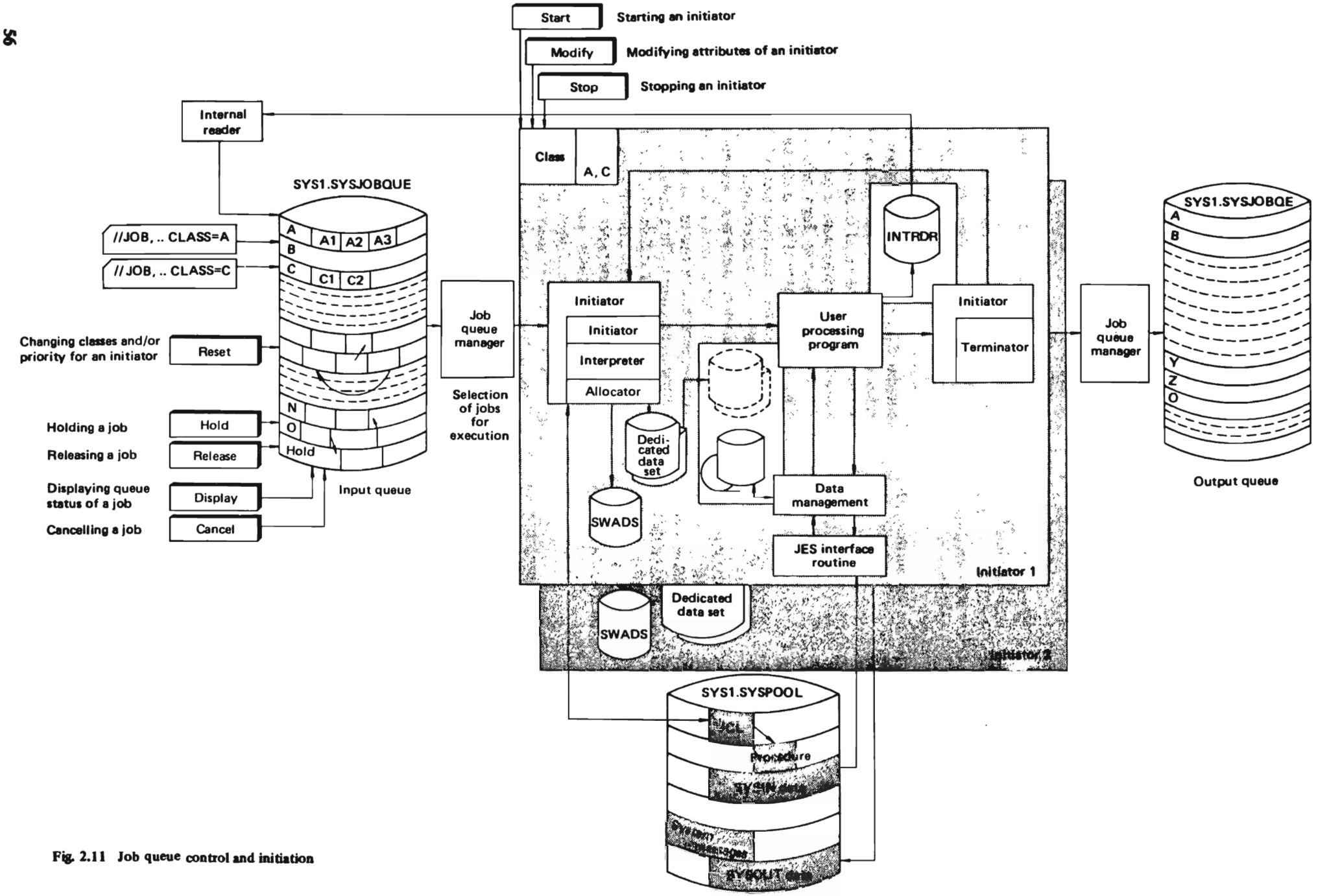


Fig. 2.11 Job queue control and initiation

whether subsequent job steps should be executed or skipped.

To improve efficiency of processing small similar jobs, OS IV/F4 furnishes an execution batching facility, which bypasses most functions in job and step initiation.

Users can optionally direct job outputs to an internal reader (INTRDR), which converts them to system inputs including any necessary JCL statements. Hence, one OS IV/F4 job can create one or more independent jobs, to be executed at a later time.

2.4.2 Job Queue Control

The SYS1.SYSJOBQE data set contains all data about each job awaiting execution needed to schedule it timely and efficiently. This data set contains three principal queues: the input queue, output queue, and held-jobs queue. The **input queue** contains control blocks describing jobs awaiting execution, the **output queue** describes jobs whose output is awaiting printing and punching, and the **held-jobs queue** contains jobs which have been explicitly delayed from selection by their submitters or by the console operator.

Jobs remaining enqueued for several hours rise in selection priority according to an installation-defined algorithm. Usually limits are set for each job class, such that the aging increments to its priority can not exceed a certain maximum value. Also, the interval between priority incrementings is defined by each installation.

The following table indicates which operator commands can be used to control the job queue:

Operator command	Functions
DISPLAY Q	Display number of jobs on the input, output, and held-jobs queues, by job class
DISPLAY N	Display names of these jobs
DISPLAY job-name	Display the class, current selection priority, and relative position on the job queue of the specified job
HOLD { Job-name / Job-class }	The specified job (or class of jobs) is held from execution

Operator Command	Functions
RELEASE { Job-name / Job-class }	The specified job (or class of jobs) is released for execution.
RESET	The class of a job is reset to its original value
CANCEL	The specified job is deleted and discarded.

2.4.3 Job Initiator Functions

In OS IV/F4, a job initiator prepares jobs for execution, yields control to individual job steps, and processes job outputs destined for local/remote printers, card punches, etc. The specific functions of an initiator are as follows:

- selection of each job from the queue.
- interpretation of its JCL statements.
- controlling its virtual storage area.
- allocating I/O devices.
- disposing of its data sets after each step, and after the entire job completes.
- issuing volume-mount messages.
- initiating job steps.
- termination processing for job steps and the entire job.

Using the selection algorithm described in Section 2.4.1, (priority aging) each initiator selects the highest priority job within the classes for which it is eligible. Selection occurs immediately after termination processing for the job previously controlled by this initiator (if any). If no jobs are enqueued for execution in classes for which this initiator is eligible, the latter enters Wait state until an appropriate job arrives.

If two or more initiators can process the same job class, each will make its selection decisions independent of the other. Of course, when one initiator has selected a particular job, the latter is immediately marked ineligible for selection by any other initiator. If a particular class is not designated for selection by any initiator on an OS IV/F4 system, any jobs entered in this class remain enqueued indefinitely. As soon as an initiator is started which is eligible for this class, corresponding jobs become available for processing.

Each initiator (and its stream of jobs) occupy one address space. Time-sharing jobs also occupy individual address spaces.

Just prior to executing the first step of a job, the initiator processes all user-furnished JCL statements and merges them with associated cataloged and in-stream procedures. Interpreter routines create corresponding control blocks in the scheduler work area data set (SWADS) for this initiator.

2.4.4 Execution Batch Scheduling

Execution batch scheduling is an extension of normal job scheduling to provide improved system performance by gathering pseudo-jobs, **execution batch jobs**, into a single input stream. Execution batch jobs are typically submitted to JES one at a time; they may have different input sources and different print and punch output routings. Execution batch scheduling collects related batch jobs into a single stream and passes them as a single SYSIN data set to a user-written execution batch processing program.

This reduces overhead associated with setting up and processing numerous individual jobs and/or job steps.

Processing programs using execution batch scheduling represent a wide variety of application areas such as:

- compile-and-go debugging compilers.
- file inquiry programs.
- hardware or software system emulators.

Typically, a program is suitable for execution batching if it handles jobs or transactions of relatively short duration. If not, the reduction in job management overhead between successive jobs may not be sufficiently large to justify use of this feature.

At JES initialization, each installation defines zero, one or more job classes as dedicated to execution batch scheduling. One or more classes may be assigned to each type of execution batch processing program. Subsequently, the user identifies which program he wishes by the appropriate class.

JES can support more than one execution batch processing program to process various kinds of batch jobs. Each execution batch processing program must be associated with at least one JES initiator.

To determine which jobs are eligible for execution batch processing, the JES reader scans all JOB statements. Instead of sending execution batch jobs to an initiator, JES invokes an appropriate procedure from PROCLIB to initiate the execution batch processing program. The job then becomes part of the SYSIN stream for the execution batch processing program.

For example, consider an order entry system that requires an inventory update and an invoice for each order. With standard processing, the normal procedure would be to batch all orders and submit them as a data stream at the end of the day to an order entry system.

However, this causes delay. Alternatively, the installation could periodically batch together orders received during a certain time period and run the job several times a day. By using the execution batch scheduling facility, orders can be processed as if the order processing program were scheduled for every order, but without the overhead of scheduling the order program for the runs.

To implement this approach, the installation would designate the order entry program as an execution batch processing program. Orders would be submitted as execution batch jobs by prefixing order data that would have been submitted in batch with a specialized JOB statement. For an order entry program designed to read batch jobs at the end of the day, the only logical changes would be (1) to enter the input stream via SYSIN (this may be a ccomplished by JCL in PROCLIB), (2) to recognize the null statement as an order separator or to establish other terminators, and (3) to ignore all other JCL statements. The program should print all information relating to one order before processing the next order, to distinguish one from another. JES would automatically schedule the order entry program sporadically and concatenate all orders into the input stream, regardless of where they originated.

Submitting input to an execution batch processing program

A representative input stream follows:

```
//JOB
JES control statements
input data
```

To submit data to an execution batch processing program, the user should follow certain rules:

- The first statement of each job must be a standard JOB statement with the specialized CLASS parameter. The class identifies which program is to receive the input. The installation associates certain classes with the execution batch facility via the system procedure library. The accounting field of the JOB statement is interpreted by JES just as it is for normal jobs.
- All JES control statements are effective with execution batch jobs, except /*OUTTPUT is ignored.
- No other JCL can be furnished. Remaining source statements comprise the SYSIN data set for the execution batch processing program, read just as if they had been placed in a DD-DATA data set and the execution batch program were invoked by standard JCL. If the execution batch program requires it, each transaction can be terminated by a statement with $$ in columns 1 and 2.

In the order entry system example mentioned earlier, the user might code the following:

```
//JOBxx   JOB    (INVO1,667),CLASS=X
/*ROUTE  PRINT RMT47
order 1
order 2
```

The /*ROUTE statement would cause the invoice to be printed at the indicated remote location.

Execution batch scheduling operations

Special actions take place when JES recognizes input for an execution batch program.

If the execution batch program is not already active, JES automatically submits an internal job which uses JCL from SYS1.PROCLIB to invoke the execution batch processing program, awaiting availability of an initiator. JES control cards are converted to JCL comment statements. The entire input stream, plus a JCL null statement added by JES, is allocated to the execution batch processing program as a SYSIN data set.

If the execution batch program is already active and awaiting another job, JES allocates the SYSIN data set as above and processing begins immediately,

without any additional use of OS IV/F4 job management.

The end of the SYSIN stream can be detected by the execution batch program when it reads the JCL null statement added by JES. After writing any remaining SYSOUT data for the completed job, the execution batch program attempts to read ahead in its input file for another transaction. JES detects this condition, temporarily forces the execution batch program in to wait state, and performs job termination actions for the execution batch job (flushes output buffers, releases input spool space, queues the job for printing, and so forth). The execution batch program remains active in the OS IV/F4 address space.

When an execution batch program is waiting, JES job selection is altered. Instead of scanning for any class eligible to execute in that address space, JES2 first tries to start an execution batch job which may be processed by the currently-loaded execution batch program. If successful, processing can begin immediately.

If no jobs of the same execution batch class are awaiting execution, other job classes for this address space are scanned in order. If a job is found, JES internally cancels the execution batch processing program; normal job scheduling then commences.

If no jobs of the other eligible classes are found, the address space and execution batch processing program remain idle, awaiting availability of a job in any eligible class. If a job enters the system whose class corresponds to the execution batch program still in the address space, processing begins immediately.

If an execution batch program ends (ABEND or normal return to OS IV/F4) JES detects this as a non-batch termination in the address space.

OS IV/F4 Job Management will again invoke the batch program when another job for its class is selected.

In summary, an execution batch processing program must have certain characteristics:

- It must read all user input from a single sequential data set.
- It must recognize a standard JOB statement (or its own control statement) to determine the beginning of each job.
- It must recognize a standard null statement ("//" followed by 78 blanks) or its own control statement to determine the end of a job.
- To ensure system integrity, it should not dynamically allocate SYSOUT data sets.

The execution batch processing program will receive an end-of-file condition when a card with $$ in columns 1 and 2 is read while processing a job. The program may continue to the next logical subfile by simply resetting appropriate bits in I/O control blocks and continuing reading, or by closing and reopening the data set to continue reading at the card following the $$ card.

Execution batch scheduling preparation

Job classes are reserved for execution batch jobs with the BATCH initialization parameter. "EXBTCH" must be used the first seven characters of each catalog-procedure name containing JCL necessary for an execution batch program.

Each batch class should be associated with one execution batch program. Each batch class should be made eligible to execute in an OS IV/F4 address space by issuing an appropriate START command.

For each combination of batch class and initiator, there must be a procedure in SYS1.PROCLIB named "nnnnncid", in which:

- "nnnnn" are the five characters assigned to &XBATCHN.
- "c" is the particular batch job class set in $$x.
- "id" is the 1- or 2-character initiator identification, corresponding to "nn" of the Inn parameter.

These procedures actually call the execution batch program for each class, and define all data sets other than the user input data set.

The procedures may be single-step, or they may have preliminary steps before the step invoking the execution batch program (stepname GO). The execution batch program invoked by this step must read its input from SY2 or the procedure must refer to DDNAME=SYSIN on a DD statement used for input by the processing program.

If a given batch class is eligible to be executed by more than one initiator (the Inn initialization parameter or $T operator command defines eligible classes), the requirement for a separate procedure name for each address space/class combination may be satisfied by alias names of a single procedure, or by distinct procedures which specify different work fields.

The following example shows the internal job that JES generates to initially load a program to process batch class X jobs for Initiator 3, assuming a default setting for &XBATCHN.

```
//$$$$$X3     JOB   1, SYS, MSGLEVEL=1
//FAKE        EXEC  $$$$$X3
//GO.SYSIN    DD    DATA,
//                  DCB=BUFNO=1
```

The following is an example of a procedure that an installation might use for a simple file inquiry program that reads inquiry input from SYSIN, checks a file, and prints responses to SYSPRINT:

```
//$$$$$X3     PROC
//GO          EXEC  PGM=FINDPART
//SYSPRINT    DD    SYSOUT=A
//PARTFILE    DD    DSN=PARTFILE.MASTER,
                    DISP=SHR
//SYSUDUMP    DD    SYSOUT=A
```

The following JCL is for the order entry system ecample:

```
//$$$$$X3     PROC
//MDSE        EXEC  PGM=ORDERIN
//MESSAGE     DD    SYSOUT=M
//INVOICE     DD    SYSOUT=(P,,INVC.)
//INVTRY      DD    DSN=MSTRINVT,
//                  DISP=SHR
//ORDERS      DD    DSN=ORDERS,
//                  DISP=MOD
```

- //MESSAGE—the installation might identify class M as a punch class. This will allow the submitter of the execution batch job to route the invoices and messages separately, as shown in the example in "submitting input to an execution batch processing program" above in this section.
- //INVOICE—defines the specially prepared output.
- //INVTRY—uses a master inventory list as a base; it is updated as the orders are received.
- //ORDERS—accumulates the day's orders. ORDERS has a disposition of MOD because the execution batch processing program is periodically started and stopped during the day.
- SYSOUT data sets—messages and invoices.
- SYSIN data sets—batch jobs processed by the execution batch processing program.

2.4.5 Controlling Interpretation and Execution

The JCL for a job, LOGON, or started task is passed through the JCL interpreter and changed into internal text. The job is then available for execution, which occurs as soon as an initiator eligible to process the job becomes available.

JCL interpretation
A job is eligible for JCL interpretation as soon as it is placed on the queue. The interpreter is invoked separately for each job. JES passes to the interpreter various parameters and a pointer to a procedure library.

Procedure library selection
The JES procedure is located in SYS1.PROCLIB. It defines job-related procedure libraries such as:

```
//PROC00      DD
...
//PROC01      DD
...
.
.
.
//PROCnn      DD
...
//anyname     DD
...
```

If multiple data sets are required they must be specified as concatenations in the JES cataloged procedure.

Class-related initialization parameters can specify the library as PROCnn. If the procedure is unspecified or specified but not found, PROC00 is used.

Execution control
Execution is controlled by the console operator by instructing initiators how to process enqueued jobs, as well as by monitoring each job and issuing commands.

JES associates one **logical initiator** residing in JES with each **system initiator** interfacing with JES. The number of active logical initiators (subject to this maximum) is controlled by the operator ($S Inn). The operator can also associate with logical initiators the order in which the classes are selected by JES.

Classes are associated with each initiator during JES initialization, subsequently by the console operator. During execution, each initiator selects nonheld jobs in priority order within their classes in the order specified for that initiator. That is, the lowest priority job in the first nonempty class is selected ahead of the highest priority job of the next class—assuming neither job nor class is held.

Initiators can be automatically started by OS IV/F4 when it is reloaded, if the installation furnishes the appropriate command statements in the corresponding SYS1.PARMLIB member. This approach is used for the standard batch-processing streams of most OS IV/F4 installations, plus any specialized initiators used for timesharing and other nonterminating subsystems.

Stopping an initiator
The console operator stops an initiator with a STOP command, which releases any resources it may hold after the currently executing job—if any—terminates. Among these resources are the scheduler work area data set (SWADS) and any dedicated data sets allocated to this initiator.

Modifying an initiator
To change attributes of an initiator, the console operator issues a MODIFY command, which may alter

- job classes: number and priority.
- upper limit for the scheduling priority for each job class.
- assignment of a uniform scheduling priority within a class, i.e., so that corresponding jobs are selected FIFO (first in, first out).

2.4.6 The Initiator Cataloged Procedure

One initiator cataloged procedure (INIT) must be contained in SYS1.PROCLIB for use by JES in

creating address spaces into which system initiators are initialized. The console operator issues a START command to create one system initiator for each active JES logical initiator. The number of active initiators must be controlled by starting and stopping JES logical initiators.

Two standard initiator procedures are furnished with each OS IV/F4 system:

INIT	initiator without dedicated data sets
INITD	initiator with dedicated data sets corresponding to DD names such as SYSUT1, SYSUT2, SYSUT3, SYSLIN, and SYSLMOD.

2.5 ALLOCATING RESOURCES TO JOBS

Various hardware and software resources are required by each job: virtual storage (when the user specifies ADDRSPC=REAL, a fixed address region in real storage), I/O devices, tape and disk volumes, data sets, and program libraries. Also described in this section are broad aspects of setting up aod processing data sets.

2.5.1 Allocating System Resources

Each batch job explicitly/implicitly requests an address space (real or virtual storage), program libraries, and data sets. Several of these requests are furnished on the JOB statement, for example the type and size of its address space (ADDRSPC and REGION parameters). Other requests are furnished on EXEC statements. Program libraries and other data sets are invariably specified by data definition (DD) statements.

The I/O allocation routines assign units, volumes, and data sets in response to DD statements at step initialization. Considerations and rules for coding DD statements may be found in **FACOM OS IV/F4 Job Management Functions and Facilities, FACOM OS IV/F4 Job Control Language Reference Manual.**

The allocation routines attempt to improve system throughput by satisfying requests in as parallel a fashion as possible. Two types of serialization must be considered: the status of devices eligible for allocation must remain static while they are being selected, and certain devices must be used in a serial manner.

The allocation routines try to satisfy requests in this order, from least serialized to most serialized:

- Allocating data set requests that require no specific units or volumes; for example, dummy and SYSIN/SYSOUT data sets. These requests need not be serialized.
- Allocating data set requests to sharable units, that is, direct access units with permanently resident or reserved volumes mounted on them. These requests need not be serialized.
- Allocating communications devices.
- Allocating mounted volumes and devices that do not need volumes. During this processing, the automatic volume recognition (AVR) function reads serial numbers of any volumes which have been premounted on serialized devices.
- Allocating online but unallocated devices that need volumes mounted by the console operator.
- Allocating all remaining requests, for example, those that need offline devices and/or devices allocated to other jobs which can not be used concurrently.

2.5.2 Storage Allocation

Each job can request an allocation of virtual storage or real storage, specifying ADDRSPC=VIRT (the default value) or ADDRSPC=REAL, respectively, on its JOB or EXEC statements. Main storage is allocated in units of 4096 bytes (4K) called **pages.** The total address space is mapped onto external page storage in units of 65536 bytes (64K).

Virtual storage for a user comprises a collection of pages in main storage and on one or more **paging devices,** specially designated DASDs. Each page accessed by the user exists either in main storage, on a paging device, or both. Not all pages needed by the user are in main storage at once; they are retrieved as needed from the paging device, as shown in Fig. 1.13.

Real storage for a user comprises a collection of pages which are all in main storage during execution of his job. Although the pages may be physically discontiguous (as shown in Fig. 1.12, dynamic address translation hardware presents them as a logically contiguous address space to the user, who need not be aware of page boundaries. Real-storage regions are not paged to paging devices, as are virtual-storage regions; they are assigned at the beginning of each job step for the duration of the entire step.

2.5.3 Specifying Unit Information

The user must explicitly/implicitly provide OS IV/F4 with information it needs to assign one or more devices to a data set. To indicate what unit or type of unit he wants, the user may code one of the following volumes for the UNIT parameter of the corresponding DD statement:

- unit address,
- device type **(generic name).**
- user-assigned group name **(esoteric name).**

The **unit address** is a three-character address comprising the channel, control unit, and unit numbers. For example, UNIT=180 indicates channel 1,

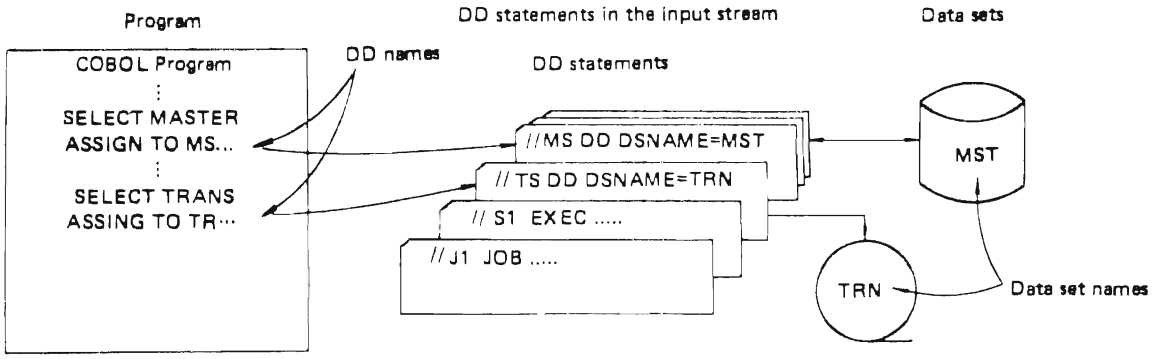


Fig. 2.12 Allocation of data sets to program input/output functions

control unit 8, and unit number 0. Specifying a unit address, however, limits unit assignments; OS IV/F4 can assign only that specific unit. If the unit is being used the job must be delayed or canceled. Unit addresses should only be specified when absolutely necessary.

A **device type** corresponds to a particular set of identical input/output devices. By coding a device type, the user allows OS IV/F4 to assign any available device of this type. For example, UNIT=F478B indicates that he wants the system to assign an available F 47 8B disk drive.

Each installation can also define **user-assigned group names** during system generation to identify a group of devices having a common function. By coding a user-assigned group name, the user allows OS IV/F4 to assign any available device from the group. For example, if the group named DISK includes all F478B and F479B disk drives and the user codes UNIT=DISK, OS IV/F4 can assign any available F478B or F478B device.

If a group contains more than one device type or class (for example, SYSSQ can refer to all tape drives and DASDs), the user should not code the group name when defining an existing data set or requesting a specific volume. The volume on which the data set resides may reguire a device different from the one assigned to it. For example, if the data set resides on a tape reel, it must be assigned to a tape drive.

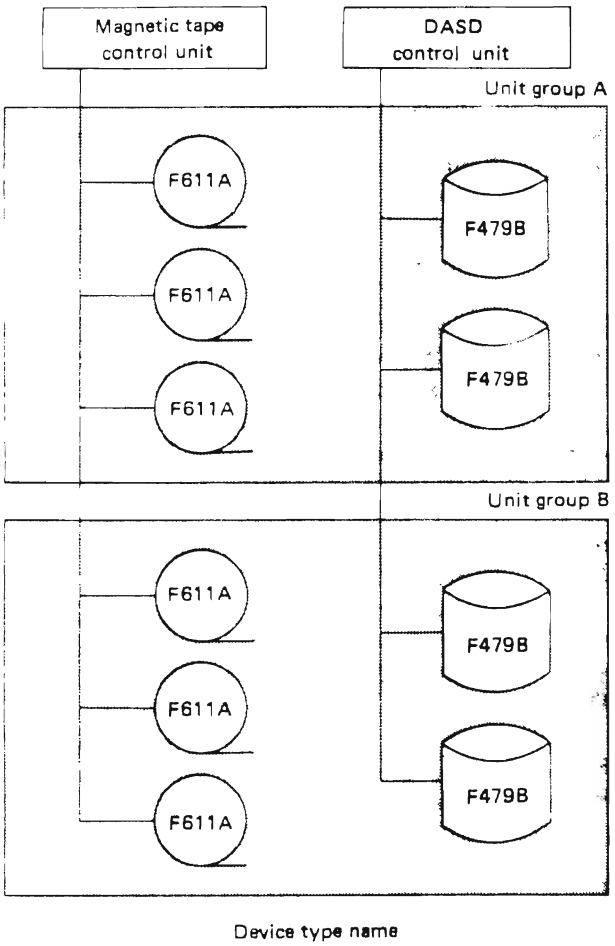


Fig. 2.13 Esample of user-assigned unit groups

Requesting more than one unit

To increase operating efficiency, the user can request multiple units for a multivolume data set or for a data set that may require additional volumes. When each required volume is mounted on a separate device, execution of the job step is not interrupted to allow the console operator to dismount and mount volumes. The user should always request multiple units when the data set can be extended to a new volume if it currently resides on a permanently-resident or reserved volume—permanently-resident and reserved volumes cannot be dismounted in order to mount a new volume.

The user requests multiple units by:

- furnishing a unit-count subparameter in the UNIT parameter.
- requesting parallel mounting.

The user can request **parallel mounting** when making a specific or non-specific volume request. OS IV/F4 counts the number of volumes requested (volume serial numbers specified on the DD statement in cataloged or passed data sets). This is compared with the volume count, if it has been specified, and OS IV/F4 assigns the larger of the specified number of devices.

When the UNIT parameter is unnecessary

OS IV/F4 can often obtain unit information from sources other than the UNIT parameter. In these cases, the user need not code the UNIT parameter.

- When the data set is cataloged. For cataloged data sets, OS IV/F4 obtains unit and volume information from the system catalog. However, if VOL=SER=serial-number is coded on a DD statement for a cataloged data set, OS IV/F4 does not interrogate the catalog. In this case, the user must code the UNIT parameter.
- When the data set is passed from a previous job step. For passed data sets, the system obtains unit and volume information from passed data set information. However, if VOL=SER=serial-number is coded on a DD statement for a passed data set, OS IV/F4 does not interrogate passed data set information. In this case, the user must code the UNIT parameter.
- When the data set utilizes volumes assigned to other data sets via VOLUME=REF syntax. In this case, OS IV/F4 obtains unit and volume information from an earlier DD statement specifying the volume serial numbers or from the catalog.

Determining numbers of volumes/units per request

Before assigning volumes and units for a job step, the allocation routines must determine:

- the maximum number of volumes per request.
- the maximum number of units per request.
- the number of units per job step.

The maximum numbers are calculated because more units than specified may actually be used. The rules for determining unit requirements are explained below under "Units per Job Step".

Minimum Number of Volumes per request

The maximum number of tape volumes or direct access volumes required to satisfy any request is the greater of:

- the volume count specified in the VOLUME parameter.
- the number of volumes whose serial numbers are available.

The number of available volumes is one of the following:

- The number of volumes whose serial numbers are specified.
- The number of volumes obtained through VOL=REF (only if VOL=REF was coded).
- The number of volumes on which the data set resided when it was passed (only if the request is for an existing data set that was passed from a prior step, and neither volume serial number nor VOL=REF was specified).
- The number of volume serial numbers obtained from the catalog (only if the request is for an existing data set not passed from a prior step, and neither volume serials nor VOL=SER was specified).
- The number of volume serial numbers minus the volume sequence number +1 (only if the request is for an existing data set in which the volume sequence number specified is not greater than the number of volume serial numbers). For example, if 8 volume serial numbers will be needed and a volume sequence number of 4 is specified, then the number of volume serial numbers to be allocated would be 5 (=8 − 4 + 1); in this case, the first three volume serial numbers will be discarded, and the fourth volume would become the first volume allocated.
- The unit count specified in the UNIT parameter (only if the unit count specified is greater than the number of volume serial numbers calculated in the previous statement, or if the request is for a new nonspecific direct access volume that does not specify VOLUME=PRIVATE).

When the required number of volume serial numbers for a request is greater than the number of specific volume serial numbers from passed data sets or from the catalog, the remainder of the volumes are assumed to be requests for nonspecific vilumes.

Maximum Number of Units per request

The maximum number of tape or direct access units required to satisfy any request is equal to the greater of:

- the unit count specified in the UNIT parameter.
- the total number of volumes required (if parallel mounting is requested).

When UNIT=AFF is specified, the unit requirements are obtained from the referenced request. The number of units shared with the referenced request is the number of units used by the referenced request.

For direct access volumes, the number of units required to satisfy a request specifying a generation data group (GDG) name is dependent upon the unit requirements of each member of that GDG. Therefore, each member is handled as a single request.

For direct access volumes, the number of units required to satisfy a VSAM data set is dependent upon the unit/volume configuration of the data set. If the data set spans multiple device types, the total

number of units required is determined by the OS IV/F4 catalog manager additional tables will then be generated by the scheduler to cause the allocation of the required number of units. For VSAM data sets, a specified unit count or parallel mount may be overridden by the system once the unit requirements for the data set are determined.

Number of Units per job step
The number of units required for a job step is not necessarily the sum of the unit requirements for each request.

The following rules tend to reduce the total unit requirements for a step:

- A volume can be allocated only to one unit. Therefore, if more than one request asks for the same volume, all requests will be allocated to the same unit.
- For DASDs, storage and/or public requests can be allocated to the same volume. Therefore, two or more such requests may be satisfied with one unit.

The following rules tend to increase the total unit requirements for a step:

- A permanently resident or reserved volume cannot be demounted. Therefore, a volume which is permanently resident or reserved will be assigned its own unit (where it is mounted) even if, through JCL specification, it was to share a unit with one or more other volumes.
- For direct access, when more than one request within a job step requires the same volume, that volume must be shared. Therefore, a direct access volume which is required by more than one request will be assigned its own unit even if, through JCL specification, it was to share a unit with one or more other volumes.
- For direct access, a VSAM data set will require additional units if the data set resides on more than one device type.
- For direct access, an additional unit is required for a private catalog volume if it is associated with and/or used to retrieve volume information about a particular data set.
- For direct access, when a GDG name is specified, additional units may be requested to satisfy the device type requirements of each individual member of the GDG.
- For tape, when conflicting unit assignments are specified for tape volumes, the volume involved in the conflict will be assigned its own unit. For example, such a conflict would exist for VOLUM2 in the following DD statements:

```
//DD1 DD  UNIT=TAPE,
          VOL=SER=(VOLUM1,
          VOLUME2)
//DD2 DD  UNIT=TAPE,
          VOL=SER=(VOLUM2,
          VOLUM3)
```

In this case, three units—one for each volume—would be assigned. If the user had requested via unit affinity that the same tape unit be used for both DD1 and DD2, then only one unit would have been assigned.

2.5.4 Volumes

A **volume** is a media unit for data such as a reel of magnetic tape, a disk pack, or a drum. Each volume is identified by a **volume serial number** of 1-6 alphanumeric characters, which is typically written onto the volume itself in machine-readable form and also onto an external label to facilitate handling/recognition of the volume by the console operator.

Volume attributes
Attributes serve to determine eligibility of a volume for dismounting and to control which data sets can be allocated to it. **Volume sharing** is defined as usage of a volume for two or more data sets within one job step or for two or more data sets accessed by concurrently-executing job steps.

Attributes of magnetic tape and direct-access volumes are the **mount attribute** and **use attribute.** The **nonsharable attribute** may be assigned only to direct access volumes.

Mount and use attributes
Every volume is assigned a mount and use attribute either when the OS IV/F4 system is loaded (IPL) via a VATLST or when the volume is first used by a job. The **mount attribute** controls volume demounting. The **use attribute** helps control allocation of mounted volumes to data set requests. The mount and use attributes are as follows:

Mount
- Permanently resident
- Reserved
- Removable

Use
- Public
- Private
- Storage
- Scratch

A **private volume** can only be allocated when its volume serial numbers are explicitly or implicitly specified.

A **public volume** is a direct-access volume eligible for allocation of temporary data sets when no specific volume is requested (and PRIVATE is not specified).

A **storage volume** is a direct-access volume eligible for allocation of both nontemporary and temporary data sets when no specific volume is requested (and PRIVATE is not specified). Storage volumes are primarily used for non-temporary data

sets; temporary data sets will be assigned to storage volumes only if they cannot be assigned to public volumes.

A magnetic-tape **scratch volume** is used temporarily within one job. After the latter completes, the reel is left mounted for use by subsequent jobs.

The following points list the mount attributes and describe how the mount and use attributes are assigned to a volume:

- **Permanently resident volumes** cannot be dismounted. Only direct access volumes can be permanently resident. Although the user may designate all direct access volumes as permanently resident in the **volume attribute list** (VATLSTxx) in SYS1.PARMLIB, the following volumes are always permanently resident:
  1) volumes that cannot be physically demounted, such as drum-storage volumes.
  2) the IPL volume.
  3) any volume containing system data sets such as SYS1.LINKIB and SYS1.PROCLIB.

  Any installation can assign to a permanently resident volume a use attribute of "public," "private," or "storage" in the VATLST member of SYS1.PARMLIB; the default attribute is "public".
- **Reserved volumes** remain mounted until the console operator issues an UNLOAD command. Both direct access and tape volumes can be reserved. A volume becomes reserved as a result of a MOUNT command or a VATLST entry (for direct access devices only). A volume is usually designated as "reserved" to avoid repeated mounting and dismounting when used by many jobs.

  An installation can designate a reserved direct access volume as "public," "private," or "storage." The use attribute is assigned to the volume either in the VATLST member of

Volume state	Temporary data set	Nontemporary data set	How assigned	How demounted
	Type of volume request			
Public/ Permanently resident*	Nonspecific or specific	Specific	VATLST entry or by default	Always** mounted
Private/ Permanently resident*	Specific	Specific	VATLST entry	Always** mounted
Storage/ Permanently resident*	Nonspecific or specific	Nonspecific or specific	VATLST entry	Always** mounted
Public/ Reserved*	Nonspecific or specific	Specific	VATLST entry or MOUNT command	UNLOAD or VARY OFFLINE commands
Private/ Reserved (Tape and direct access)	Specific	Specific	VATLST entry or MOUNT command (MOUNT command only for tape.)	UNLOAD or VARY OFFLINE commands
Storage/ Reserved*	Nonspecific or specific	Nonspecific or specific	VATLST entry or MOUNT command	UNLOAD or VARY OFF LINE commands
Public/ Removable (Tape and direct access)	Nonspecific or specific	Specific	VOLUME=PRIVATE is not coded on the DD statement. (A nonspecific request and a temporary data set for tape also causes this assignment.)	When unit is required by another volume
Private/ Removalbe (Tape and direct access)	Specific	Specific	VOLUME=PRIVATE is coded on the DD statement. (Specific request or a nontemporary data set for tape also causes this assignment.)	At job termination or when the unit is required by another volume.

* Direct access volumes only.

** Note that VARY OFFLINE effectively accomplishes dismounting.

Fig. 2.14 Summary of volume type and data set requests

SYS1.PARMLIB or in a parameter of the MOUNT command, depending on how the volume becomes reserved.

A reserved tape volume is always private.

- **Removable volumes** are neither permanently resident nor reserved. They can be demounted either after the end of the job in which they are last used or when the unit on which the volume is mounted is needed for another volume.

  A use attribute of "private" or "public" can be assigned to a removable direct access volume when the PRIVATE volume subparameter is coded or omitted, respectively.

  A removable tape volume can be assigned a use attribute of "public" or "private". The use attribute of "public" is assigned when the PRIVATE subparameter is omitted, a nonspecific volume request is made, and the data set is temporary (a system-generated data set name or a disposition of DELETE). The use attribute of "private" is assigned when the PRIVATE subparameter is coded, a specific volume request is made, or the data set is nontemporary (a non system-generated data set name or a disposition other than DELETE.)

Fig. 2.14 summarizes the types of volumes that satisfy specific or nonspecific volume requests for temporary or nontemporary data sets; how these attributes are assigned; and how volumes are demounted.

Nonsharable attribute

OS IV/F4 assigns the nonsharable attribute to direct access volumes that may require demounting during execution of a step. When a volume has the "nonsharable" attribute, it cannot be assigned to any other data set until the nonsharable attribute is removed. It is removed at the end of the step that was using it as nonsharable.

The nonsharable attribute is never assigned to a permanently-resident or reserved volume. It is always assigned to a volume used to satisfy any of the following requests:

- specific volume request that specifies more volumes than devices.
- a nonspecific volume request if it specifies PRIVATE and a volume count greater than the number of devices.
- a request for unit affinity with an earlier data set defined in the job step, when the data sets reside on different volumes.
- a request for deferred mounting of the volume on which the data set resides.

Fig. 2.15 shows the OS IV/F4 action for sharable and nonsharable requests.

To illustrate when the nonshaarable attribute is set, suppose JOBA has indicated a need for two volumes but only one unit is specified. In this case, the operator will later have to mount JOBA's second volume. JOBB is willing to share the first volume mounted. If JOBA were to request mounting of the second volume while JOBB is executing, JOBB would fail. To avoid this problem, the system marks JOBA's volume request as "nonsharable" so that no other job can use these volumes while JOBA is executing.

Satisfying specific volume requests

In the following cases, OS IV/F4 can satisfy a request for a specific volume that is already mounted:

- The volume is permanently resident or reserved. The volume is assigned regardless of the requested use attribute, and the use attribute is not changed by the allocation.
- The DASD volume is removable, does not have the nonsharable attribute, and is being used by a concurrently executing step. If the user's request would make the volume nonsharable, OS IV/F4 delays assigning the volume to his job until all other job steps using the volume have terminated.
- The DASD volume is removable but not allocated. The use attribute (private or public) assigned to the volume is determined by presence or absence of a PRIVATE subparameter.
- The tape volume is a scratch volume and is not in use. The use attribute of private is assigned to the volume if the request is for a permanent data set or if PRIVATE is coded.

Fig. 2.16 shows how user requests affect use attributes.

The request is:	The volume is allocated:	
	Sharable	Nonsharable
Sharable	allocate the volume	wait*
Nonsharable	wait*	wait*

* The operator has the option of deleting the request. The request will always fail if waiting is not allowed.

Fig. 2.15 Description of volume allocations with respect to sharable requests

The request is:	The volume is:	
	Private	Public
Private	stays private	changes to private
Public	stays private	stays public

Fig. 2.16 Private and public volume requests

Satisfying nonspecific volume requests
There are four types of **nonspecific volume requests:**

- a private volume for a temporary data set
- a private volume for a nontemporary data set
- a nonprivate volume for a temporary data set
- a nonprivate volume for a nontemporary data set

OS IV/F4 satisfies these requests as described below. Since it satisfies the first two types of requests in the same way, they are described jointly.

- For a nonspecific volume request for a private DASD or tape volume, OS IV/F4 requests the console operator to mount a volume. He should furnish a volume with most/all space available, giving the user control over all space on the volume. Once mounted, the volume is assigned the use attribute of "private."
- For a nonspecific volume request for a non-private direct access volume that is to contain a temporary data set, OS IV/F4 attempts to a ssign a public or storage volume that is already mounted. If no space is available, it requests the operator to mount a removable volume.

  If OS IV/F4 selects a mounted volume, its use attribute remains the same. If a removable volume is mounted, the system assigns it the use attribute of "public."

  For a nonspecific volume request for a non-private tape volume that is to contain a temporary data set, OS IV/F4 assigns a scratch volume that is already mounted, or it requests the operator to mount a tape volume. Once mounted, OS IV/F4 assigns the volume the use attribute of "public."
- For a nonspecific volume request for a non-private direct access volume that is to contain a nontemporary data set, OS IV/F4 assigns a storage volume if one is mounted on an eligible device. Otherwise, it treats the request as a nonspecific volume request for a private volume.

  For a nonspecific volume request for a non-private tape volume that is to contain a nontemporary data set, OS IV/F4 treats the request as a nonspecific volume request for a private volume.

Deferred mounting of volumes
If a job step defines a data set that may not be needed, depending on conditions determined during execution, the user can request (using the DEFER subparameter) that OS IV/F4 not mount volume(s) containing the data set until the latter is opened. This can eliminate unnecessary operator mounting of volumes on direct access devices. No other job step can use **deferred-mount volumes** until the job step specifying DEFER ends. If DEFER is coded for a new data set which could be placed on a direct access device, DEFER is ignored.

I/O load balancing
OS IV/F4 attempts to satisfy nonspecific volume requests so as to optimize the balance on channels, control units, and devices. Whenever the step initiator chooses a device in this situation, it scrutinizes the current load on each of these I/O facilities and chooses that device and path—among all devices capable of satisfying the nonspecific request—which is lightest loaded at present. Loads are estimated according to intensity of I/O activities on these devices and paths over a recent fixed-length time interval.

2.5.5 Data Sets

This section discusses four kinds of data sets: temporary, non-temporary, dummy, and dedicated. **Temporary data sets** are created, used, and deleted within a single job; they are used for intermediate working storage. **Nontemporary data sets** are typically retained between two or more related jobs. All non-temporary data sets on DASD must have **data set names**, identifiers of 1—44 characters divided into **simple names** (at most eight alphanumeric characters) separated by periodes (".").

Dummy data sets are used to bypass I/O functions requested by executing programs. If a program reads a dummy input data set—indicated by a first operand of "DUMMY" on the corresponding DD statement—it receives an immediate end-of-file signal from OS IV/F4. If a program writes a dummy output data set, output records are automatically discarded by OS IV/F4 data management.

Dedicated data sets may be permanently allocated to an initiator by the cataloged procedure which started it. Dedicated data sets are used for the same functions as temporary data sets; their contents may not be passed between jobs. The performance

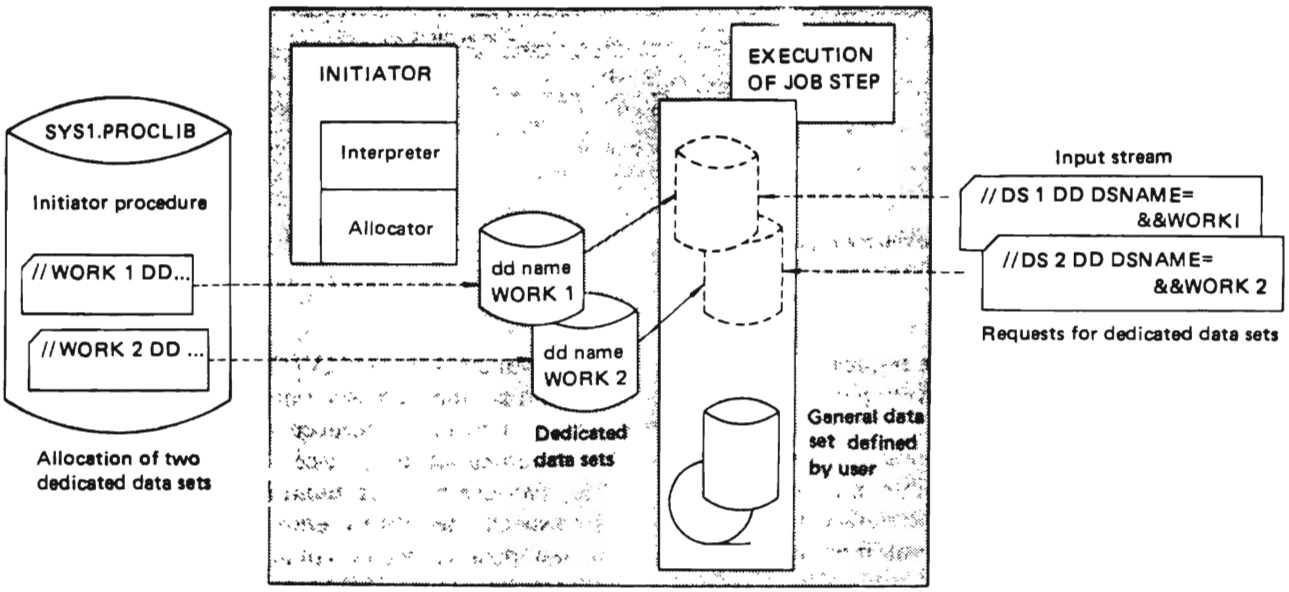


Fig. 2.17 Definition and use of dedicated data sets

advantage of dedicated data sets over temporary data sets is primarily in reduced CPU overhead and turnaround delays. Each time a temporary data set is allocated or deallocated, OS IV/F4 expends considerable effort in selecting an available unit, reading its VTOC, and allocating space on this volume. Since dedicated data sets are pre-allocated, most of this overhead is avoided. Another advantage of dedicated data sets is that an installation can deliberately assign them to channels and devices to achieve balance among these I/O facilities:

The user requests a dedicated data set conditionally, since the initiator which selects his job may/may not have dedicated data sets allocated to it. If not, his request will be automatically satisfied with a temporary data set allocated by OS IV/F4. If a corresponding dedicated data set exists but is too small to satisfy his request, an ordinary temporary data set will be allocated. Hence, the request will be satisfied with one of the dedicated data sets for this initiator (if any) only if certain attributes of size and format match pre-existing attributes of the corresponding dedicated data set, as shown in Fig. 2.17.

2.5.6 Program Libraries

The system library of executable programs is named SYS1.LINKLIB; it stores the OS IV/F4 routines, compilers, Assembler, utility programs, linkage editor, loader, sort programs, and—at the option of each installation—frequently-used application programs. Users need not specify LINKLIB as their program library, since it is the default source of executable programs.

Each installation—or user—can define an arbitrary number of private libraries of executable programs. A **private library** must be defined in each job whre it is used. If it is used in only one or two steps of a job, it can be defined as a step library by a STEPLIB DD statement of the following form:

```
//STEPLIB  DD  DSN=data-set-name, . . .
```

A STEPLIB statement must be furnished in each step where the library is to be used.

A **job library** is analogous to a step library, except that it is available for all steps of a job:

```
//JOBLIB  DD  DSN=data-set-name, . . .
```

The JOBLIB statement immdiately follows the JOB statement, preceding the EXEC statement for the first job step.

It is possible to furnish both job and step libraries for a single job. In this case, the directory of the specified step library is searched first for the requested program (member of the program library, which is a partitioned data set). If the requested program is not found in the step library, OS IV/F4 searches the directory of the job library (if furnished). If the requested program is in neither the step nor job library, OS IV/F4 searches for it in LINKLIB.

A **temporary program library** can be created, used, and deleted within a single job. A temporary library is merely a temporary partitioned data set used as a program library.

2.5.7 Status and disposition of data sets

Disposing of data sets at the end of a job step is

known as **disposition processing.** The user requests disposition processing for a non-VSAM data set by coding the DISP parameter on the DD statement defining the data set. (VSAM data sets are handled differently. For information on VSAM, the user should refer to the **FACOM OS IV/F4 VSAM Functions and Facilities** and **FACOM OS IV/F4 AMS Commands Reference Manuals.**) In the DISP parameter, the user can code the following subparameters:

- Data set status as the first subparameter, indicating whether the data set is new, old, sharable with other jobs, or modifiable.
- Normal disposition as the second subparameter, indicating how the data set should be handled if the job step terminates normally.
- Conditional disposition as the third subparameter, indicating how the data set should be handled if the job step terminates abnormally.

If the user omits any of these subparameters — or if he omits the DISP parameter entirely — OS IV/F4 supplies default values, as described under "Default Disposition Processing."

Data set status

The user indicates the status of a data set by coding one of the following values:

- NEW — the data set is being created in this job step.
- OLD — the data set existed before this job step.
- SHR — the data set existed before this job step and can be read simultaneously by other jobs.
- MOD — the OS IV/F4 assumes the data set exists and will position the read/write mechanism after the last record in the data set; if OS IV/F4 cannot find volume information for the data set, it assumes the data set will be created in the job step.

By coding SHR, the user permits shared control of the data set; his access is usually restricted to reading the data set, as opposed to writing or updating it. By coding NEW, OLD, or MOD, the user implicitly requests **exclusive control** of the data set. Shared and exclusive control are described in this chapter under "Insuring Data Set Integrity".

Specifying a disposition for the data set

The user can specify a **normal disposition** for the data set, used if the job step terminates normally (successfully). He can also (or alternatively) specify its **conditional disposition,** to be used if the job terminates abnormally.

For normal disposition, the second parameter of the DISP parameter specifies that the data set be:

- deleted, by coding DELETE;
- kept, by coding KEEP;
- cataloged, by coding CATLG;
- uncataloged, by coding UNCATLG; or
- passed, by coding PASS.

NOTE: Disposition of a data set depends entirely on the DISP parameter. However, disposition of volumes on which the data set resides also depends on the volume status when the volume is dismounted.

For conditional disposition (third subparameter of the DISP parameter), the user can code any of the above values with the exception of PASS.

Data sets allocated to steps that abnormally terminate and do not allow automatic restart are disposed of as specified by the conditional disposition. If a job step abnormally terminates during execution and a conditional disposition is not specified, OS IV/F4 follows the normal disposition. If a job step fails during step allocation:

- any data set created in that job step is deleted.
- any data set that existed before that job step is kept.

Disposition processing differs for data sets on DASD or magnetic tape volumes. The DASD **volume table of contents** (VTOC) contains control blocks describing non-VSAM data sets and available space on the volume. How to manage tape and direct access volumes when specifying a particular disposition is described below.

When the user specifies KEEP or PASS for a cataloged data set, OS IV/F4 assumes that he wants the data set recataloged if volume information was obtained from the catalog and if OS IV/F4 determines that the catalog entry must be updated. If the job step performs catalog maintenance and the user wishes to avoid recataloging, he should refer to the data set by its specific unit and volume serial number when coding the corresponding DD statement.

Deleting a data set

By specifying DELETE, the user requests that the data set's space on the volume be released at the end of the job step (when coded as the normal disposition) or if the step abnormally terminates (when coded as the conditional disposition). If the data set resides on a public tape volume, OS IV/F4 rewinds the tape and makes this volume available for use by other job steps. If the data set resides on a private volume, the tape is rewound and unloaded, and a KEEP message is issued. If the data set exists on a DASD volume, OS IV/F4 removes the control block describing the data set from the VTOC; its space on the volume is then available for allocation to other data sets.

In one case, however, a data set on a direct access volume will not be deleted, even if the user specifies DELETE: when the expiration date or retention period has not expired. The user can specify how long a data set should be kept by assigning a retention period or expiration date in the LABEL parameter on the corresponding DD statement.

If the user wishes to delete a cataloged non-VSAM

data set, the data set entry in the system catalog is also removed, provided OS IV/F4 obtained volume information for the data set from the catalog (that is, its volume serial number was not coded on the DD statement). If OS IV/F4 did not obtain volume information from the catalog, the data set is deleted but its entry in the catalog remains. If OS IV/F4 encounters an error while attempting to delete a data set, its entry in the catalog will not be removed. (The data set may or may not be deleted, depending on where the error occurs). The user can furnish an access method services DELETE command or the JSEPROGM UNCATLG command to delete a non-VSAM entry from the catalog.

DELETE is the only valid conditional disposition for a data set with no name or a temporary name. If a disposition other than DELETE is specified, OS IV/F4 assumes DELETE.

Keeping a data set

By specifying KEEP, the user asks OS IV/F4 to keep a data set intact until a subsequent job step requests that the data set be deleted, or until the expiration date or retention period—indicating the length of time a data set must be kept—has elapsed. If he does not specify a time period, OS IV/F4 assumes a retention period of zero days.

For DASD data sets, the VTOC entry describing the data set and the data set itself are kept intact. For data sets on tape, OS IV/F4 rewinds and unloads the volume and issues a KEEP message to the operator.

Cataloging a data set

Cataloging allows the user to keep track of and retrieve data sets. Data sets can be cataloged in the system master catalog or in user (private) catalogs. When retrieving a cataloged data set, the user need not specify volume information; he needs only to code the DSNAME parameter and a DISP status other than NEW.

To catalog a non-VSAM data set, the user codes CATLG for its disposition; OS IV/F4 creates an entry in the catalog that points to the data set. The CATLG disposition implies KEEP.

The user can specify a CATLG disposition for an already-cataloged data set. This should be done when lengthening the data set with additional output (MOD) such that the data set may exceed one volume. If OS IV/F4 obtained volume information for the data set from the catalog (that is, if the volume serial number is omitted from the DD statement) and if the user codes DISP =(MOD,CATLG), OS IV/F4 updates the entry to include serial numbers for any additional volumes.

A collection of cataloged data sets kept in chronological order can be defined as a **generation data group** (GDG). The entire GDG is stored under a single data set name; each data set within the group, a **generation data set,** is associated with a generation number that indicates how far removed the data set is from the original generation. For more information on defining and creating generation data groups, the user should consult the **FACOM OS IV/F4 Data Management Functions and Facilities.**

Uncataloging a data set

To remove the entry describing a non-VSAM data set from the catalog, the user codes UNCATLG for its disposition. Specifying UNCATLG does not request the initiator to delete the data set — only the catalog reference is removed. If the user requests access to this data set in a subsequent job, he must include volume information on his DD statement.

Passing a data set

If several steps in a job need the same data set, each step using the data set can pass it for use by subsequent steps. A data set can only be passed within one job.

To pass a data set, the user codes PASS as its normal disposition; he cannot specify PASS for a conditional disposition. He continues to code PASS in each step referencing the data set until the last time it is used in the job. At this time, he should assign it a final disposition within this job.

Specifying the name of a passed seet without citing its volume serial number is called **receiving** the data set. Identical data set names (whether or not they reference the same data set) can be passed at the same time. Such identical data set names are received in the same order in which they are passed. A data set name that has been passed N times can be received no more than N times. A data set cannot be passed and received within the same step.

Disposition processing of unreceived passed data sets

A data set can be passed by a job step and not subsequently received by another job step. In such a case, if the earlier job step terminates abnormally, unreceived data sets that specified a conditional disposition when passed are processed as specified in their conditional disposition, with four exceptions. If the conditional disposition requires updating of a user catalog:

- and CATLG is specified for a data set with a first-level qualifier of a catalog name or alias, the data set will not be cataloged.
- and UNCATLG or DELETE (of a cataloged data set) is specified for a data set with a first-level qualifirer of a catalog name or alias, the data set will not be uncataloged.
- and CATLG is specified for a data set with no qualifier or with a qualifier that is not a catalog name, the data set will be cataloged in the master catalog.
- and UNCATLG or DELETE (of a cataloged data set) is specified for a data set with no qualifier or with a qualifier that is not a catalog name, an

attempt will be made to uncatalog the data set from the master catalog.

Data sets that do not specify a conditional disposition—those specified as (NEW, PASS) in this job—are deleted; all others are kept.

If no job step abnormally terminates, unreceived data sets specified with DISP=(NEW,PASS) are deleted; other unreceived data sets are kept.

Default disposition processing

If the user omits a DISP parameter altogether—or omits one or more subparameters—OS IV/F4 furnishes various default values.

If the user omits the status subparameter, OS IV/F4 assumes a value of NEW. If he omits the second or third subparameters, OS IV/F4 determines how the data set should be handled according to its status. Data sets that existed prior to this job step are automatically kept (data sets for which OLD, SHR, or MOD is coded and the volume information is available). Data sets created during this job step are automatically deleted (data sets for which the user coded NEW or MOD and volume information was not available, or for which he did not code a status).

If a step abnormally terminates before it actually begins execution (for example, during allocation of units and volumes or direct access space), OS IV/F4 ignores the disposition furnished by the user, automatically keeping existing data sets and deleting new data sets.

For example, if the user codes:

DISP=(,PASS,CATLG)

OS IV/F4 assumes the data set is new. If the job step abnormally terminates during execution, OS IV/F4 will catalog the data set, as instructed by the conditional disposition of CATLG. If the step abnormally terminates befor it actually begins execution, OS IV/F4 will delete the data set, since it is a new data set.

Bypassing disposition processing

If the user defines a data set as a dummy data set, OS IV/F4 ignores any user-furnished DISP parameter and omits disposition processing. For details the reader should consult Sections 2.5.5 and 2.7.2.

Insuring data set integrity

The user can request either **exclusive control** of various data sets in his job—allowing no other concurrently-executing job to access these data sets—or **shared control,** allowing them to be concurrently accessed by other jobs also specifying shared control. The process of securing control of data sets is called **data set integrity processing.**

Data set integrity processing avoids conflicts between two or more jobs requesting simultaneous access to the same data set. For example, a job named READ and another named MODIFY may simultaneously request the data set FILE. READ needs to read and copy certain records, MODIFY deletes some records and changes other records in the FILE data set. If both jobs access FILE concurrently, READ cannot accurately access its records—cannot be sure of the integrity of the data set. MODIFY should gain exclusive control of the data set; READ can share control of FILE with other jobs needing read-only access to the data set. The user should indicate the type of control in the DISP parameter on the DD statement defining the data set.

Exclusive control of a data set

When a job has exclusive control of a data set, no other job can use the data set until the controlling job terminates. A job should acquire exclusive control over a data set prior to modifying, adding, or deleting records.

In some cases, the user may not need exclusive control over the entire data set. He can request exclusive control of individual blocks by coding READ, WRITE, and RELEX macro instructions. These instructions are described in the **FACOM OS IV/F4 Data Management Macro Instructions Reference Manual.**

To request exclusive control of a data set, the user codes NEW, OLD or MOD as the first DISP subparameter.

Shared control of a data set

A data set on a direct access storage device can be used concurrently be several jobs; however, none of the jobs should change the data set drastically. To request shared control, the user codes SHR as the first DISP subparameter. If more than one step of his job needs the data set, he must code SHR every time he defines it if it is to be available to other concurrently-executing jobs. Data set integrity processing is performed once per job; a data set has either shared or exclusive control. If the user codes NEW, OLD or MOD on any reference to a data set, OS IV/F4 assigns exclusive control to the data set for the entire job; a references requesting exclusive control overrides any number of references permitting shared control.

Example of disposition processing

```
//DISP   JOB    ... MSGLEVEL 1
//S1     EXEC   PGM=JSEBR14
//D1     DD     DSN=ABC, DISP=(SHR, KEEP)
//D2     DD     DSN=SYSA, DISP=(OLD, DELETE,
//                 UNCATLG)
//D3     DD     DSN=SYSB, UNIT=F478B,
//                 VOL=SER=F478B1,
//                 SPACE=(CYL, (4,2,1)),
//                 DISP=(NEW, KEEP, CATLG)
//D4     DD     DSN=&&SYS1, DISP=(MOD' PASS),
//                 UNIT=F478B, VOL=SER=F478B2,
//                 SPACE=(TRk, (15,5,1))
//S2     EXEC   PGM=JSEBR14
//D1     DD     DSN=&&SYS1, DISP=(MOD, DELETE),
//                 UNIT=F4788,
//                 VOL=SER=F478B2,
//                 SPACE=(TRK, (15,5,1))
```

1. The JOB statement requests that all JCL statements and system messages be printed.
2. D1 in step S1 defines a data set that already exists and can be shared with other data sets. It is to be kept on the volume after this job step.
3. D2 in S1 defines a data set that already exists, cannot be shared with other data sets, is to be deleted at the end of the job step, and is to be uncataloged if the program abnormally terminates.
4. D3 in S1 defines a new data set that is to be assigned a specific volume(F478B1) on a F478B device. The data set is to be kept on the volume at the end of this job step if the step terminates normally, but it is to be cataloged if the program abnormally terminates.
5. D4 in S1 defines a temporary data set that is to be created in this job step. It is to be assigned to volume F478B2 on a F478B device with 15 primary tracks, 5 secondary tracks and a directory. This data set is to be passed to subsequent steps in this job.
6. D1 in S1 refers to the temporary data set defined in D4 of S1. When this step completes, the data set is to be deleted.

2.5.8 Automatic Volume Recognition (AVR) and Volume Setup

To facilitate smooth and efficient mounting of tape and DASD volumes, OS IV/F4 furnished AVR and a JES-based setup facility.

Automatic volume recognition (AVR)
This facility recognizes a volume when the operator mounts it on a previously-empty drive. Hence, the operator takes the initiative in selecting and mounting volumes, in contrast to the **demand mounting** option, where OS IV/F4 determines which, where, and when volumes are to be mounted.

Immediately after a labelled volume is mounted, AVR reads the volume label and enters it into an internal OS IV/F4 table. If the volume is unlabelled (necessarily magnetic tape) AVR will unload it after attempting to read the non - existent label. Otherwise, AVR will accept standard, JIS, and ANSI labels, plus most non-standard labels. Each installation must install its own routines for automatic recognition of non-standard tape labels.

AVR permits console operators to pre-mount reels and packs required by jobs requiring setup, as described below. Since the operator knows in advance which volumes will be required for each job, he can premount them on available drives prior to releasing the job. This reduces delays in setting up jobs to an absolute minimum.

Setup
The user informs OS IV/F4 which mountable volumes are needed for his job by furnishing one or more SETUP statements immediately after his JOB statement:
/*SETUP vol-ser-no-1 [,vol-ser-no-2] [,vol-ser-no-3]

Each volume serial number should indicate — explicitly if not implicitly — what type of volume (tape reel, 100-megabyte disk pack, 200-megabyte pack, etc.) is required; in case of a tape reel, the user should indicate whether the file protection ring should be inserted/removed by the console operator prior to mounting the reel.

Each job furnishing at least one SETUP statement is automatically placed onto the hold queue by JES. After the operator has retrieved requested volumes from the volume library, he may premount some/all of them if the OS IV/F4 system includes the AVR option. Whether he mounts volumes or not, he then can release the job, since its volumes are either mounted or available for quick mounting near corresponding drives.

2.6 JOB EXECUTION

In a typical OS IV/F4 system, many batch and time-sharing jobs execute simultaneously most of the time. Each batch initiator selects, initiates, and terminates jobs non-stop so long as it remains active (console operator has not issued a STOP command to it) and at least one class of jobs for which it is eligible is non-empty.

As jobs execute, individual steps may be executed or bypassed according to user-furnished conditions (COND parameter) typically based on the success/failure of prior steps.

The following subsections describe how OS IV/F4 processes multiple batch jobs simultaneously, executes and terminates each job step, conditionally executes or bypasses steps, and terminates jobs.

2.6.1 Processing Multiple Jobs

It is typical to service ten batch streams plus numerous time-sharing users from a single OS IV/F4 system. The principal reason for performing multiple independent computations concurrently is to utilize the powerful Fujitsu **central processing units** (CPUs) fully. The speed disparity between a typical CPU and a typical **file peripheral** (tape or disk drive with its associated channel and control unit) is quite large. Hence, OS IV/F4 services many different jobs with one CPU (two CPUs in the case of a multiprocessor configuration) and a large collection of file peripherals. Some file peripherals are dedicated to particular users, others are shared by many users.

One unit of work for a CPU is called a **task** in OS

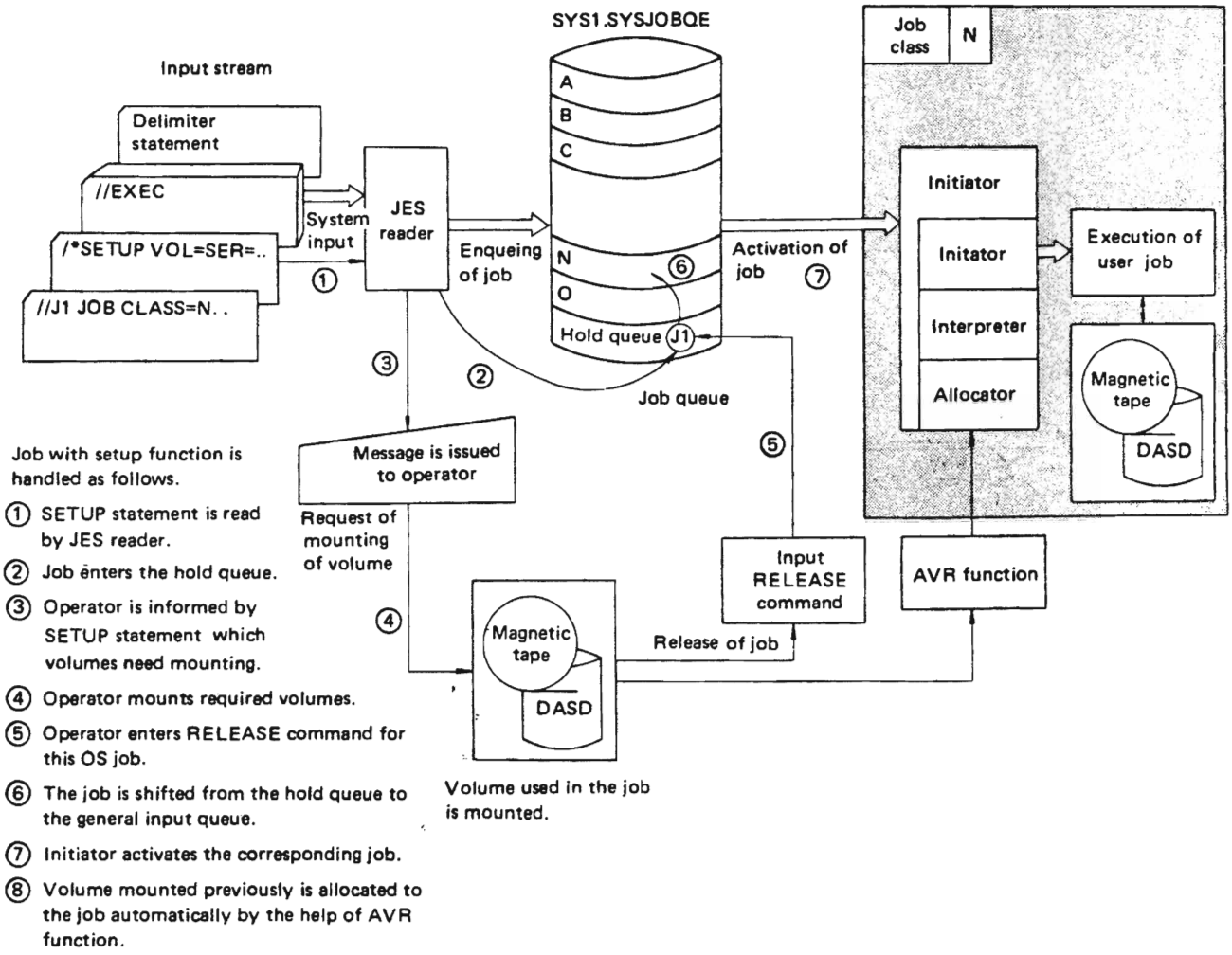


Fig. 2.18 Setup function and activation of job

IV/F4. Since OS IV/F4 processes many tasks simultaneously, it is considered to be a **multitasking** or **multijobbing** operating system. If two or more CPUs share a pool of main storage and peripherals, the hardware configuration is called a **multiprocessing** or **multiprocessor** system.

2.6.2 Execution of Jobs and Job Steps

Each job begins with a JOB statement and ends with a null statement (or, by default, the delimiter statement for the last SYSIN data set or the last JCL statement). Each job step begins with an EXEC statement which either names a particular program to be executed or invokes a cataloged or in-stream procedure, as shown in Figs. 2.18 and 2.19. Expansion of procedure calls into several EXEC, DD, and other JCL statements is described in the FACOM OS IV/F4 Job Management Functions and Facilities.

Job steps and initiators

Each job is fully processed by one initiator, which is dedicated to this job until it terminates. Each initiator (and the stream of jobs it processes) occupy one virtual address space of 16 million bytes, which is defined at the time the console operator starts the initiator.

Each initiator starts a job step after allocating its device and data set resources by issuing an ATTACH macro instruction naming the program for that step. The first step to be attached in this way is called the **job step task.** Ordinarily this is the only task in this address space; however, the job step task can issue ATTACH macro instructions to start one or more **subtasks** which execute concurrent with one another, their job step task (**parent task**), and job step tasks for other active initiators.

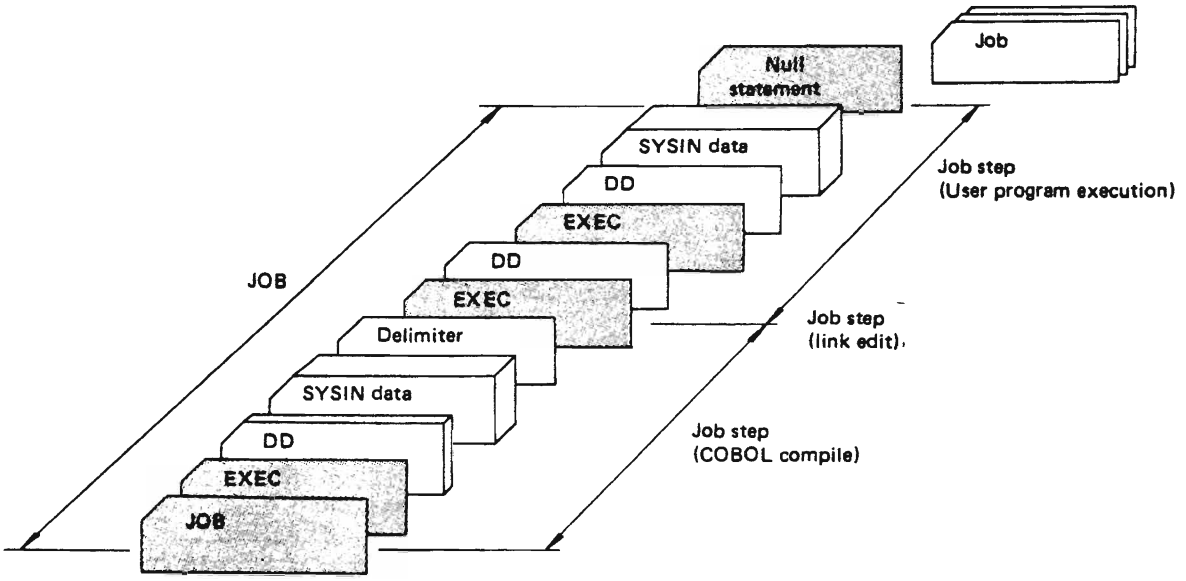


Fig. 2.19 Jobs and job steps

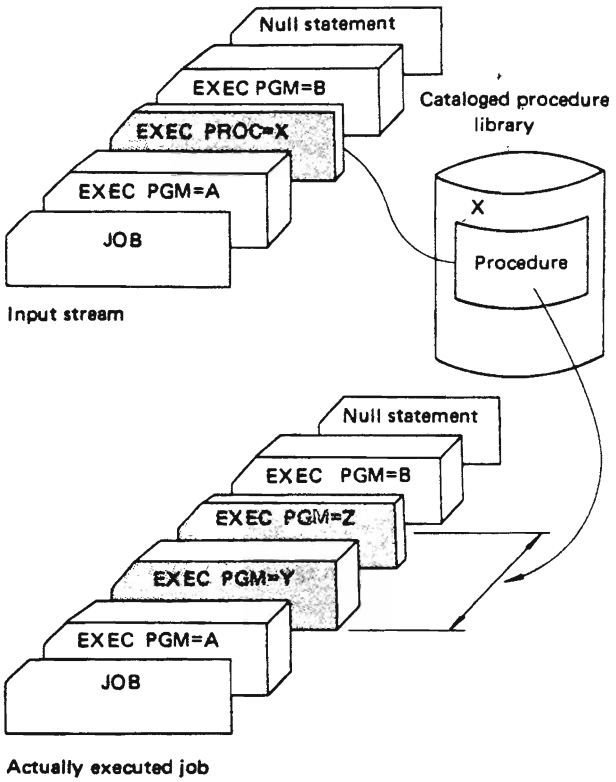


Fig. 2.20 Using a cataloged procedure

Dispatching priority

Earlier in this chapter, the concept of job selection priority was discussed. **Dispatching priority** is different from selection priority; it specifies the relative priority of any task for controlling the CPU. The user can set the dispatching priority for a particular job step by furnishing a DPRTY parameter on the corresponding EXEC statement:

$$\text{DPRTY} = (d_1, d_2)$$

where d_1 is the broad job-step priority and d_2 is a refinement of this priority. The priority used by OS IV/F4 is $616*d_1 + d_2$, $0 \leq d_1 \leq 13$, $0 \leq d_2 \leq 15$.

Higher values of the dispatching priority give it preference over other tasks (from the same or other address spaces) for gaining control over the CPU. If d_1 is omitted, its value defaults to the selection priority (PRTY parameter) of the entire job. The default value for d_2 is 11.

The initiator limits dispatching priorities for all tasks in its address space. Associated with each job class processed by this initiator are two limiting value: the upper limit, and the uniform dispatching priority. Either of these value—or both—may be omitted. The **upper limit** is the maximum value (between 0 and 13) which d_1 can take; if omitted, d_1 can take any value up to 13. The **uniform dispatching priority** applies to all jobs in this class processed by this initiator; it is a constant value for d_1 which overrides any DPRTY values furnished by users.

Automatic priority group (APG)

OS IV/F4 normally chooses dispatching priorities for all user tasks according to their recent histories of using the CPU and various I/O devices. This facility is called **APG** and is based on the following approach. OS IV/F4 automatically monitors usage of the CPU and all tape and disk channels. Periodically (every few seconds), the dispatching priorities of all tasks operating under APG are changed; tasks which have recently used the CPU heavily are given temporarily lower dispatching priorities for the next time period, while tasks which have not heavily used the CPU are given temporarily higher dispatching prioroties. Hence, I/O-limited tasks have chronically higher priorities than compute-limited tasks, the latter gaining-or-losing dispatching priority in round-robin sequence.

Jobs not asigned to APG operate altogether higher or altogether lower than those in APG, and their dispatching priorities are not automatically modified by OS IV/F4. Most jobs will operate most efficiently if they are assigned to APG, whether they are I/O-limited, CPU-limited, or balanced. To select APG, the user omits DPRTY parameters from corresponding EXEC statement.

Table 2.1 Operator communication macro instructions

Macro Instruction	Full Name	Function
WTO	Write to operator	Program issues message on one or more consoles, as indicated by a console-distination card
WTOR	Write to operator with reply	Program issues message on one or more consoles, then awaits the operator's reply, which is read into a program buffer
QEDIT	Queue edit	Program accepts any MODIFY or STOP command previously submitted for this job by the operator
WTL	Write to log	Program writes a message onto the system log data set (SYS1.LOGREC)
DOM	Delete operator message	Program requests deletion of message currently displayed on operator CRT console, previously issued by this job step by a WTO or WTOR macro instruction

Operator Communications

During execution of his job, the user can issue messages to the console operator using the assembler-language macro instructions described in Table 2.1 (and corresponding verbs in higher-level languages, such as DISPLAY and ACCEPT in COBOL).

Initiating tasks by START commands

System tasks may optionally be started by operator commands, such as readers (START RDR) and initiators (START INIT). Procedures for these tasks are predefined in the system procedure library (SYS1.PROCLIB) and are similar in format to ordinary cataloged procedures.

Most user jobs are read from JES readers or submitted through internal readers. A user job can also be initiated by a START command, whose operand is the name of a cataloged procedure. This job comprises one job step, comparable to processing one EXEC statement (although the cataloged procedure can itself invoke several different programs). Installations can invoke frequently-used system utility programs in this way.

Although system and user tasks do not require initiator services when started this way, they each occupy one address space as usual. Differences between jobs submitted through JES readers and via START commands are summarized in Table 2.2.

2.6.3 Terminating Job Steps

When a job step terminates, OS IV/F4 regains control via its step terminator, which determines whether the ending was normal or abnormal (ABEND). Even if a task ends normally, the step

Table 2.2 Comparison of the task activated by START command and general jobs

Comparison Item / Type of Task	Input mode	Task protection key	Time monitoring during execution?	SMF records	Check point/ restart?	Maintenance processing of data sets	CANCEL command permitted?
System task	Procedure in SYS1.PROCLIB invoked by START command	0	No	Not collected	Impossible collected	Required except for readers, writers and VTAM	Impossible except during allocation
Job step initiated by a START Command	Procedure in SYS1.PROCLIB invoked by START command	≠ 0	Yes	Not collected	Impossible	Required	Possible
Ordinary job submitted through JES reader	Inputted from the input stream, and read-in by JES reader	≠ 0	Yes	Collected	Possible	Required	Possible

terminator processes its resources (main and secondary storage, data sets, etc.) carefully; there is little operational distinction between normal and abnormal termination processing .

A job may be abnormally terminated by a CANCEL command. Whenever a job terminates abnormally, the user may receive a formatted SYSOUT dump of his address space if he has previously requested this option.

At the termination of each job step, the initiator releases any resources acquired solely for this step and passes other resources (e.g., passed data sets) to subsequent job steps. The initiator determines whether succeeding steps should be executed based on return codes issued by current/previous steps and whether the current step terminated normally.

Normal end

In each compiler or assembler language, facilities exist for issuing a normal end ("RETURN," "EXIT," etc.) or abnormal end ("ABEND,"etc.). In all cases, when the job step task ends (as defined in Section 2.6.2) the corresponding job step is considered terminated.

Abnormal end

As noted earlier, an abnormal end (ABEND) is differentiated from a normal ending by OS IV/F4 primarily in terms of specialized diagnostic processing for the former. Typical ABEND causes are the following conditions:

- Unrecoverable hardware error.
- Erroneous usage of a system macro instruction.
- Erroneous data-set usage.
- Exceeding a job or job-step limitation, such as CPU time, SYSOUT quantity, or protracted WAIT state.
- Job cancelled by console operator.

ABEND can be automatically issued by OS IV/F4 or requested by the user program.

For obtaining a partial/complete printout of his address space (**dump**), the user should furnish either a SYSABEND or SYSUDUMP DD statement at the time he submits his job; these statements should be included with each step possibly susceptible to ABEND. A SYSUDUMP statement instructs OS IV/F4 to print the contents of the user's virtual storage plus relevant OS IV/F4 tables and register save areas. A SYSABEND statement requests the same information plus a formatted display of the OS IV/F4 nucleus.

An assembler-language user can regain control after ABEND occurs with a specialized exit routine. In some cases, the user program can correct errors causing ABENDs and resume normal execution.

Execution limits

Each installation can limit four resource utilizations by user jobs. In addition, each user job can optionally specify more stringent limitations on these resources:

- CPU time for the entire job.
- CPU time per step.
- Time in uninterrupted WAIT state.
- SYSOUT quantity.

The JOB-statement TIME parameter limits total CPU time for the job. The EXEC-statement TIME parameter limits CPU time for this step (including all executions within a cataloged procedure), subject to the overall job limitation on CPU time.

If a job remains in uninterrupted WAIT state for several minutes, it is likely to be entirely dormant or erroneously handled; for example, it may have missed a necessary I/O interruption, or it may be awaiting mounting of a tape reel. OS IV/F4 will automatically cancel such a job, according to an installation-specified limit for uninterrupted WAIT state.

The quantity of printed/punched records directed to a SYSOUT device may become excessive during a job, possibly through a programming error. The user can prespecify a limitation for a SYSOUT data set by an OUTLIM parameter on the corresponding DD statement. Overriding this is the aggregate SYSOUT limitation specified by the installation for the corresponding initiator.

At the end of each job step (and also at the end of the job), OS IV/F4 passes control to an SMF routine. Each installation can add its own exit routines to SMF in order to capture statistics on system performance and resources used by particular jobs.

Releasing resources

At the end of each job step, most of its resources are reclaimed by OS IV/F4, so that they can be issued to other users (or to subsequent steps of the same job). Passed data sets are retained by this initiator for subsequent steps. If a step terminates abnormally, OS IV/F4 reclaims all resources from the user, whether these resources are released in an orderly fashion or not.

2.6.4 Conditional Execution of Job Steps

Normally, steps of a job are processed in their input sequence one by one. However, if a job step has an unsatisfactory termination (either in the OS IV/F4 ABEND sense or due to bad input data), it can set a return code to prevent needless execution of subsequent steps. For example, if a compile-load-go job encounters a serious source-program error during the compilation step, it should generally bypass the "load and go" step.

Return codes

Each job step issues a return code — explicitly or

implicitly — when it terminates. The **return code** is an integer between 0 and 4095, inclusive. By convention, OS IV/F4 compilers, assemblers, utility programs, and application packages issue return codes of 0 for normal ending and larger values for various severities of errors. Subsequent job steps can test these return codes to determine whether they should be executed or skipped, using COND parameters on corresponding EXEC statements:
COND=((condition-code-1,operator-1),(condition-code-2,operator-2), . . .)

Each **condition code** is an integer between 0 and 4095, which is compared against return codes from one or more previously-executed steps. **Operators** are the six relational operators: EQ, NE, GT, LT, GE, and LE, which correspond to "equal,""not equal,""greater than," etc. Up to eight tests of the above form — "(condition-code, operator)" — can be included in one COND parameter, if any of the tests is satisfied during execution of the job, the step bearing this COND parameter is skipped by OS IV/F4.

COND parameter on a JOB statement
If the user furnishes a COND parameter on his JOB statement, he thereby requests that step-skipping tests be applied to the second and all subsequent steps, using return codes passed from preceding steps. Whenever one of the tests is satisfied, all remaining steps in this job are skipped.

COND parameter on an EXEC statement
The user can furnish a COND test in either/both of the following formats:

(condition-code,operator)	Test condition code against return codes from all prior steps
(condition-code, operator, step-name)	Test condition code against the return code from this particular prior step.

COND parameter with an abnormal termination
A job step sets a return code only when it terminates normally. In general, if a step terminates abnormally, all subsequent steps are automatically skipped by OS IV/F4. However, two special forms of the COND parameter permit specific job steps to be executed subsequent to an abnormally-terminating step:

COND=EVEN	This step is executed even if a prior step has abnormally terminated.
COND=ONLY	This step is executed only if a prior step has abnormally terminated.

For example, STEP1 of a job updates records in a data set. If STEP1 abnormally terminates, the user may want to execute STEP2, which will print the data set. He should specify that STEP2 should be executed only if STEP1 abnormally terminates by coding ONLY in the COND parameter on the EXEC statement for STEP2.

Specifying return code tests
In the COND parameter, the user may specify tests to determine if the system should bypass a job step. If OS IV/F4 determines that a comparison is true, the job step is skipped (if COND was coded on the EXEC statement) or all remaining job steps are skipped (if COND was coded on the JOB statement).

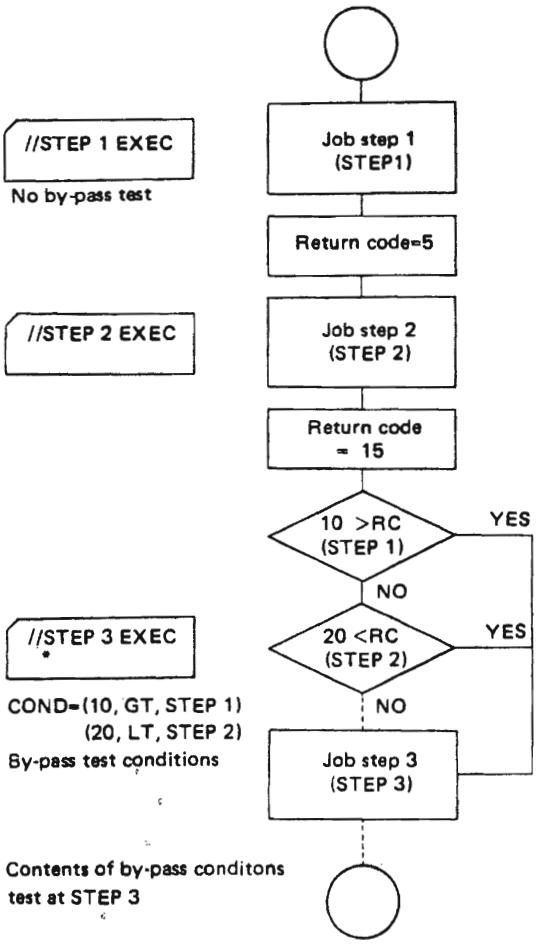
For example, if the user codes COND=((10,GT),(20,LT)) he is asking, "Is 10 greater than the return code, or is 20 less than the return code?

If the return code is 12, neither test is satisfied: no job step is skipped. All tests the user specifies must be false if processing is to continue without skipping any job steps. If the return code is 25, the first test is still false, but the second test is satisfied: 20 is less than 25. OS IV/F4 will bypass one job step or all remaining job steps, depending on whether the COND parameter was coded on the EXEC statement or on the JOB statement.

Example of routing a job through the system
The purpose of the following job is to execute five steps to perform an unspecified function. Not all of the steps will execute because of conditions are placed on them. See Fig. 2.21.

```
//ROUTE  JOB   (D58706),ROEGER,MSGLEVEL=(1,1),
//             CLASS=E
//STEP 1 EXEC  PGM=JSEBR 14
//DD1    DD    SYSOUT=A
//STEP 2 EXEC  PGM=JSEBR14,COND=EVEN
//DD2    DD    SYSOUT=A
//STEP 3 EXEC  PGM=JSEBR14,COND=ONLY
//DD 3   DD    SYSOUT=A
//STEP 4 EXEC  PGM=ABEND806
//DD4    DD    SYOUT=A
//STEP5  EXEC  PGM=JSEBR14,COND=ONLY
//DD5    DD    SYSOUT=A
```

1. This job will use the installation-defined priority default.
2. The JOB statement specifies that only JCL statements and messages are to be written, and that the job is assigned to job class E.
3. All SYSOUT data sets will be directed to output class A.
4. STEP1 will execute normally.
5. STEP2 will execute normally.
6. STEP3 will not execute.
7. STEP4 will execute and will abnormally terminate. (ABEND806 program issues an ABEND macro instruction).
8. STEP5 will execute because a preceding step did abnormally terminate.



Parameter description	(10, GT, STEP 1)	(20, LT, STEP 2)
Significance	10 > Return code of STEP 1	20 < Return code of STEP 2
Test satisfied?	Yes because 10 > 5	No because 20 < 15
By-pass conditions judgement	Either of conditions is satisfied? .... Yes	

When COND parameter in EXEC statement is designated, the execution of the job step is based on the completion status (return codes) of the preceding job steps.

Fig. 2.21 Example of multiple condition codes

2.6.5 Terminating Jobs

When the last step of a job has completed, or when remaining steps have been skipped by OS IV/F4, the job terminator routine gains control and releases all resources held by the job. All SYSOUT data sets created by the job are entered onto the JES queue, ready for transcription to printers, card punches, and — via RES — remote terminals as soon as the latter are readied and available.

2.7 SYSTEM OUTPUT

The principal outputs from each OS IV/F4 batch job are (a) permanent data sets on disk and tape, (b) punched card decks, (c) printed listings, and/or (d) magnetic-tape images of printed/punched outputs. Outputs of type (a) were discussed in Section 2.5.7. The current section describes outputs of types (b) - (d), which are collectively called SYSOUT data sets. They contain a combination of messages issued by OS IV/F4 — to record job initiation, step initiation, device allocation, volume mounting, step termination, job termination and device disposition events

— and outputs generated by compilers, assemblers, editors, utility programs, and application programs.

All SYSOUT data sets are processed by JES writers. JES queues output data according to data set characteristics such as output class, forms, print train, and forms control buffer name. Data sets are also queued by installation writer name and destination, as discussed in the "external writer" section. These characteristics are obtained from the SYSOUT DD statement or the demand OUTPUT statement. With the exception of held data sets and spinoff data sets, output for jobs, started tasks, or time-sharing users with identical characteristics are queued together in a data set group pointed to by a **job output element** (JOE). Each held and spinoff data set is queued separately and constitutes a "group" of one data set. Each data set group is considered a processing entity with a set of processing characteristics. JES selects work by data set groups and will — if the separator option is specified — delimit each group with separator pages or cards. See Fig. 2.22.

```
SYSOUT = (A, 2 PRT), UCS = PN
SYSOUT = A, UCS = PN              Total of four JOEs
SYSOUT = B                        built
SYSOUT = A
-------------------------------------------
SYSOUT = A
SYSOUT = A
    .                             Total of three
    .                             JOEs built
    .
SYSOUT = B
SYSOUT = B
    .
    .
    .
SYSOUT = (A, ,2 PRT)
```

Fig. 2.22 Relationship of SYSOUT specification to number of job output elements

JOEs built for job-related output are duplicated according to the number of job copies requested by the REPEAT parameter of the WRITER statement. This allows the number of copies being processed for any job to be governed by which devices are available for output.

2.7.1 Types of SYSOUT Data

By coding various JCL parameters the user can request output data sets, listings of JCL statements, system messages, and abnormal termination dumps. By coding various JCL and JES statements, he can request special forms processing, routing of output to specific devices, and multiple copies of certain data sets within a job. The JES statements have several options which complement those of JCL statements, with some additional features such as multiple destinations, left and right indexing features for printers, and data set grouping.

This section includes three topics:

- JCL statements and system messages
- Abnormal-termination dumps
- Output data sets

JCL statements and system messages

OS IV/F4 displays messages about a job concerning allocation of units and volumes, disposition of data sets, and termination of job steps and the job. The user can request that these messages — and/or JCL statements from the job and from procedures called by the job — be included on his output listing.

By coding the MSGLEVEL parameter on the JOB statement, the user informs the system what statements and messages he wants included on his output listing. The notation used on output listings to identify cataloged and in-stream procedure statements is described in the chapter "using cataloged and in-stream procedures."

By coding the MSGCLASS parameter on his JOB statement, the user assigns messages and JCL statements to an output class. A default class is assigned if MSGCLASS is not coded.

Abnormal-termination dumps.

To obtain an address-space dump in the event of abnormal termination of a job step, the user codes a DD statement defining a dump data set. The name of the DD statement must be either SYSABEND or SYSUDUMP. If both are present, the last occurrence will be used.

The SYSABEND or SYSUDUMP DD statement can provide a dump displaying the processing program's virtual storage area, the system nucleus, the entire system queue area, all local system queue areas, and any active link pack area (LPA) modules for the failing task. Descriptions of dumps and information on reading dumps are included in the **OS IV/F4 Guide to Debugging.**

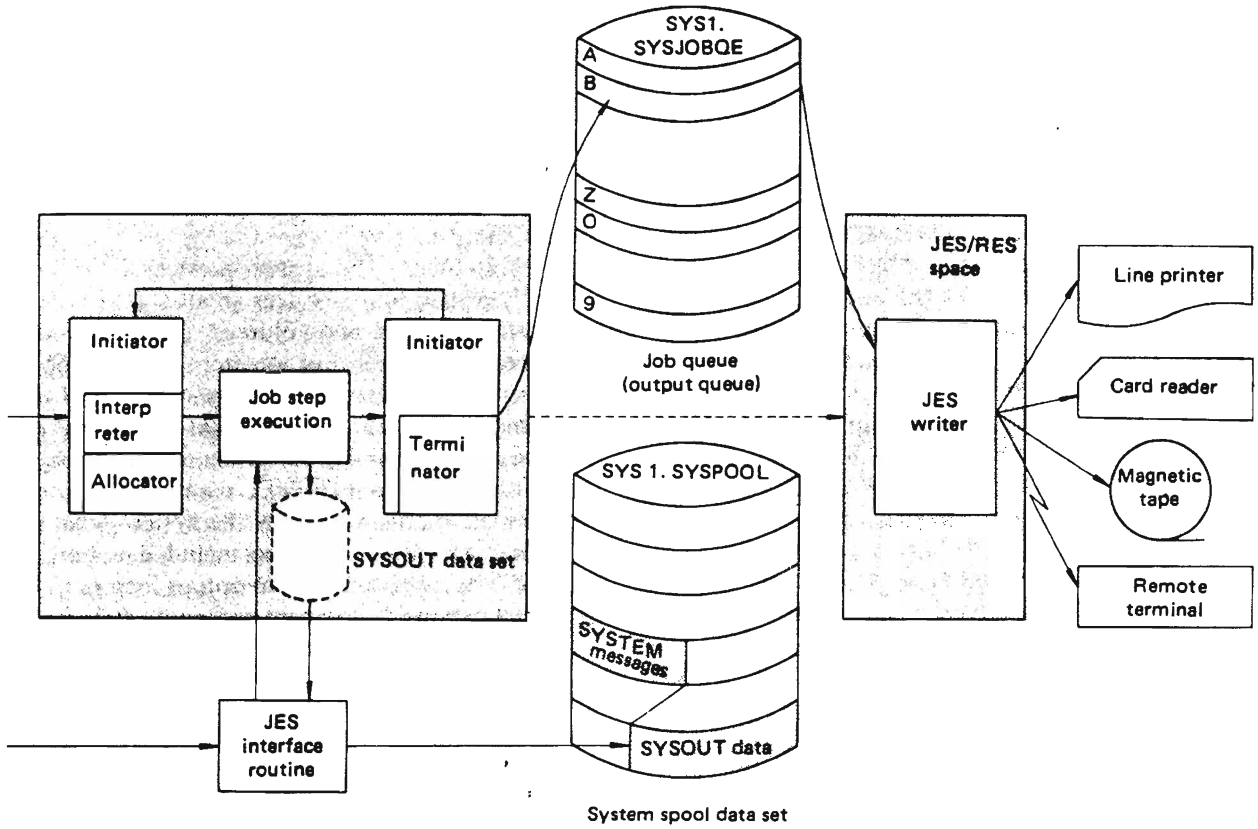
To have the dump printed, the user either assigns the dump to an output class with the SYSOUT parameter of his DD statement or codes the UNIT parameter and specifies the printer he wants. To store the dump, he defines the data set as he would any other data set, coding DSNAME, DISP, UNIT, and VOLUME parameters. If the data set should go to a direct access device, he must code a SPACE parameter.

If a private data set is specified and more than one dump is possible, the data set should be specified with a disposition of MOD, as it will be closed after each dump.

Output data sets

There are two ways to write output data sets:

- Assign the data set to an output class.
- Specify the device onto which the output should



Contents of SYSOUT data set prepared at a job step are actually written-in to the SYS1.SYSPOOL by the help of JES spooling function.

Fig. 2.23 SYSOUT Data set and spooling

be written.

When the user assigns aa data set to an output class, it is handled by JES. The data set is first written to the JES spool volume and then written to the final output device by either JES or an external writer. If the user specifies the device in his UNIT parameter, when the device becomes available, it is exclusively assigned to his job, under the control of his program.

Assigning data sets to SYSOUT classes

Output classes contain output with similar characteristics written to the same device. Thirty-six possible output classes can be coded on either the SYSOUT or MSGCLASS parameters. The letter and number names have no inherent meaning — each installation defines its own output classes. For example, output class W might contain low priority output; class X might be reserved for high-volume output. If the user wants output data sets and messages from his job to be printed on the same output listing, he should specify SYSOUT=*, which implies the same output class in the SYSOUT parameter as for his MSGCLASS output.

Each installation can designate certain SYSOUT data sets as reserved. **Reserved classes** contain data sets which should be held — deferred from JES output processing. If an output class specified by a MSGCLASS parameter is not designated as reserved, it will not be held. Data sets can be explicitly held by coding either the HOLD=YES JCL parameter or the HOLD parameter on the ALLOCATE and FREE TSS commands. (Refer to the **FACOM OS IV/F4 TSS Command Language Reference Manual,** for information on TSS commands.) Jobs can be released from hold state by the console operator or by a time-sharing user with an OUTPUT TSS command. By using reserved classes, holding or releasing desired print data sets is controlled by MSGCLASS parameters on JOB statements. See Fig. 2.23.

Device specification

To write an output data set without using the JES SYSOUT service, the user should code a UNIT parameter on his DD statement defining the device onto which the data set is to be written. OS IV/F4 will allocate the device exclusively to his job if the device is available: no other job can write output to that device until it is released. Jobs cannot share a non-SYSOUT device assigned via a UNIT parameter, as they can when output is assigned to

SYSOUT classes.

Data management routines write program output to the device specified in the UNIT parameter. Specifying particular output devices with UNIT parameters is not recommended for most OS IV/F4 installations.

Processing SYSOUT classes

JES is a highly efficient facility for writing outputs. JES supports local and remote printers, punches, and magnetic tapes as devices onto which output classes may be written. An external writer supports tapes and DASDs with installation-defined writer routines.

Job-related output is neither held nor spun off nor processed by a user-written writer. A **spun-off data** set is made available for output processing before job termination. Job-related output is retained until the end of a job, then printed by JES. All dynamically deallocated SYSOUT data sets are spun off and, as with held output, are not considered part of job-related output.

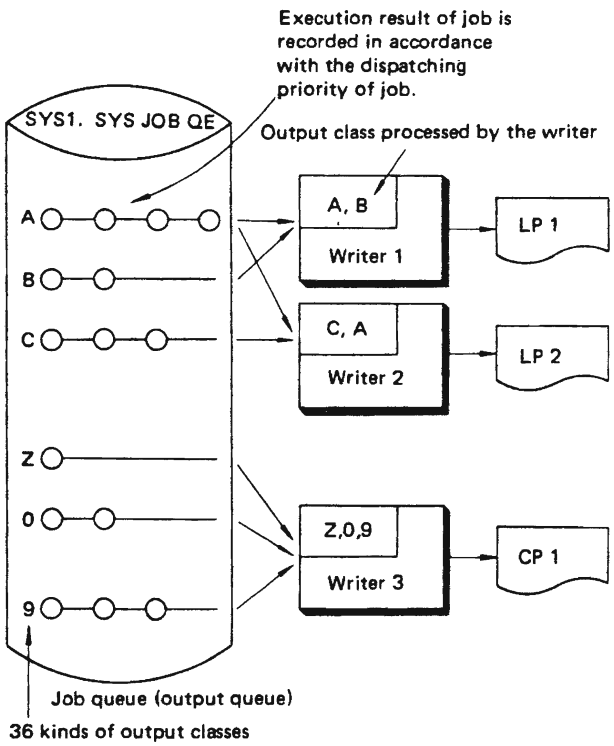


Fig. 2.24 Output class and writer

Outputs are printed on a single listing if such parameters as CLASS, FORMS, FCB, UCS, and DEST have similar characteristics for these data sets and a user-written writer is not specified. An installation may choose to direct all data sets specifying the same output class as the MSGCLASS parameter to the same listing, even though their FORMS, UCS, FCB and/or DEST parameters may differ.

For an external writer, the console operator determines what data sets are selected. He can direct certain outputs to the same listing, even though their FORMS, DEST, UCS, and FCB parameters are not identical. See Fig. 2.24.

Either an OS IV/F4 external writer or an installation writer can process output. An external writer must be started by the console operator, as described in the **FACOM OS IV/F4 System Programmer's Guide.**

Limiting output records

The number of logical records in a SYSOUT data set can be limited by specifying the maximum number of records in an OUTLIM parameter. For example, a program which is printing may go into an endless loop. The user can anticipate this problem and limit the number of records printed before OS IV/F4 discontinues output processing.

Requesting multiple copies of a SYSOUT data set

The user can control the number of hard copies produced from his SYSOUT data sets. As many as 255 copies of an output data set can be requested by a COPIES parameter on the SYSOUT DD statement defining the data set or on the JES OUTPUT control statement.

Requesting forms and print chain control

When requesting that an output data set be printed, the user can give JES special instructions on how to print the data set.

- A special output form.
- A special character set, when output is being printed by an F650D printer with the universal charactor set (UCS) feature.
- A special image which controls how many lines per inch are printed and the length of the form, when the data set is written to an F650D printer with the forms control buffer (FCB) feature.
- A specific carriage-control tape, when the data set is written to a printer without the FCB feature.

Requesting a special output form

Special forms may be requested by including the form name in the SYSOUT parameter of the DD statement defining the data set or on the OUTPUT control statement. For example, the user can assign a data set to a printed output class and specify that is be printed on a special form. JES and the external writer insure that the proper form is mounted.

Requesting a special character set

A universal character set (UCS) feature is requested by coding the UCS parameter on a DD statement defining an output or SYSOUT data set or by coding UCS on the OUTPUT control statement for a SYSOUT data set. The user can request UCS

features for different sets of characters to be printed for various applications.

Chain identifier for a F650 D Printer	Characteristics of chain/train
AN	Arrangement A, standard EBCDIC character set, 48 characters
HN	Arrangement H, EBCDIC character set for FORTRAN and COBOL, 48 characters ASCII character set.
PCAN	Preferred alphameric character set, arrangement A
PCHN	Preferred alphameric character set, arrangement H
PN	PL/I alphameric character set
QN	PL/I preferred alphameric character set for scientific applications
RN	Preferred character set for commercial applications of FORTRAN and COBOL
SN	Preferred character set for text printing
TN	Character set for text printing, 120 characters
XN	High-speed alphameric character set for a printer
YN	High-speed preferred alphameric character set for a printer

Fig. 2.25 Standard character sets

To request a special character set for a printer, the user should specify the code identifying the character set in the UCS parameter or the OUTPUT statement. Codes of the FACOM standard character sets are summarized in Fig. 2.25. Some of these character sets may not be available at a particular OS IV/F4 installation. In addition, an installation can design character sets to meet special needs and assign unique codes to them. Hence, the user should consult his system programming staff for a complete list of available character sets for his installation.

The image of a character set must be stored in the **system image library** (SYS1.IMAGELIB) prior to its usage on a UCS-equipped printer. Each installation defines standard images corresponding to some/all print trains defined in Fig. 2.25 and stores them in SYS1.IMAGELIB, so that individual users need be concerned only with non-standard images and print trains.

If the user wishes to print all letters of a particular alphabet in upper case, he can specify the FOLD option on corresponding SYSOUT statements; this option maps lower-case characters onto their upper-case counterparts, in order to speed up printing and reduce the need for changing printer trains.

The user can furnish a VERIFY parameter with his SYSOUT statements, which requests JES to display the character set mounted on the line printer just prior to printing his data sets. This enables the operator to verify that the print train and UCS image currently in place are correct for these data sets.

Requesting a forms control buffer

The F650D printer has a **forms control buffer** (FCB) which controls line feeding and skipping according to an image previously stored in the printer control unit. Each installation can create as many FCB images as it needs, which are stored in SYS1.IMAGELIB just like the UCS images. Standard FCB images are as follows:

- STD1 printing 6 lines/inch on forms which are 8.5 inches long; and
- STD2 printing 6 lines/inch on forms which are 11 inches long.

To select a particular image, the user codes the corresponding FCB identifier on his SYSOUT DD statement.

The user can request either or both of two verification options with his FCB parameter:

- Verify that the correct FCB image has been loaded by OS IV/F4.
- Verify that the forms are correctly positioned within the printer vis a vis the initial status of the FCB image.

Controlling JES writers

While a writer is printing or punching, the console operator can issue WRITER commands to control these activities. Hence, these commands cannot be issued by the programmer. The operator can request forward feeding of forms, logical backspacing of printing (JES rereads part/all of a SYSOUT data set), temporary suspension of output, repetition of an output, etc.

- Page feeding: with an FSP option on a WRITER command, the operator skips printing forward a specified number of logical pages; he can delete all remaining output for this data set if necessary.
- Logical backspacing: with a BSP option, the operator can repeat all/part of a SYSOUT data set, or even the entire job output.
- Modifying line spacing: with an LSP option, the operator can change spacing from the current value (single, double, or triple spacing) to any other of these three options. Alternatively, the operator can cancel default spacing and let preprogrammed values in the SYSOUT stream control this printer.
- Temporarily suspending output: with a HOLD option, the operator can temporarily suspend output from a job, which is then entered onto the hold queue for this JES writer. No output data are lost by suspending printing/punching in this way. The writer selects another job output ready for processing and commences processing at once.
- Repeating an output: with a REPEAT option, the operator can request additional copies of a particular data set or of the entire job output.

Modifying operations of a JES writer

Using a MODIFY command, the operator can exercise the following control over a writer:

- By specifying CLASS=a,b,c,. . . the operator redesignates which SYSOUT classes are to be processed and their priority order.
- By specifying PAUSE=FORMS, the operator requests a pause so that he can change output forms.
- By specifying PAUSE=DATASET, the operator requests the output device be stopped briefly after each data set has been processed by the writer.

2.7.2 Special SYSOUT Controls

The programmer can exert special controls over selected SYSOUT data sets, such as directing them to a hold queue, suppressing them altogether, providing his own writer, and routing them to specific destinations.

Delayed writing of SYSOUT data sets
Data sets can be delayed from normal printing — or delayed for inspection from a time-sharing terminal prior to actually printing on that terminal — if the user specifies reserved classes or codes a HOLD parameter. For example, an installation can delay printing a very large data set to prevent monopolizing a SYSOUT device until smaller data sets have been written. If a data set requires special forms that are not immediately available, it can be held until the console operator has retrieved and mounted those forms. When HOLD=YES is specified on a DD statement, the data set is placed on a hold queue until the operator releases it. The user should notify the operator (using the NOTIFY parameter for TSS or the MESSAGE statement for JES) when his data set is ready for processing, because no message will be sent to the operator. The data set can be released by the operator or time-sharing user.

Suppressing SYSOUT data sets
By coding a special SYSOUT or UNIT parameter, the user can suppress writing of a data set by defining it as a dummy data set. This is useful when testing a program; the user may not want data sets printed until he is sure they contain meaningful output. Suppressing a data set saves processing time.

When furnishing a SYSOUT parameter, the user should also code a DUMMY parameter to define a dummy data set. When DUMMY is coded, the SYSOUT parameter is ignored and the data set is not written.

The user can also suppress an output data set by specifying a particular installation-defined class which has been defined to delete data sets. This technique can be used by an installation to suppress the output from started tasks such as those initiated by START and MOUNT commands.

If a device to which the data sets will be written is specified by a UNIT parameter, the user can assign the data sets dummy status by coding DUMMY or by assigning a data set name of "NULLFILE." Parameters other than DUMMY, DSNAME = NULLFILE, and/or DCB are ignored; no units are assigned to these data sets. When the program requests that a dummy data set be "written," the request is recognized but no data is transmitted. Dummy data sets are available for access by the basic sequential access method (BSAM) or queued sequential access method (QSAM).

If any other access method is used, OS IV/F4 considers this to be erroneous and terminates the job abnormally.

Private writers
Instead of (or in addition to) a standard OS IV/F4 JES writer, an installation can furnish one or more private writers. They must be installed in the system link library (SYS1.LINKLIB) prior to usage. The user requests a private writer by naming it in the SYSOUT parameter of corresponding DD statements.

Routing job outputs
The default destination for each job output is its point of submission: a local printer/punch, if the job was submitted through a local JES reader; a remote print/punch, if the job was submitted via RES through a corresponding reader. However, the user can request that selected data sets be routed to different local/remote destinations by furnishing DEST parameters on corresponding DD statements. In addition, the console operator can issue ROUTE commands to direct outputs to local or remote devices.

Bypassing writers
As noted earlier, a user can omit standard or private writers altogether by specifying a UNIT parameter for a data set, omitting the SYSOUT parameter from the corresponding DD statement. This practice is not generally recommended, since it requires the requested device to be available at the (unpredictable) time when the job is executed; it bypasses the efficient device-queuing approach of the JES writers. Also, the job can execute only as fast as limiting I/O devices, which are quite likely to be printers and punches if the user writes output directly to them.

2.7.3 Demand Output

Most OS IV/F4 job requests are handled on a **closed shop** basis, where I/O-control staff received and submit jobs to the system, also managing all output devices and returning printed/punched outputs to the user somewhat after they have been generated. The **demand output** facility of OS IV/F4 permits users to operate output devices themselves or to request outputs on demand from the SYSOUT queues within JES.

Specifying demand outputs
To receive his job output, the user need furnish only a specialized command statement, needing no other JCL or command statements. He must have indicated with a /*OUTPUT JES statement that output is to be directed to the **demand output** class rather than a normal SYSOUT class.

Demand outputs are held on spool volumes until requested by users, who submit special-format identification cards into a **demand reader,** a special-purpose card reader not also serving as a JES reader. As each identification card is received by JES, it immediately releases all outputs being held for this user to a printer/card punch designated for demand outputs. As soon as these output devices complete any current tasks, they commence processing these just-requested outputs.

Temporary dumping of demand output
If many users at an installation create demand outputs, it is possible for the aggregate size of these outputs to saturate the spool volumes. Since this can be troublesome to system operations, the console operator can issue a command to temporarily dump demand outputs to task or disk. These outputs can be printed/punched by standard utility programs at the operator's convenience.

Requesting demand output
The objectives of a demand output facility are to minimize the workload of the I/O-control staff at an OS IV/F4 installation and hence to maximize the simplicity and speed with which users can retrieve their own outputs from the OS IV/F4 output queue. To facilitate this, it is usually desirable to designate one or more card readers as demand readers; these should be located external to the principal computer room, convenient to the users' work area. Likewise, one or more printers (and possibly card punches) should be in this same area, dedicated to demand outputs. If possible, a display console should also be available to users, so that they can inquire about and request their jobs without assistance from the installation's staff; this approach characterizes an **open shop.**

2.7.4 Writer Procedures

Each JES writer is started by a writer procedure previously cataloged into the system PROCLIB:

WTR Standard procedure utilizing card punches and line printers

WTRT Procedure utilizing magnetic tape for output.

2.8 CHECKPOINT/RESTART

The OS IV/F4 checkpoint/restart facility permits reexecution or resumption of a job which was either abnormally terminated due to an internal program or data error or interrupted by a system hardware/software failure. Additional details on using checkpoint/restart may be found in the **FACOM OS IV/F4 Checkpoint User's Guide.**

2.8.1 Overview

Checkpoint
Execution of a job, may terminate abnormally due to an unrecoverable hardware error, operator error, logical error in the program, or erroneous input data. If the job is short, it is simplest and least troublesome for the user to resubmit his job. However, if the job is long-running, the user may wish to take precautionary measures to avoid losing substantial processing time if his job is prematurely terminated. Also, the user may wish to avoid the effort and turnaround delay of resubmitting his job. Such precautionary measures are defined to OS IV/F4 as **checkpoints,** where the user copies important program and data-set status data to a tape or disk file. "Checkpoints" refer both to procedures for copying this data and also to the records containing this status data.

Checkpoints can be taken within or between job steps. The following sections define how to restart jobs under various conditions and user-selected options.

Restart function
A **restart** is an attempt by a user or console operator to reprocess or resume a partially processed job. If from the beginning of a job step, the attempt is called a **step restart;** if from a checkpoint within a step, it is called a **checkpoint restart.** In either case, restarts can be designated by the user — at the time he initially submits his job — as **automatic** (requiring no user intervention) or **deferred** (requiring the user to explicitly request resumption by submitting a special job).

Restart options
Checkpoint and restart facilities furnished by OS IV/F4 are closely related; the user can elect any of the following four options when he initially submits a longrunning job that requires checkpointing:

- automatic step restart (ASR)
- automatic checkpoint restart (ACR)
- deferred step restart (DSR)
- deferred checkpoint restart (DCR)

These are summarized in Table 2.3.

Table 2.3 Checkpoint and restart

Type of restart \ Characteristics of restart	Restart method	Restart point
Automatic step restart (ASR)	System schedules restart automatically when job step ends abnormally	Beginning of job step which ended abnormally.
Automatic checkpoint restart (ACR)		From last checkpoint in job step which ended abnormally.
Deferred step restart (DSR)	User must submit a restart job	Beginning of any step in the job
Deferred checkpoint restart (DCR)		From any checkpoint within any step of the job.

2.8.2 Checkpoint/Restart Processing

This section describes how jobs are restarted under the four options defined in Section 2.8.1.

Automatic restart
If a job specifying ASR or ACR is interrupted for any reason, t is automatically requeued for execution by JES

Automatic checkpoint/restart (ACR)
OS IV/F4 will utilize the last checkpoint of the job step that terminated abnormally to determine where and how to resume processing (middle of job step).

Automatic step restart (ASR)
OS IV/F4 will restart the job step which was interrupted from its beginning.

Conditions required for automatic restart
When a step ends abnormally, its return code (if any) must match the range of **restart return codes** allowed by the installation. Also, the user must have furnished a RD parameter in his original JOB statement specifying automatic restart. Finally, the operator must authorize restart of each interrupted job.

Operator actions
As OS IV/F4 resumes processing a checkpointed job, it issues a message to the console operator asking if the job should be restarted. This confirmation prevents an erroneous job from failing, resuming, and failing again in an endless cycle.

Also, if it is inconvenient to restart the job, the operator can defer it until later, since he has three options for each job requesting automatic restart:

YES	Restart at once
NO	Terminate the job abnormally
HOLD	Postpone restarting the job

Postponing automatic restart
When the restart, the job is reentered onto the OS IV/F4 hold queue. Later, the operator releases this job with a RELEASE command, just as if it had been entered with TYPRUN=HOLD on its JOB statement.

Deferred restart
If the user wishes to explicitly review and resume a job if it should fail to execute properly, he specifies "deferred restart" on his initial JOB statement. If the job fails, he can elect to delete it from the OS IV/F4 hold queue or to restart it with a specialized job bearing a RESTART parameter on its JOB statement.

Deferred checkpoint restart
The user can request restart from any checkpoint within any job step; he designates these two values with his RESTART parameter.

Deferred step restart
The user can request restart from the beginning of any step in the interrupted job.

Data sets during check point/restart
During restart from the middle of a step, OS IV/F4 automatically repositions any tape or disk data sets — tapes are spaced forward the correct numbers of files and blocks; disk SEEK addresses are appropriately set. Unit-record and paper-tape devices are not positioned by OS IV/F4, and the user must make his own provisions for reprocessing them.

Restart cannot reconstruct the original contents of DASD data sets which are updated in place. Hence, corresponding jobs can be restarted only if updating can be accurately resumed from the point of interruption.

2.8.3 Taking Checkpoints

To take a mid-step checkpoint, the user must write an assembler-language CHKPT macro instruction. Hence, it is quite cumbersome to take mid-step checkpoints in such higher level languages as Cobol

and PL/I. The user must also define which data set (on tape or DASD) is to receive checkpoint records.

CHKPT macro instruction
This macro instruction indicates which status data should be written into the checkpoint record and whether automatic restart is permitted. If a job uses two or more checkpoint data sets, the CHKPT macro instruction indicates which is to be used; it also gives each checkpoint record a **checkpoint identifier.**

Checkpointed data
The primary information needed to restart a job mid-step is an image of its virtual storage area, attributes of its data sets and corresponding DD statements, and various control-program parameters.

Checkpoint identifier
To each checkpoint corresponds a unique identifier; either OS IV/F4 or the user can assign checkpoint identifiers. As each checkpoint record is written by OS IV/F4, its identifier is displayed on the operator console. During a deferred checkpoint restart, the user can designate which checkpoint is to be used by its identifier.

Table 2.4 Restrictions on cehckpoint data sets

Attribute \ Type of Volume		DASD	Magnetic tape
Number of volumes		1	> 1
Data set organization		Sequential or partitioned	Sequential
Record format		Underfined length (U format)	Underfined length (U format)
Block length (bytes)	Min.	600	600
	Max.	Track length	32,760
Other restrictions		Secondary space allocations not allowed	The following formats are permitted: • 9-track or 7-track • Standard or non-standard label, or no labels

Checkpoint data set
Checkpoints may be written onto tape or disk, as indicated in Table 2.4. Additional records can be written into the same data set for other purposes. If the data set is partitioned, each checkpoint record becomes a separate member. If the data set is sequential, multiple checkpoint records can be written, possibly interspersed with other data.

Multiple checkpoint data sets
Within a job step, the user can define one or more checkpoint data sets. His CHKPT macro instructions indicate which data sets are to be used. Even if one data set becomes unreadable, the user can optionally use an alternate copy if he has created it. Identification of each checkpoint is displayed to the operator when it is written: job name, DD name associated with this checkpoint, device name, volume serial number, and the checkpoint identifier.

2.8.4 JCL statements for Restarting a Job

When a job requiring checkpoints is initially submitted, its JCL statements must contain an appropriate **restart definition** (RD) parameter. If the job must be restarted, the user furnishes an appropriate RESTART parameter for a deferred checkpoint restart, plus a DD statement whose name is "SYSCHK."

Restart definition
The RD parameter can be furnished on a JOB or EXEC statement, the former value overriding if RD is furnished on both. RD indicates whether automatic restart should be attempted and/or whether checkpoint records should be written initially by the executing program:

RD=R Take checkpoints and allow automatic restart
RD=NR Take checkpoints for deferred restart
RD=NC No checkpoints or restarts
RD=RNC No checkpoints, but automatic step restart is permitted.

These options are summarized in Table 2.5.

Table 2.5 RD parameter

RD =	If CHKPT macro instructions are issued		No CHKPT macro instructions are issued
	Checkpoints written?	Automatic Checkpoint restart?	Automatic step restart?
R	YES	YES	YES
NR	YES	NO	NO
NC	NO	NO	NO
RNC	NO	NO	YES

RESTART parameter
If a user restarts a job on a deferred basis, he must furnish a RESTART parameter on his JOB state-

ment. If restarting at the beginning of a step, its name is his RESTART value. If restarting in the middle of a step, the user must also furnish the checkpoint identifier.

SYSCHK DD statement
For a deferred checkpoint restart, the user must furnish a SYSCHK DD statement naming the data set containing the desired checkpoint. For automatic checkpoint restart, the SYSCHK statement is unnecessary since OS IV/F4 records which data set contains the last-written checkpoint record.

2.9 SYSTEM MANAGEMENT FACILITIES (SMF)

SMF has two primary functions: to collect and record nontrivial events from system start-up to system shut-down, and to permit each installation to set its own limitations on resources used by jobs and job steps. Data collected by SMF routines — whether furnished in OS IV/F4 or by the user — are called **SMF data**; corresponding routines for capturing these data are called **SMF exit routines.** Each installation can capture and process SMF data to meet its particular needs for resource management and usage accounting. Additional details on SMF are contained in the **FACOM OS IV/F4 System Programmer's Guide** and the **FACOM OS IV/F4 SMF Implementation Guide.**

2.9.1 Collecting SMF Data

Standard OS IV/F4 routines capture and record SMF data, independent of one another but in a uniform format. Each SMF record type is specialized as to function and point of creation, although SMF records fall into four broad categories:

- per-job usage of resources, which facilitates accounting and monitoring of individual users
- data set usage
- volume usage
- usage of mainframe resources, and incidence of system-wide events.

While generating an OS IV/F4 system, an installation specifies which types of SMF records are to be collected during routine system operations; the console operator can elect more or fewer SMF types to be collected when reloading OS IV/F4.

Each SMF record contains a header field containing the following information:

- record type (two-digit number)
- capture time (date, hours, minutes, and fractional seconds)
- system identifier
- system model identifier
- job name
- associated JES-Reader time and date.

Accounting records
Accounting records are records which show the user's name, utilized system resources, and end status of a job or job step, upon completion of each job or job step. Accounting records are of three types:

- Type-4 End-of-step SMF records
- Type-5 End-of-job SMF records
- Type-6 Output writer SMF records

Table 2.6 shows when each of the above records is collected and its contents.

Table 2.6 Accounting records

Type 4 (End of step)
• SMF record header • Step name • Step sequence number • Program name • Date and time of step start • Date and time of step end • Step priority • Step termination indicators • Step CPU time • Size of virtual memory used • List of I/O devices utilized • EXCP count for each device • Number of page-ins for step • Number of page-outs for step • Step completion code*
Type 5 (End of job)
• SMF record header • Date and time that the reader processed the job statement • Date and time of job start • Date and time of job end • Number of steps in job • Job class • Job priority • Program name • Number of accounting fields specified in JOB statement • Job CPU time • Output class • Job termination indicator • Job completion code**
Type 6 (End of SYSOUT class processing)
• SMF record header • Date and time of SYSOUT start • SYSOUT class • Number of logical records processed by writer • Form number

* At normal termination, this field contains the return code. In the case of abnormal termination, the code may be either the system ABEND or user ABEND code.

** Completion code of the last step of this job will be shown here.

Table 2.7 Data set activity records

Type 14 (End of INPUT or RDBACK (*1) date set processing)
• SMF record header • Creation date • Expiration date • Device type • Number of volumes • Volume serial numbers • Pertinent portions of system control blocks • Record format • Record length • EXCP count
Type 15 (End of OUTPUT, UPDAT, INOUT, or OUTIN (*1) data set processing)
• Same as record type 14
Type 17 (When user data set is scratched)
• SMF record header • Data set name • Number of volumes • Volume serial numbers
Type 18 (When data set is renamed)
• SMF record header • Old data set name • New data set name • Number of volumes • Volume serial numbers
Type 62 (At opening of VSAM component or cluster)
• SMF record header • VSAM catalog name • Volume serial number containing the catalog • VSAM component or cluster name • Number of online volumes containing the component or cluster • Volume serial number • Device type
Type 64 (When a VSAM component is closed, (*2) or another volume is switched to, or no more space available)
• SMF record header • State indicator (close, volume switch, or no space available) • Indicator of component being processed (data or index component) • VSAM catalog name • VSAM component name • Current RBA (Relative byte address) • Volume serial number • Device type • Channel and unit • DASD cylinder and track address • Number of records updated • Number of records retrieved • Number of records deleted • Number of unused control intervals • Number of split control intervals • Number of split control areas • EXCP count
Type 68 (When VSAM catalog entry is renamed (*3))
• SMF record header • VSAM catalog name • Old entry name • New entry name

(*1): These are data sets residing on a magnetic tape volume or direct access volume.

(*2): If a cluster is closed, one record is written for each component in the cluster.

(*3): A record is written for each VSAM cluster, component, data set, or catalog entry name change.

Data set activity Records

Data set activity records are written when data sets are closed, scratched, or renamed, and when VSAM components are opened, closed or renamed.

- Type 14 INPUT or RDBACK data set activity record
- Type 15 OUTPUT, UPDAT, INOUT, or OUTIN data set activity record
- Type 17 Scratch data set status record
- Type 18 Renamed data set status record
- Type 62 VSAM component opened status record
- Type 64 VSAM component closed status record
- Type 68 Renamed VSAM entry record

Table 2.7 shows when each of the above records is collected and its contents.

Volume records

Volume records describe the space available on direct access volumes, error statistics on magnetic tape volumes, and data spaces in a VSAM catalog. Volume records are of three types:

- Type 19 Direct access volume record
- Type 21 Error statistics for tape volume record
- Type 69 VSAM data space record

Table 2.8 shows when each of the above records is collected and its contents.

Table 2.8 Volume records

Type 19 (When volume on a DASD is dismounted, or when HALT or SWITCH command is processed)
• SMF record header* • Device type and address • Volume serial number • Owner identification of direct access volume • VTOC address • Number of DSCBs • Number of unused alternate tracks • Number of unallocated cylinders and tracks • Number of cylinders and tracks in the largest free extent • Number of unallocated extents
Type 21 (When data set on magnetic tape is closed or EOV processed)
• SMF record header* • Device type and address • Volume serial number • Number of SIOs (Start I/O) • Number of recoverable read errors • Number of recoverable write errors • Number of unrecoverable read errors • Number of noise blocks • Tape density

Type 69 (When defining, extending, or deleting VSAM data space)
• SMF record header • Device type and address • VSAM catalog name • Volume serial number • Number of unallocated cylinders and tracks • Number of unused extents • Number of cylinders and tracks in the largest contiguous unallocated area on the volume

* SMF record header does not include data ralated to the job.

System Usage Records

System usage records describe the overall system operation statistics. These records are of the seven types listed below.

- Type 0 IPL record
- Type 1 System statistics record
- Type 8 I/O device configuration record
- Type 9 VARY ONLINE record
- Type 10 Allocation successful record
- Type 11 VARY OFFLINE record
- Type 12 End of run record

Table 2.9 shows when each of the above records is collected and its contents.

Table 2.9 System usage records

Type 0 (During system initialization after IPL)
• SMF record header(* 1) • Size of real storage area • Size of virtual storage area • SMF options in effect(* 2)
Type 1 (At IPL and at termination of first job step following interval of elapsed time (* 3)
• SMF record header (* 1) • CPU waiting time • Total page-ins for the entire system during interval • Total page-outs for the entire system during interval
Type 8 (During system initialization after IPL)
• SMF record header(* 1) • Class, type, and address of each online I/O device
Type 9 (Upon execution of VARY ONLINE command)
• SMF record header(*1) • Identificaton of each device added to the system configuration
Type 10 (After a device is added to the configuration)
• SMF record header • Identification of I/O devices which have become available
Type 11 (After execution of VARY OFFLINE command)
• SMF record header(* 1) • Identification of I/O devices removed from the system configuration
Type 12 (When HALT or SWITCH command is executed)
• Same as record type 1(* 4)

(* 1): SMF record header does not include data related to the job.

(* 2): SMF options include the following types of data:
- Upper limit of the job's continuous waiting time.
- Designation of SMF data set.
- Designation for selection of SMF data records.
- Presence/absence of SMF exit request.

(* 3): The system statistics record is written at 10-minute intervals. After SMF initialization, the record contains statistics accumulated during the IPL process.

(* 4): This record contains statistics accumulated during the interval between the last Type 1 record written and execution of HALT or SWITCH command.

SMF data sets

Two SMF data sets exist at all times on permanently-resident volumes of an OS IV/F4 installation: SYS1.MANX and SYS1.MANY. The data sets need not be on the same volume, but they must have the same device type. When one of the data sets is filled during system operation, OS IV/F4 notifies the console operator that it is automatically switching to the other SMF data set. Hence, OS IV/F4 alternates between SYS1.MANX and SYS1.MANY. When the operator learns that one of these data sets is full, he should soon schedule a special batch job to dump it to an ordinary sequential data set on DASD or magnetic tape. The special OS IV/F4 utility used for dumping SMF data also resets the dumped extent to "available" status, so that the SMF routines can write again to it when the alternate data set becomes filled.

2.9.2 SMF Exit Routines

During system generation, each OS IV/F4 installation can furnish as many SMF exit routines as it wishes, using SMF exit linkages defined in the **FACOM OS IV/F4 SMF Implementation Guide.** These routines become part of the OS IV/F4 control program for that installation; hence, they should be thoroughly tested before production usage. These routines can complement standard OS IV/F4 facilities for tasks such as the following:

- Validate JCL statements, testing parameters for accuracy, assigning/verifying job classes and

priorities, and assigning default values for omitted JCL parameters.
- Modify maximum elapsed time or maximum CPU time allowed for each job step.
- Increase SYSOUT limits during step execution.
- Halt writing of certain SMF records.
- Write supplemental SMF records, as designated by each installation.
- Interrupt execution of a job or job step.
- Write installation-dependent console messages.
- Calculate and record SYSIN totals per job.
- Calculate and record SYSOUT totals per job.
- Calculate and record turnaround times.

To write supplemental SMF records, an installation should use SMFWTM macro instructions, using record types in the range 128–255.

SMF job management exits

The following nine exit points are available:
- input-stream validation.
- job start.
- JCL validation.
- job-step start.
- SYSOUT limit exceeded.
- CPU time limit exceeded.
- SMF-record validation.
- job end (execution completed).
- job purge (SYSOUT processing completed).

Table 2.10 shows schematically where these exits are located within OS IV/F4 Job Management.

Standard SMF exit parameters

Each SMF exit routine receives the following parameters when OS IV/F4 passes control to it:
- job name.
- date and time of job entry.
- system identification.
- intra-job communication field.
- inter-job communication field.
- step sequence number.
- SMF option flags.
- job priority.
- job class.
- number of SYSOUT records.

The intra- and inter-job communication fields are used to pass information among SMF exit routines; the **intra-job field** is initialized at the start of each job, while the **inter-job field** can be used to communicate parameters between successive jobs processed by this initiator.

Table 2.10 summarizes these nine SMF exit points.

Table 2.10 Characteristics of SMF exit points

Exit point	When SMF receives control	Passed parameters in addition to standard SMF exit parameters
JOB statement validation	Reader receives JOB statement	Image of JOB statement
Job start	Initiator has selected next job	Programmer name, priority, and JOB-statement accounting parameter
JCL validation	1. Prior to interpreting each JCL statement 2. After all JCL statements have been interpreted	Images of JCL statements
Job step start	During step initiation, just prior to I/O device allocation	Step name, programmer name, and EXEC statement accounting parameter
SYSOUT limit exceeded	From I/O supervisor if SYSOUT is exceeded	
Time limit exceeded	1. CPU limit for job exceeded 2. CPU limit for step exceeded 3. Uninterrupted WAIT limit exceeded	Time-limit code
SMF-record validation/change	Just prior to transcribing block of SMF records to DASD	Block of SMF records
Job step end	End of step	Programmer name, step return code, step CPU time, EXEC statement accounting parameter, and the Type-4 SMF record
Job end	End of job	Programmer name, job return code, job CPU time, JOB-statement accounting parameter, and Type-5 SMF record
Job purge	After JES has transcribed all SYSOUT for job, just prior to deleting job from JES tables	Programmer name, job CPU time, and JOB-statement accounting parameter.

SMF exit points during job processing
Table 2.10 summarizes SMF exit points available during job processing. Each installation can furnish its special processing routines at some/all of these points.

JOB statement validation exit
Whenever the JOB statement of a new job is read, JES yields control to this SMF exit routine (if furnished), which decides whether to accept the job. If accepted, JES enters the job onto its input queue; otherwise, the job is discarded, and the JES reader skips forward to the next JOB statement.

Job start exit
Whenever an OS IV/F4 job initiator selects a job for processing, it first yields control to this SMF exit routine, which either confirms the job's eligibility for processing or deletes it.

JCL validation exit
As the initiator retrieves each JCL statement from the spool data set, it yields control to this SMF exit routine, which validates its syntax, checks installation-dependent parameter values, and deletes the job if it fails to conform to certain limits. After the routine returns control to the initiator, the JCL statement is interpreted as usual.

Job step start exit
As each step is initiated, control passes to this SMF exit routine, which decides whether to execute or delete this step.

SYSOUT limit exceeded exit
If the default (or user-furnished) SYSOUT limit is exceeded, control passes to this SMF exit routine, which decides whether execution should continue or be terminated abnormally.

Time limit exceeded exit
According to which limit is exceeded, the SMF exit routine decides whether execution should continue or be terminated abnormally.

SMF-record validation/change exit
Whenever the standard SMF-output routine is ready to write a block into the current SMF data set, it yields control to this exit routine which validates and/or changes the SMF records according to special needs of the installation.

Job step end exit
At the end of each job step, the OS IV/F4 step terminator yields control to this SMF exit routine prior to writing the standard Type-4 SMF record. The routine decides whether this record should be written and whether execution should continue.

Job end
At the end of a job, the OS IV/F4 job terminator yields control to this SMF exit routine, which decides whether the corresponding Type-5 SMF record should be captured.

Job purge
After all SYSOUT classes have been processed for a job, JES yields control to this SMF exit routine, which compiles and writes any appropriate accounting information. Immediately after the routine returns control to JES, this job is purged from the system — all of its control blocks and data records deleted.

2.10 MULTIPLE CONSOLE SUPPORT (MCS)

OS IV/F4 furnishes support of multiple operator consoles. This **multiple console support** (MCS) provides the following facilities:

- Backup console service
  When one console fails, the operator can specify an alternate console to process messages destined for the original console. In this case, route codes of the two consoles are not merged, and the alternate console receives only messages with its route codes. However, if the alternate console is the only active OS IV/F4 console remaining on that processor, it receives all messages.
- Operator action messages
  For MCS consoles configured in conversational mode or in roll deleteable mode, action, messages remain on each screen until deleted by the program issuing them or manually by the console operator. If an MCS console was defined as an output-only console, its messages can be deleted from the screen by entering a system command from an associated MCS console.
- Screen-oriented displays on CRT consoles
  At system generation, each MCS CRT console screen can be divided into multiple screen areas for receiving displays in out-of-line or non-message screen areas. To make decisions such as "operator action" MCS consoles can also be designated as "output-only."
- Authority levels
  Each MCS console may be assigned authority levels to restrict which commands it will display. The MCS master console on each processor must be enabled for all system commands.
- Log facilities
  Each installation can furnish a hardcopy device for logging, or it can log messages into system log facilities on DASD.

Up to 32 consoles can be operated on each OS IV/F4 system. The **main console** is the primary

one for operator-OS IV/F4 dialogue; all others are considered to be **auxiliary consoles,** used to manage specialized functions such as reel mounting, pack mounting, printer forms and train changes, etc.

2.10.1 Operator Commands and Messages

Operator commands (or simply, **commands**) are entered by operators through MCS consoles; **operator messages** (simply, **messages**) are displayed on one or more consoles by OS IV/F4 or executing programs. Most commands can also be entered by operators or users on punched cards through JES readers.

2.10.2 Display Consoles

An OS IV/F4 alphanumeric display console comprises a display screen, keyboard and, optionally, a **selector pen** (SP) and **program function keys** (PFKs). The screen can display 24 lines of 80 characters, as shown in Fig. 2.26.

Table 2.11 Operator commands

Verb	Function
CANCEL	Delete indicated job from system queues and/or execution
CONTROL	Define functions for a display console
DISPLAY	Display current system status
DUMP	Dump virtual image of indicated job onto SYS1.DUMP
HALT	Halt all OS IV/F4 activities
HOLD	Hold specified job in queue, or specified job class
LOG	Write specified text into SYSLOG data set
MODIFY	Alter/halt processing of an initiator or output class
MONITOR	Display system status continuously
MSGRT	Define status display area for a console
MOUNT	Notify OS IV/F4 that a volume has been mounted
RELEASE	Release a held job (or class of jobs)
REPLY	Reply to computer-issued message
RESET	Reset attributes of a job class, output class, or job priority to the OS IV/F4 default values
SET	Set one or more basic OS IV/F4 parameters
START	Start a system task, JES reader, JES writer, etc.
STOP	Stop the indicated system task, JES reader, etc.
STOPMN	Stop MONITOR functions
SWAP	Permit swapping a volume to a different drive
SWITCH	Switch SMF data sets
UNLOAD	Notify OS IV/F4 to permit unloading of a volume
VARY	Logically connect/disconnect a CPU, memory module, channel, or device from OS IV/F4
WRITELOG	Move contents of SYSLOG data set to JES writer

Format of display screen (Fig. 2.26)

- Message output area
  This area contains operator messages, commands, responses to commands, etc. in their sequence of entry. Each line is prefixed by a symbol indicating its origin, as shown in Table 2.12.

Table 2.12 Message prefixes

Prefix	Significance
*	System message requiring operator action
@	User-program message requiring operator action
–	Informational message from OS IV/F4
+	Information message from user program
'	Message which has already been acknowledged

- Program function key display
  This area is discussed below under "Entering Commands."
- Designation line
  This line displays OS IV/F4 status information about this console.
- Input area
  This area displays commands or responses recently entered by the operator
- Alarm line
  This line displays alarm messages about display console errors.

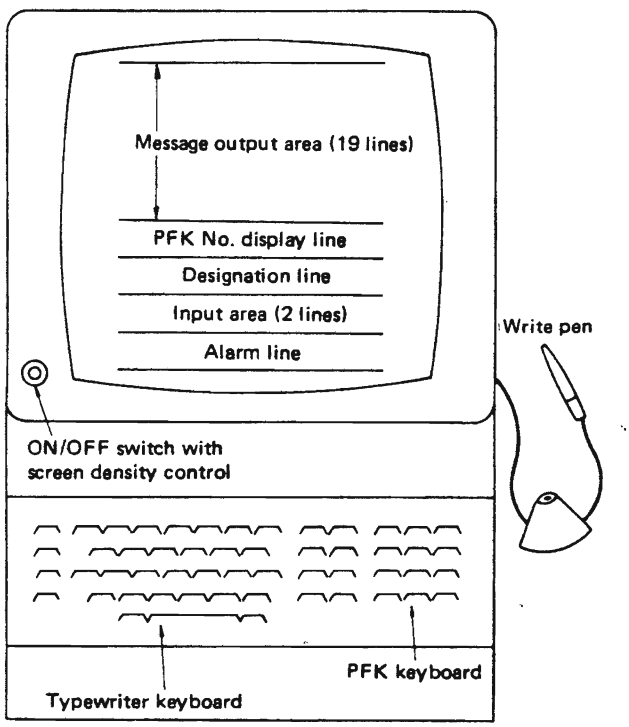


Fig. 2.26 Console display unit and standard format of its screen

Entering commands
In addition to keying in commands with the typewriter keyboard, the console operator can utilize the SP and/or PFKs.

- With PFKs
  Up to 12 PFKs can be used with an OS IV/F4 system, each corresponding to one or more commands. Each depression of a PFK enters a command in either conversational or non-conversational mode. In **conversational mode,** each command is displayed on a separate line corresponding to the particular PFK. In **non-conversational mode,** all PFK-entered commands are displayed in a single input area.
- With SP
  Instead of pressing a PFK, the operator can point to a PFK number displayed on the screen with the selector pen, which is sensed by the display screen and accepted as equivalent to depressing this key.

Status displays
OS IV/F4 provides the following information when requested by a DISPLAY or MONITOR command:

- status of I/O devices.
- status of jobs or system tasks.
- job queue status.
- consoles status.
- currently-executing jobs and system tasks.
- job name and its starting and ending times.
- allocations of data sets to DASD volumes.
- unallocated DASD space.

Defining the status display area
With a CONTROL command, the operator can define several display areas within the overall message output area, each to contain specific status information from the above list according to subsequent MSGRT commands. If multiple areas are not requested, all messages will be displayed consecutively. Table 2.13 shows how two status display areas may be defined.

Dynamic versus static displays
A **static display** is presented only when requested by the operator, a **dynamic display** is periodically updated by OS IV/F4, as requested by a MONITOR command. A STOPMN command halts dynamic display of console messages.

Control of the display screen
During system operation, the message output area gradually fills with commands and messages. Additional commands and messages can be entered only if earlier information is automatically or deliberately erased:

- Automatic erasure
  The oldest displayed information is automatically erased by OS IV/F4.
- Deliberate erasure
  The operator must delete obsolete messages and commands by indicating them with CONTROL commands or the selector pen. In either case, remaining messages move to the top of the screen, creating empty lines at the base of the message output area to receive subsequent messages.

2.10.3 Definition of Multiple Consoles

MCS comprises one main console and up to 31 auxiliary consoles. Specific functions are assigned to auxiliary consoles during system generation, and these functions can be changed at any time by operators with VARY commands.

Command groups
Operator commands fall into five functional groups, as shown in Table 2.13.

- Data displays and message replies
  These commands display the status of individual jobs or the entire system; they do not affect the operational system status.
- System control
  These commands manage OS IV/F4 jobs and queues.
- I/O control
  These commands change the logical status of I/O devices, channel paths, volumes, etc. They include the MOUNT, UNLOAD and VARY commands for volume management.
- Auxiliary consoles control
  These commands alter the configuration or functions of auxiliary consoles.
- Main console control
  These commands manage operation of the main console.

Only the main console can issue all five groups of commands. Any console can issue data display commands. The authority to issue system control, I/O

Table 2.13 Command groups

Command group	Commands		
Data displays and message replies	CONTROL MSGRT STOPMN	DISPLAY MONITOR	LOG REPLY
System control	CANCEL HOLD RESET STOP WRITER	DUMP MODIFY SET SWITCH	HALT RELEASE START WRITELOG
I/O control	MOUNT UNLOAD	SWAP	VARY (unit other than a console)
Auxiliary-consoles control	VARY-CONSOLE VARY-ALTCONS		
Main console	CONTROL-M VARY-MSTCONS		VARY-HARD COPY VARY-AUTH

control, and auxiliary-consoles control commands is restricted to certain auxiliary consoles (possibly none) during system generation. Auxiliary consoles can be allocated wider or narrower authority during system operation by VARY commands. Table 2.13 lists the command groups; Table 2.11 has previously described their functions.

Message destinations

OS IV/F4 and user programs display console messages by issuing write to operator (WTO) and write to operator with reply (WTOR) macro instructions (or higher level-language equivalents, such as the COBOL DISPLAY and ACCEPT verbs). An optional **destination code** can be included with each message to select to which console (or group of consoles) the message should be directed.

The correspondence between destination codes and consoles is defined for each installation during system generation. Each console can be assigned none, one, or more destination codes, so as to achieve a consistent function. Messages without destination codes or whose codes do not correspond to consoles are directed to the main console. Standard destination codes are defined in Table 2.14. A representative set of destination-console correspondences is shown in Fig. 2.27A.

Table 2.14 Standard OS IV/F4 destination codes

Code number	Function
1	Important messages for the main console, requiring operator action
2	Important informational messages for the main console
3	Messages about magnetic tape reels
4	Messages about DASD volumes
5	Messages about magnetic tape data sets
6	Messages about DASD/tape data sets
7	Messages about unit-record devices (card readers, punches, printers)
8	Messages about communications controllers, links, etc.
9	Messages regarding system or data security, e.g., passwords
10	Messages about hardware errors, maintenance actions, etc.
11	Write to programmer (WTP) messages, transmitted also to a SYSOUT data set
12	Emulation messages
13-15	Optional user destination codes
16	Unassigned code

Alternate consoles

To each console can be assigned an **alternate console**, which assumes the functions of the basic console if the latter suffers hardware or software trouble. Alternate consoles are assigned during system generation and can be changed during system operation by VARY commands. One console can serve as the alternate for several basic consoles, but each basic console can have only one alternate at a time.

In Example 1 of Fig. 2.28, the main console serves as the alternate for auxiliary consoles A, B, and C, and console A serves as the alternate for the main console. In Example 2 of Fig. 2.28, console A is the alternate for the main console; if both the main console and console A have problems, console B is the alternate for both. Example 3 of Fig. 2.28 shows an illegal configuration of alternate consoles, since the main console is defined with three alternates.

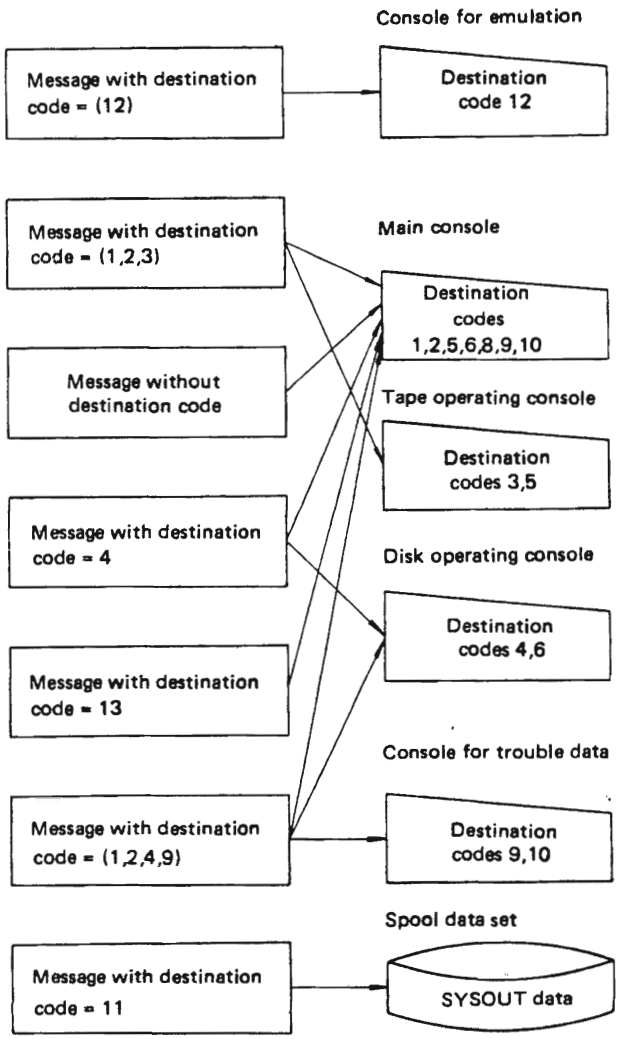


Fig. 2.27 Example of distributing messages to consoles according to their destination codes

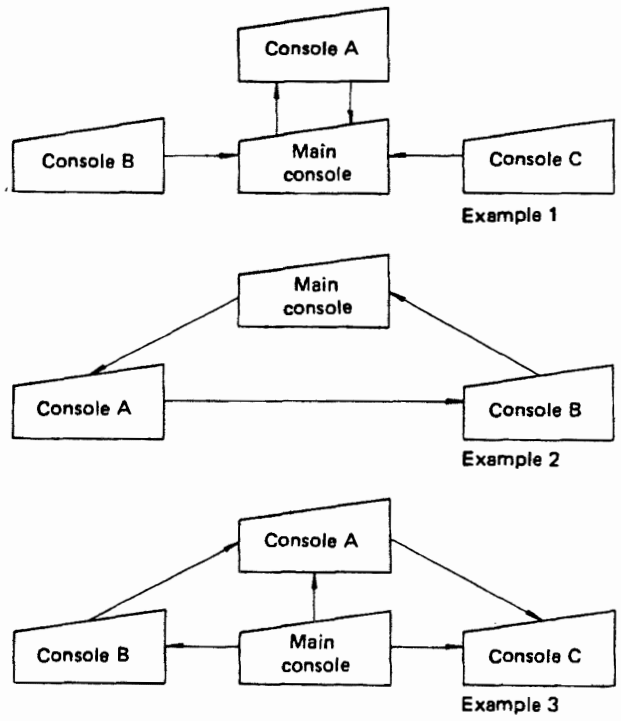


Fig. 2.28 Examples of alternate console configurations

Hardcopy log and system log

The **hardcopy log** records in a consecutive stream commands entered from all consoles, plus messages written to them. Messages and commands may be selectively logged, as defined during system generation and optionally modified during system operation by VARY commands.

The hardcopy log may be directed to a line printer or a character printer, as defined during system generation and optionally modified by VARY commands.

The **system log** is a consecutive record of system activity comprising two kinds of data:

- LOG commands issued by operators.
- Messages created by users with Write To Log (WTL) macro instructions.

If the system log is directed to a hardcopy device, all such information can be collected at one place.

The **system log data sets** are named SYS1.SYSVLOGX and SYS1.SYSVLOGY, suballocated from the spool data set (SYS1.SYSPOOL). During system operation, log records are written to one of these data sets until it is full; OS IV/F4 automatically switches to the other data set, and it also automatically transcribes the full data set to a SYSOUT queue defined at system generation. Hence, the system log is periodically printed out via a JES writer.

2.11 STARTING/STOPPING OS IV/F4 OPERATIONS

2.11.1 System Start

During initialization of OS IV/F4, the following operations must be performed, either automatically or with selected operator choices and parameters:

- load the OS IV/F4 control-program nucleus into main storage.
- initialize all related control tables and work areas.
- set date and time — operator entry.
- override selected operating parameters — operator entries.
- start system tasks associated with VTAM (Virtual Telecommunications Access Method) and NCP (Network Control Program), if the system operates communications equipment.
- start JES, and optionally RES.

Below are described four major activities during system start.

- initial program loading (IPL).
- nucleus initialization program (NIP).
- master scheduler initialization program (MSIP).

Initial Program Loading (IPL)

The IPL routine is stored on the SYSRES volume at a fixed location; it is brought automatically into main storage when the console operator presses the Load key on the control panel. The IPL routine loads the nucleus of the control program from SYSRES into lower main storage and NIP and MSIP into upper main storage, then yields control to NIP.

Nucleus Initialization Program (NIP)

NIP verifies whether I/O devices predefined at this installation are ready to operate. It accepts various parameter settings from the console operator and sets initial values into control tables. NIP opens several system data sets such as PARMLIB, SVCLIB, PROCLIB, and LINKLIB. NIP then formats the **external page data set** (EPS) and loads the link pack area (LPA) in virtual storage. NIP finally transfers control to MSIP.

Master scheduler initialization program (MSIP)

The OS IV/F4 master scheduler processes commands entered by the console operator relating to such system tasks as VTAM, TSS, AIM, and various JES readers, writers, and intiators. Some of these tasks are started automatically, others require explicit operator commands. MSIP sets the time of day based on operator entries, initializes JES and RES, the job queue data set (SYS1.SYSJOBQE), the system log (SYS1.SYSVLOGX and SYS1.SYSVLOGY), and the SMF routines and data sets (SYS1.MANX and SYS1.MANY). After completing all these functions, MSIP yields control to the OS IV/F4 Supervisor, which dispatches the high-

est priority task at the start of routine operations.

Collectively, the sequence of IPL, NIP, and MISP activities are called **system loading** or, loosely, IPL.

Time of day

NIP requests the operator to set/confirm the date and time at which the system is loaded, based on wallclock time. Once set, OS IV/F4 maintains an accurate digital record in main storage of the time and date, which can be interrogated by the TIME macro instruction (or equivalent higher-level language facility).

Automatic-start options

Many default system parameters can be set during system generation, helping minimize the number and complexity of operator entries during IPL. Collections of related parameters may be optionally stored as members in the **system parameter library** (SYS1.PARMLIB). As each member is retrieved during NIP and MSIP, the operator can intervene to override selected parameters to meet various contingencies. In this way, initial system loading (and reloading after a planned/unexpected stop) can be performed quickly yet flexibly, setting the following parameters and options:

- JES configuration—numbers of readers, writers, corresponding I/O devices.
- structure and format of spool data sets.
- initialization/not of the job queue data set.
- automatic START commands.
- premounted DASD volumes: which (if any) should be kept permanently mounted, etc.
- PROCLIB format.
- size of systems queue area (SQA).
- subroutines to be kept in the fixed link pack area (FLPA).
- various SMF options.

Automatic processing of commands

Once an OS IV/F4 system has been loaded, the operator can enter commands either through a console or on punched cards through a JES reader. When the system is initially loaded, he can request that sequences of commands be presented automatically to MSIP.

2.11.2 Stopping an OS IV/F4 System

During a typical day, the console operators may need to stop all system operations one or more times to make major equipment changes (e.g., take peripheral devices offline for preventive maintenance), change personnel shifts, perform inspections, etc. To perform this in an orderly loss-free manner, the operator issues STOP commands to readers, writers, and initiators. When the system has thus been quiesced, the operator issues a HALT EOD command to stop OS IV/F4 altogether, closing SMF data sets and transcribing remaining log entries to the system log device.

2.11.3 System Start and Restart

Initialization of OS IV/F4 and JES/RES are distinct processes. Basically, this means that those processes associated with initialization (coldstart or warmstart) are specified separately for OS IV/F4 (reloading LPA, etc.) and for JES (initializing the job queues). Although during any given IPL it is possible to coldstart OS IV/F4 or JES and warmstart the other, the usual procedure is to warmstart both. A detailed description of the process and options for IPL generally — including starting and stopping JES — may be found in the **FACOM OS IV/F4 Operator's Guide.** The description given here is an overview with considerably less detail.

The objective of **warmstart** is to maintain on queues (a) all jobs which have been previously entered through JES readers, (b) all job outputs awaiting processing by JES writers, and (c) other initialized system functions such as LPA.

If a serious hardware or software failure has occurred, the operator may be required to **coldstart** an OS IV/F4 system, which deletes all enqueued jobs and job outputs from the spool volumes and reformats both the spool data sets and the system job queue

Table 2.15 Job salvage possibilities during system warmstart

Status of job when OS IV/F4 halted	Warmstart feasible for this job?	Necessary actions
During JES reading	No	Reenter entire job
Awaiting initiation	Yes	None
During job initiation	Sometimes	Job must be re-entered if new data sets have already been allocated, etc.
During execution (no checkpoint)	Sometimes	Resumption from beginning of last step permitted if JOB statement specified RD=R or RD=RNC
During execution (checkpoint taken)	Yes	Operator decides which checkpoint to utilize
During job termination	Sometimes	(Complex option)
Awaiting JES writer service	Yes	None
During JES writing	Yes	Output can be resumed from a convenient point; operator controls this by WRITER command

(SYS1.SYSJOBQE data set).

Procedures for warmstarts and coldstarts are quite similar; the operator is asked by MSIP whether warmstart or coldstart is to be performed. Table 2.15 shows the status of jobs and data sets when OS IV/F4 stops under various conditions, and whether the system can be warmstarted in each case.

2.12 JOB CONTROL STATEMENTS AND PROCEDURES

A user can write programs in one or more programming languages. OS IV/F4 will translate them into a universal machine language, so that the instructions can be executed and work performed. The **job control language** (JCL) directs OS IV/F4 in handling application programs. When submitting programs to OS IV/F4, the user provides JCL statements to define work to be done, methods to be used, and necessary resources. In addition he can obtain special input and output processing by including JES control statements for the job entry subsystem. A collection of related problem programs is submitted to the operating system as a job. A **job** is made up of one or more **job steps**, each of which is a unit of work associated with the overall application.

2.12.1 JCL statements

The OS IV/F4 job control language contains nine types of statements. The name and purpose of each statement are summarized in Fig. 2.29.

Name of statement	Purpose
// JOB (job)	marks the beginning of a job; assigns a name to the job.
// EXEC (execute)	marks the beginning of a job step; identifies the programs to be executed or the cataloged or in-stream procedure to be called; assigns a name to the step.
// DD (data definition)	identifies a data set and describes its attributes.
/* (or two characters designated by the user to indicate delimiter)	indicates the end of data placed in the input stream.
// (null)	marks the end of a job.
// PROC (procedure)	for cataloged procedures, assigns default values to parameters defined in the procedure; for in-stream procedures, marks the beginning of the procedure.
// PEND (procedure end)	marks the end of in-stream procedure.
//* (comment)	contains comments.
// (command)	enters system operator commands through the input stream.

Fig. 2.29 Job control statements

Every job requires exactly one **JOB statement** (to identify the job), one or more **EXEC statements** (to identify each job step), and various **DD statements** (to identify data sets used by the job). The **null statement** is optional; placing it within the job causes JCL statements up to the next JOB statement to be ignored. JES ignores null statements.

The **delimiter statement** indicates "end of data" in the input stream. Paired **PROC** and **PEND statements** define a set of JCL statements as an in-stream procedure. Each **command statement** contains one operator command submitted through the input stream, used primarily by console operators. **Comment statements** can be used to make programs readily understandable by other programmers.

In addition to identifying data sets, job steps, and a name for the job, the user can code parameters on JCL statements to request resources and services from OS IV/F4, which is responsible for managing all resources of the entire system. OS IV/F4 automatically performs many services in processing jobs; however, the user can influence the processing of a job by including JCL parameters. For example, JES automatically selects most jobs for execution, but the user can influence when the job is selected or delay its selection by coding parameters on the JOB statement.

- **JOB statement**
  By furnishing various parameters on his JOB statement, the user can provide accounting information for the installation's accounting routines, define execution characteristics, specify conditions for early termination of his job, request a specific class for job scheduler messages, hold a job for later execution, and limit the maximum amount of time the job can use the central processing unit. These are summarized in Table 2.16.
- **EXEC statement**
  Parameters on the EXEC statement define which program or cataloged procedure is to execute. They also provide job-step accounting information, specify conditions for bypassing or executing the step, assign a limit for CPU time used by the step, and pass information to a processing program such as the linkage editor.
- **DD statement**
  Parameters on the DD statement provide OS IV/F4 with such information as the name of a data set, names of volumes on which it resides, type of I/O device for this data set, format of its records, whether it is old or new, size of newly-created data sets, and which access method will be used to create or refer to the data set.
- **Delimiter statement**
  This terminates each system input data set (SYSIN data). "/*" is normally used, but a special delimiter can be defined by a DLM parameter on a SYSIN DD statement.

- **Null statement**
  This statement indicates the end of a job.
- **PROC statement**
  This statement marks the beginning of an in-stream procedure or cataloged procedure.
  Parameters of a PROC statement assign default values to symbolic parameters of the procedure.
- **PEND statement**
  This statement indicates the end of an in-stream procedure.
- **Comment statement**
  This statement furnishes arbitrary user comments to be displayed on the SYSOUT listing.
- **Command statement**
  Most operator commands can be punched onto cards and entered into the input stream, normally by the console operator.

Table 2.16 Format of JOB statement

```
//job name   JOB   accounting-parameters, programmer-name[,
//                 other-parameters ]
```

parameters	Other parameters of JOB statement purpose
ADDRSPC	Define whether pagable storage is to be allocated to this address space
CLASS	Job class
COND	Define conditions under which remaining job steps are skipped
MSGCLASS	Output class for system messages
MSGLEVEL	Contents detail for system messages
NOTIFY	Notify a TSS user when his job completes execution
PRTY	Selection priority
RD	Restart definition; conditions under which job may be restarted
REGION	Size of region/address space
RESTART	Define where job should be restarted
TIME	CPU time limit for entire job
TYPRUN	Define whether job should be held, scanned or executed normally

Table 2.17 Format of EXEC statement

```
//step-name  EXEC { PGM=program name
                  { PROC=procedure-name } [, other parameters]
                  { procedure name
```

Parameter	Parameters of EXEC statement purpose
ACCT	Accounting parameters for this step
ADDRSPC	Define whether pagable storage is to be used for this tep
COND	Define conditions under which this step is to be skipped
DPRTY	Dispatching priority for this step
PGM	Defines one program
PROC	Optional keyword for procedure to be executed
REGION	Size of region address space
TIME	CPU time limit for this step

Table 2.18 Format of DD statement

```
//dd-name   DD   [ * ]                [ , other parameters ]
                 [ DATA ]
                 [ DUMMY ]
```

Parameter	Parameters of DD statement purpose
	Data Set Identification
DSNAME (or DSN)	Data set name
	I/C path, volume, and unit specification
AFF	Channel affinity (obsolete parameter; ignored by OS IV/F4)
SEP	Channel separation (obsolete parameter; ignored by OS IV/F4)
UNIT	I/O device
VOLUME	Volume identification
	Data Set Attributes
DCB	Data control block attributes
DDNAME	Postpone data set definition to indicated DD statement
DISP	Status and disposition
DUMMY	Dummy data set
LABEL	Label and security attributes
	DASD space allocation
SPACE	Ordinary space request for new data set
SPLIT	Request for a split-cylinder allocation
SUBALLOC	Request for a suballocation of space
	System input data set
*	Following statements are SYSIN data set (without "//" in position 1 -2)
DATA	Following statements are a SYSIN data set (possibly with "//" in positions 1 -2)
DLM	Ending delimiter for this SYSIN data set is indicated pair of characters
	System output data set
COPIES	Number of output copies to be made of this data set
DEST	Destination of remote (RES) output
FCB	Designate special forms control buffer
HOLD	Hold this output on queue until released by operator
OUTLIM	Limit on number of output records
SYSOUT	Define class for system output data set
UCS	Designate special buffer for Universal Character Set feature
	VSAM data set
AMP	Attributes
	RES/TSS features
DYNAM	Define dynamic data set in TSS
TERM	Distinguish RES and TSS terminals

2.12.2 The JES Statements

The user can control the setup and return of a program by coding JES control statements and placing them in the input stream. Two JES statements can be used with JCL to direct execution of a job. Fig. 2.30 shows the name and purpose of each statement.

Name of statement		Purpose
/*	OUTPUT	Requests return of demand outputs to a user
/*	SETUP	Lists mountable volumes needed for the job

Fig. 2.30 JES control statements

2.12.3 Specifying Job Parameters with JCL Statements

To request processing of a job, the user submits JCL statements and any related input data to SO IV/F4 through an I/O device chosen by the operator. The input unit can be a card reader, a magnetic tape, a terminal, or a direct access device. The sequence of JCL statements and input data for jobs submitted through an input unit is called the **input stream.**

A JCL statement consists of one or more 80-byte records. Typical jobs are originally submitted to OS IV/F4 for execution in the form of 80-column punched cards. OS IV/F4 is able to distinguish a JCL statement from data included in the input stream. In columns 1 and 2 of all statements except the delimiter statement appear "//." The delimiter statement contains "/*" in columns 1 and 2; a comment statement contains "//*" in the first three columns.

A job can be simple or complicated; the user can submit a procedure in the input stream or call a cataloged procedure. Fig. 2.34 shows some examples of what jobs can look like. Although only one example shows the use of JES statements, these statements could have been placed with all the jobs.

2.12.4 Examples of JCL Statements

A complete set of JCL statements for a COBOL compile, link-edit, and execute job is shown in Fig. 2.31. Each collection of source statements in the input stream becomes one SYSIN data set if it is delimited by statements of the following format:

```
//SYSIN  DD  *     (or DATA)
  [Input statements]
/*
```

An arbitrary number of jobs may be read from a card reader, magnetic tape, or DASD device as a single input stream. A typical input stream is shown in Fig. 2.32. If the stream is read from a tape or DASD, the operator can request that only part of the stream be processed by naming the starting and/or ending jobs in his START command for the associated reader:

```
START  RDR,FIRSTJOB,LASTJOB
```

If FIRSTJOB is omitted, the JES reader processes jobs up to LASTJOB: if LASTJOB is omitted, the reader begins with FIRSTJOB and reads to the end-of-file mark in the input stream.

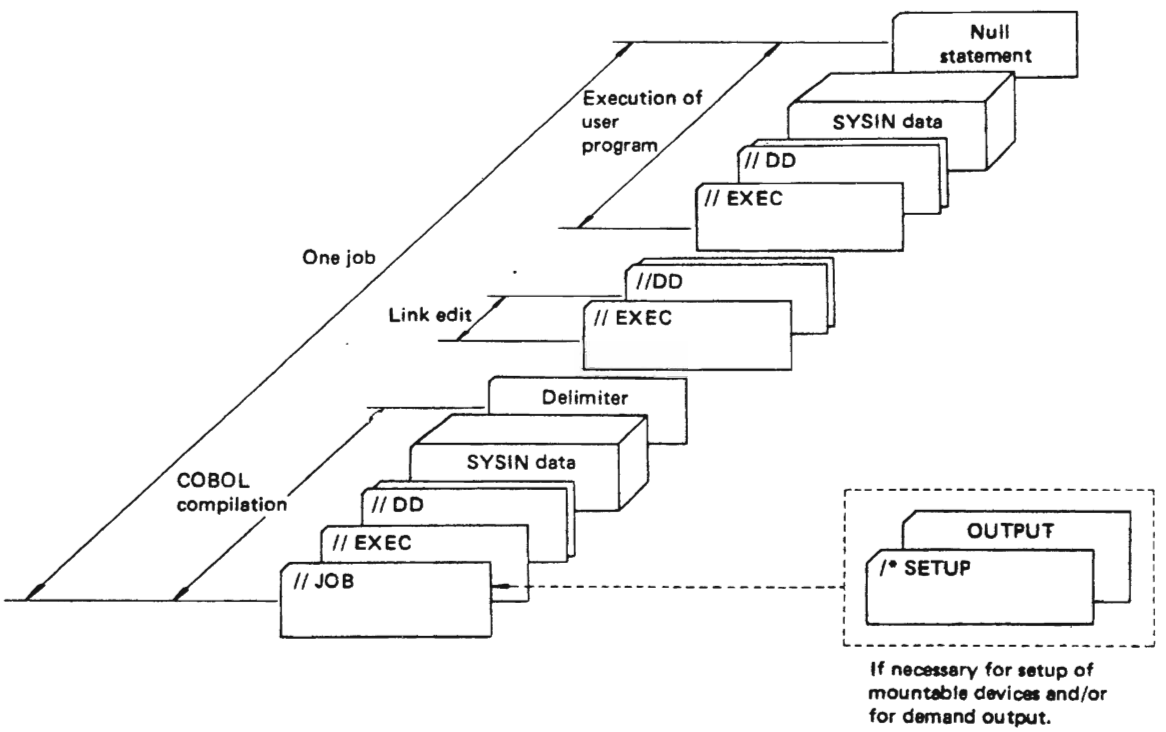


Fig. 2.31 Examples of job control statements

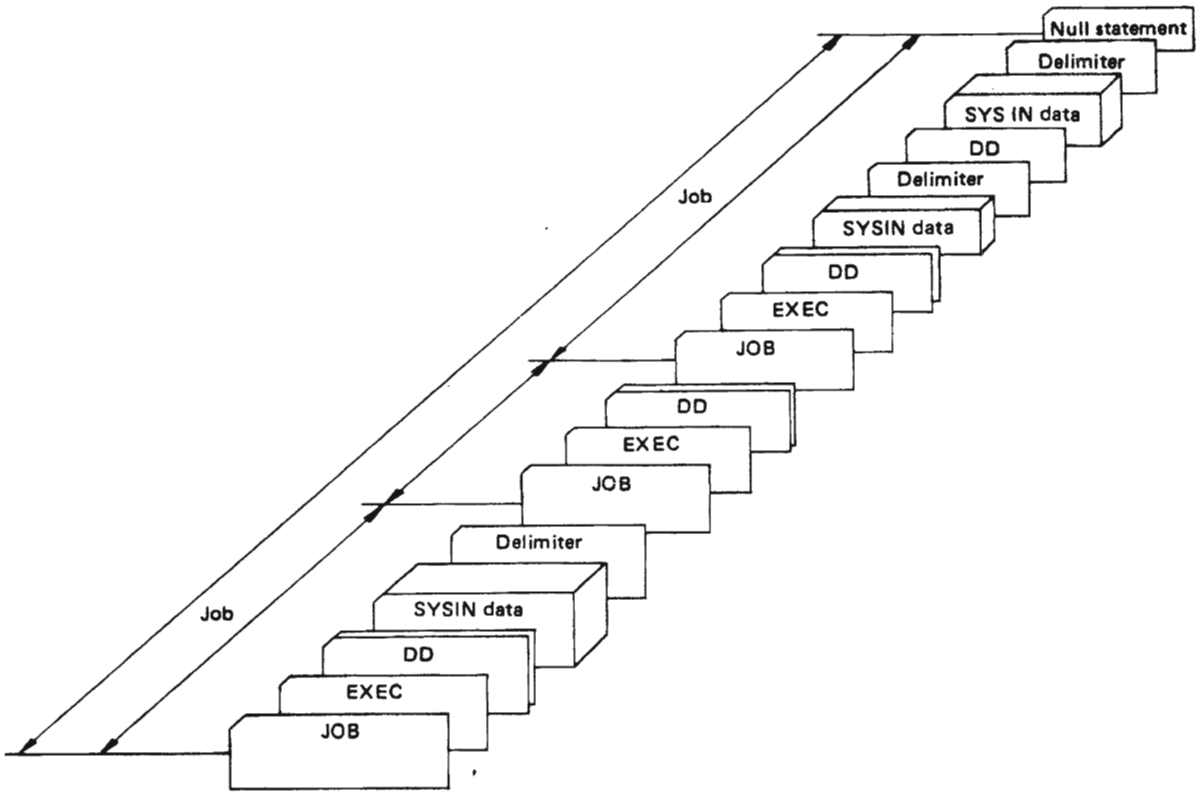


Fig. 2.32 Multiple jobs in one stream

2.12.5 Cataloged and In-Stream Procedures

Often the same set of JCL statements are used repeatedly with little or no change (for example, to specify compilation, link-editing, and execution of programs). To save programming time and to reduce the frequency of errors, each OS IV/F4 installation prepares standard job step definitions and places (**catalogs**) them into a partitioned data set known as the **system procedure library** (SYS1.PROCLIB). A set of JCL statements placed in the procedure library is called a **cataloged procedure.** A cataloged procedure comprises EXEC and DD statements, plus optional JCL comments statements.

By furnishing a JOB statement and one EXEC statement, the user can retrieve a specific cataloged procedure. The name of the procedure is specified on the EXEC statement. The effect is the same as if the JCL statements in the cataloged procedure appeared in the input stream in place of the EXEC statement which called the procedure. If necessary, the user can modify the cataloged procedure by overriding selected statements and/or parameters of these statements.

Before putting a procedure into the procedure library, the user may want to test it by converting it to an in-stream procedure. An **in-stream procedure** is a set of JCL statements that can be used repeatedly in EXEC statements of a particular job. A principal function of in-stream procedures is testing cataloged procedures prior to entering them into the procedure library.

In addition to standardized procedures for compilations, linkage editing, and loading, OS IV/F4 furnishes procedures for easy invocation of sort/merge programs, utility programs, and system tasks such as VTAM, TSS, JES readers and batch initiators.

Contents of procedures

Cataloged and in-stream procedures contain standardized job control statements needed to perform applications. A procedure contains one or more **procedure steps,** each step consisting of an EXEC statement that identifies the program to be executed and DD statements defining data sets to be used or produced by the program. The program requested on the EXEC statement must exist in a private library or the system link library (SYS1.LINKLIB). If the user requests a program from a private library, the procedure step calling that program must include a DD statement with the dd-name STEPLIB that defines the private library; the STEPLIB DD statement is described in Section 2.5.6.

Cataloged and in-stream procedures cannot contain:

- EXEC statements that refer to other cataloged or in-stream procedures;
- JOB, delimiter, or null statements;
- DD statements defining private libraries to be used throughout the job (DD statements with the dd-name JOBLIB);
- DD statements defining data in the input stream (statements including * or DATA parameters).
- JES control statements; they are ignored.

Identifying an in-stream procedure

To identify an in-stream procedure, the user must furnish PROC and PEND job control statements to delimit the body of the procedure. The PROC statement must be the first statement of an in-stream procedure; with it, the user assigns the procedure a name, used for subsequent calls to the procedure. Optionally, the user can also assign default values to symbolic parameters contained in the procedure. A **symbolic parameter** is a special identifier (symbol preceded by a single ampersand) representing a parameter, subparameter, or value in a procedure; including symbolic parameters in a procedure is described in **FACOM OS IV/F4 Job Management Functions and Facilities.**

If the user does not assign default values to one or more symbolic parameters on a PROC statement, these must be furnished by calls to this procedure. The simplest form of the PROC statement — for example, to identify an in-stream procedure named PAYROLL — would be as follows:

```
//PAYROLL  PROC
```

The PEND statement marks the end of an in-stream procedure. The user may include a name on the PEND statement and comments, but these fields are optional. Both of the following examples are acceptable:

```
//ENDPROC    PEND    end of in-stream pro-
                     cedure
//           PEND
```

The following example illustrates an in-stream procedure named SALES consisting of two procedure steps. Note that STEP2 includes a STEPLIB DD statement to define the private library in which the program JUGGLE can be found. Fig. 2.33 shows additional examples of in-stream procedures and their usage.

```
//SALES    PROC
//STEP 1   EXEC   PGM=FETCH
//DD1A     DD     DSNAME=RECORDS (BRANCHES),
//                DISP=OLD
//DD1B     DD     DSNAME=RECORDS (MORGUE),
//                DISP=MOD
//STEP 2   EXEC   PGM=JUGGLE
//STEPLIB  DD     DSNAME=PRIV. WORK, DISP=OLD
//DD2A     DD     SYSOUT=A
//         PEND
```

Placing a cataloged procedure in a procedure library

The major difference between cataloged and in-stream procedures is where they are placed; in-stream procedures are placed within the job that calls them. A procedure library is simply a partitioned data set containing cataloged procedures.

OS IV/F4 defines a standard procedure library named SYS1.PROCLIB, but each installation can define additional procedure libraries with different names. When a programmer calls a cataloged procedure, OS IV/F4 merges a copy of the procedure into his JCL statements; therefore, a cataloged procedure can be used by more than one programmer simultaneously.

To add a procedure to a procedure library, the installation uses the JSEUPDTE utility program. It can also use the JSEUPDTE utility to permanently modify an existing procedure. Before modifying an existing cataloged procedure, however, the system programmer should notify the operator, who must delay execution of jobs that might use the procedure library while it is being updated. Details on using the JSEUPDTE utility are included in the **FACOM OS IV/F4 System Utilities User's Guide.** Before placing or modifying a cataloged procedure in a procedure library, the submitter should test it without overriding any parameters, in order to ensure that procedure statements are syntactically correct. Additionally, he should test the procedure by running it as an in-stream procedure several different ways.

No special JCL statements are needed to identify a cataloged procedure. The PEND statement is never used, the PROC statement is optional. The user needs to code a PROC statement (first statement in a cataloged procedure) only when he wants to assign default values to symbolic parameters. The name of the PROC statement is not necessarily the name of the cataloged procedure; its submitter assigns the procedure a name when adding it to the procedure library.

Using cataloged and in-stream procedures

To use a cataloged or in-stream procedure, the user specifies the procedure name on an EXEC statement. He can modify the procedure by adding DD statements; overriding, adding, or nullifying parameters on EXEC and DD statements; and assigning values to symbolic parameters. Calling and modifying procedures is explained in greater detail in the following paragraphs.

Calling cataloged and in-stream procedures

To call a cataloged or in-stream procedure, the user names the procedure in the first operand of his EXEC statement, optionally preceded by the keyword PROC:

- Procedure-name.
- PROC= procedure-name.

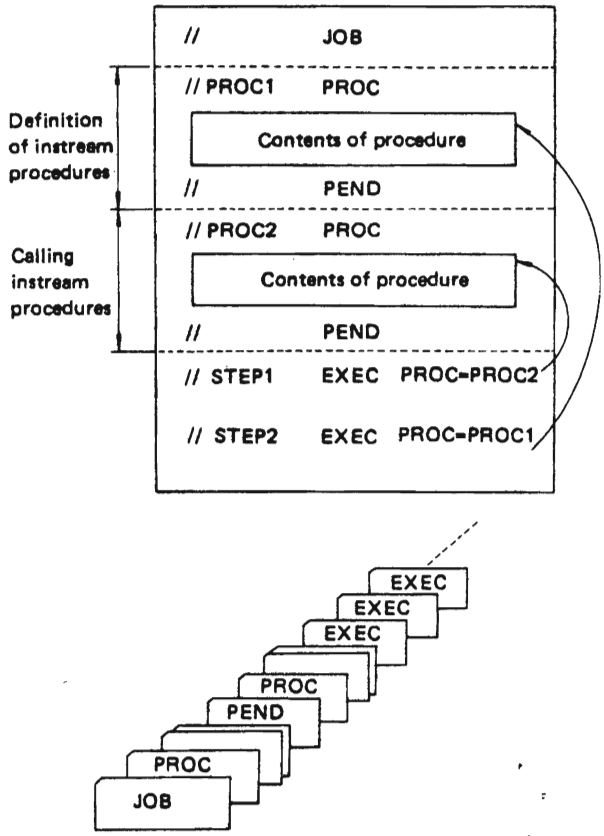


Fig. 2.33 Examples of in-stream procedures

A cataloged procedure must exist in the procedure library before usage. JES is responsible for fetching cataloged procedures, as described in Section 2.4.6. To call a cataloged procedure named PROCESSA, the user should furnish:

//CALL EXEC PROCESSA or
//CALL EXEC PROC=PROCESSA

When using an in-stream procedure the user includes the procedure — beginning with a PROC statement and ending with a PEND statement — with his JCL statements for the job; such procedures must follow the JOB statement and appear before EXEC statements that call them. The user can include as many as fifteen different in-stream procedures in one job. He can use each procedure as many times as he wishes in the job.

On the EXEC statement, he can also code JCL modifications required for this execution of the procedure.

Allowing for changes in cataloged and in-stream procedures

The usefulness of cataloged and in-stream procedures would be destroyed if each programmer were obliged to permanently modify them every time he wanted to make a change. When writing a procedure, the submitter can define as symbolic parameters those parameters, subparameters and values likely to vary each time the procedure is used.

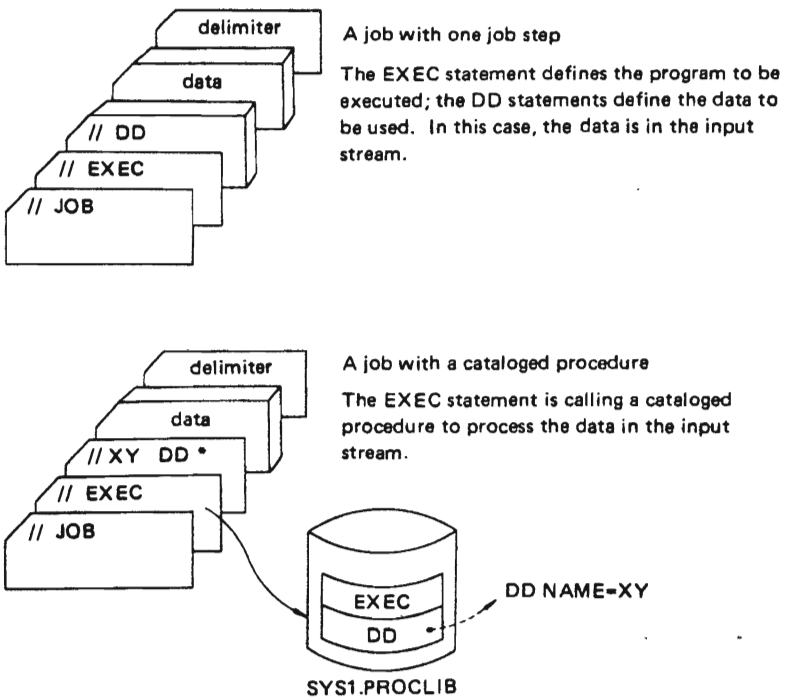


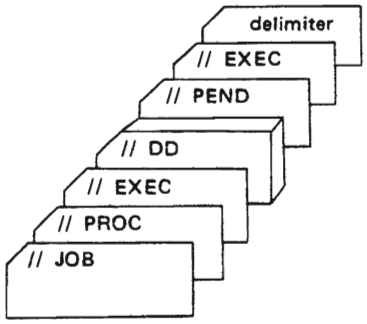
Fig. 2.34 Examples of jobs using procedures (part 1)

Modifying cataloged and in-stream procedures

The user can modify a procedure by:

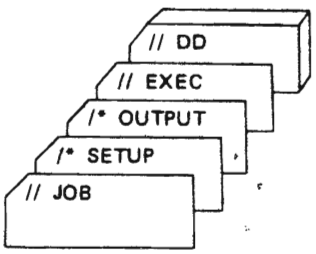
- Assigning values to (or nullifying) symbolic parameters contained in the procedure.
- Overriding, adding, or nullifying parameters on EXEC and DD statements in the procedure.
- Adding DD statements to the procedure.

Any changes he makes are in effect only during this single invocation of the procedure.



A job with an in-stream procedure

The EXEC statement refers to an in-stream procedure which is shown using the PROC and PEND statements.



A job with JES statements

A simple job using JES control statements. The command and any comment statements would be the only control statements to be placed in front of the JOB statement.

Fig. 2.34 Examples of jobs using procedures (part 2)

CHAPTER 3
REMOTE ENTRY SERVICES

3.1 Overview

3.1.1 Functions and Facilities

RES provides a facility for remote entry of batch jobs to OS IV/F4. It does not provide interactive processing, which is offered in OS IV/F4 by the time sharing system (TSS). RES users submit jobs from remote batch terminals, each typically comprising the following hardware components:

- controller (often based on a minicomputer).
- card reader.
- line printer.
- operator console.
- optionally, card punch, paper-tape equipment, etc.
- data communications interface.

Jobs can be prepared and submitted to RES just as to JES; their formats are usually no different for RES than for JES, except as described below. Jobs can optionally be entered remotely at one location and outputs returned to another remote location or to the central OS IV/F4 installation. Likewise, jobs can optionally be centrally entered via JES with their outputs routed to one or more remote terminals via RES. Hencee, JES and RES are functionally similar and can be used for complementary purposes.

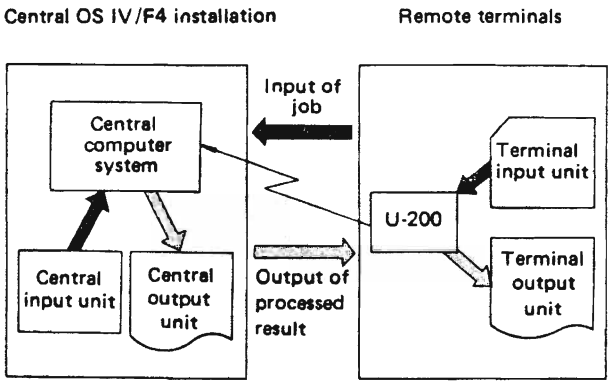


Fig. 3.1 Remote batch processing

3.1.2 Controlling Output Destinations

RES allows users to submit jobs to a central computing center from a work station and to route output to work stations.

The default output location is the submitting location, whether a remote work station or the central installation (destination of LOCAL). To receive the output at the submitting location, the user simply assigns output data sets to any output class (with the SYSOUT parameter) and messages from his job to an output class (with the MSGCLASS parameter). RES offers most of the options for writing data sets available to jobs whose outputs are handled at the central installation. The user can request that:

- a data set be held until the operator requests that it be printed.
- special output forms be used, by specifying the corresponding form name in his SYSOUT parameter.
- multiple copies of a data set be printed or punched.

Whether at a remote station or at the central installation, the user can also request that a data set be routed to another destination. To route an output data set to another destination, he codes the identification of that destination in the DEST parameter of his DD statement defining the data set. If he codes a destination on his SYSOUT statement, it overrides the default destination. Work stations are identified by destination identifications established by each installation. A destination parameter causes outputs to be routed to local printers or punches, or to any remote station.

3.1.3 Remote Entry of Jobs

With RES, the user can submit jobs and receive system output at remote facilities as if the jobs had been submitted at the central installation. The remote facility must be connected to the central computer by a (point-to-point) binary synchronous communication link. The remote facility becomes a

logical extension of the local computer facility and is operated by a person called a **remote operator.**

There are two types of RES stations: a **remote terminal,** which does not have a CPU, and a **remote workstation** that does have a CPU. For example, FACOM U-200 workstation can be used for entering jobs into and receiving data from RES. A processor—for example, another FACOM computer—executes a RES-generated program that allows it to send and receive jobs from a larger computer operating under OS IV/F4 RES. A **terminal station system program** (TSSP) is established by a special RES support program, RMTGEN, during system generation. Each workstation furnishes printers, punches, card readers, and/or a console. A **remote station** is a collective term for a remote terminal or a remote workstation.

Reading, printing, and punching between a CPU and a remote terminal (i.e., without a CPU) take place one action at a time. For example, the terminal can either transmit print data or transmit punch data or read an input stream, but it cannot perform two or more of these actions simultaneously. The remote operator selects the sequence of these actions. How this is done is presented later in this section under, "Altering the Sequence of Operations from a Remote Terminal."

Communication between the local CPU and remote workstations uses a RES facility called **interleaved transmission** that allows multiple print and punch streams to be transmitted concurrent with multiple console messages and input streams received centrally by RES. Using interleaved transmission, an OS IV/F4 installation can maintain several operations simultaneously. Operators at remote terminals and at workstations without RES consoles can enter operator commands into the input stream in the normal manner, i.e., via their card readers. OS IV/F4 schedules replies to these commands back to the requesting remote stations for printing on corresponding remote printers.

Remote lines can be configured as dedicated or nondedicated. The overall configuration is defined at system generation and optionally modified during system start-up when the remote stations are specified. If the station parameter RMTnnn designates a line number, the corresponding line is **dedicated** to that station. Lines not selected by station parameters at initialization are **nondedicated lines,** eligible to be dynamically connected to any nondedicated station.

Remote stations not physically connected to a CPU — stations that must be connected via dial facilities — normally do not specify dedicated lines, so that they may access any available nondedicted line. There are other reasons for specifying a line as nondedicted, even if the line is physically connected to a remote station:

- A LOGON card is not required for connecting stations to dedicated lines. If furnished, a LOGON card is ignored, since stations on dedicated lines are considered active when their lines are started. Line and station password authorization is only enforced for nondedicted lines and stations.
- One physically-connected station can be initialized as multiple nondedicted stations for use by different groups at different times. Usage of each logical station is defined at LOGON and LOGOFF times. Data routed to a logical station will only be transmitted while that logical station is logged on.
- If remote stations are initialized as nondedicated, one remote station can be used as backup for an inoperable station by being logged on with the identification of the inoperable station.
- A station attached to a dedicated line is considered active whenever its line is active. Line activation is under control of the central operator, who is not aware of station usage on dedicated lines. Via his console he is aware of station usage when nondedicated stations log on and off. JES allocates resources for remote lines while they are active, which is only between logon and logoff for nondedicated lines.

One advantage of specifying dedicated lines is that corresponding stations need not log onto RES, a manual process at all remote terminals.

It is possible to configure lines and stations utilizing dial facilities as "dedicated." However, only one station identification and set of station characteristics can be associated with each dedicated line.

Starting remote sessions

Since communications lines are considered inactive by JES immediately after system start-up, each line must be activated using a RES START command, either by the operator, with commands entered into a job stream (for example, through the JES initialization deck) or through the automatic command processor.

A nondedicted remote station must first submit a LOGON statement in the following format:

Column	Description
1	/*LOGON
16	REMOTEnnn
25	Password-1
73	password-2

- REMOTEnnn defines the remote station requesting logon. The numbers must be left-justified with no leading zeroes.
- Password-1 defines the password established at initialization or changed by the operator for that line. If the line has a password, then password-1 is required. To establish password, the installation sets the LINEnnn RES initialization parameter. This password can be subsequently changed or displayed by the operator.
- Password-2 is established at initialization, one password assigned to each terminal. If the ter-

minal has a password, then password-2 is required. To establish password-2, the installation sets the RMTnnn RES initialization parameter. The password ensures that the station logging on is a valid station.

A line is dynamically allocated when activated. The operator can deactivate and deallocate a line using the STOP command of JES/RES.

A remote device is considered active when its remote station becomes active, provided that the automatic start option has been specified for the device by the START subparameter in a Rnnn.RDm, Rnnn,PRm, or Rnnn.PUm initialization parameter. Otherwise, the device is considered inactive and must be started either by a remote or local operator command. The interval in which a remote terminal is active is called a **session.**

Altering the sequence of operations from a remote terminal

Two RES options allow the remote terminal operator to control the sequence of operations at his terminal.

During RES generation, an installation can specify a **delay time** ($WAITIME parameter) between printing and punching outputs for each job. This delay permits the operator to ready the card reader and prepare the terminal to transmit data. RES will sense this condition and read the input stream before resuming printing or punching.

When each printer or punch device is defined at RES generation/initialization, (Rnnn.PRm or Rnnn.PUm parameters), the **suspend mode** of operation can be specified or omitted. If the suspend mode is in effect, a remote operator can alter the sequence of operations by stopping an output device. When he again readies the device, RES will simulate an output suspension by flushing its current I/O buffers and printing any defined remote separator page. RES will then determine if the remote card reader is ready; if so, its input stream will be read. If not, RES selects the highest-priority output, which can be either resumption of the suspended operation or printing/punching another data set. The delay must be sufficiently long for the terminal to notify RES of the stopped device state. The delay interval depends on the terminal type.

If suspend mode is not in effect, the current operation is resumed after the device is readied again.

Options for disconnecting remote lines

At RES initialization, each installation uses the LINEnn statement to choose whether each line is to have the **abortive disconnect** feature. If selected, RES automatically disconnects the line by simulating a $E command sequence when the communications control program (CCP) detects a not-ready data set.

If abortive disconnect is not selected, the line will remain active and wait for the data set to be readied or for operator-intervention conditions under which a CCP can detect a not-ready data set depend on line configurations.

An installation can also instruct RES to automatically disconnect an inactive station by coding a non-zero value into the DISCINTV parameter of RMTnnn during RES initialization. When this disconnect interval has elapsed without any data being sent or received on the line, RES will disconnect the line by simulating a $E command sequence.

Job flow

All remotely-entered jobs are transferred from RES to JES, where their source images are stored on spool data sets and their queue entries on the system job queue (SYS1.SYSJOBQE). Thereafter, these jobs are processed by OS IV/F4 identically as those entered locally, in particular with respect to job classes and priorities. The central operator — and also the remote operator — can observe and control progress of the jobs. Upon termination of each job, its outputs are written into spool data sets using up to 36 SYSOUT classes, just like local outputs. To each active terminal corresponds a RES writer, which returns designated outputs to the terminal according to the priority of its SYSOUT classes.

Remote operator messages

Remote operators can send messages to the central installation or to other terminals by issuing SEND commands; the SEND facility is also available to central operators. By issuing a LISTBC (list broadcast messages) command, a central operator or a remote operator can display the contents of the system broadcast data set.

Stopping remote sessions

The remote operator issues a LOGOFF command to terminate a session. Thereafter, RES will neither accept jobs from this terminal nor attempt to transmit outputs to it. Fig. 3.2 displays the overall flow of jobs and commands in RES.

3.1.4 System Configuration

RES supports remote workstations such as the FACOM U-200 system, to which is typically attached a typewriter, line printer, and card reader. Controlling each U-200 (or equivalent workstation) is a **terminal station system program** (TSSP), which communicates efficiently and compatibly with the VTAM and NCP components of OS IV/F4 at the central installation. See Chapter 6, "Telecommunication Management" for details of VTAM and NCP. Fig. 3.3 shows a typical RES configuration.

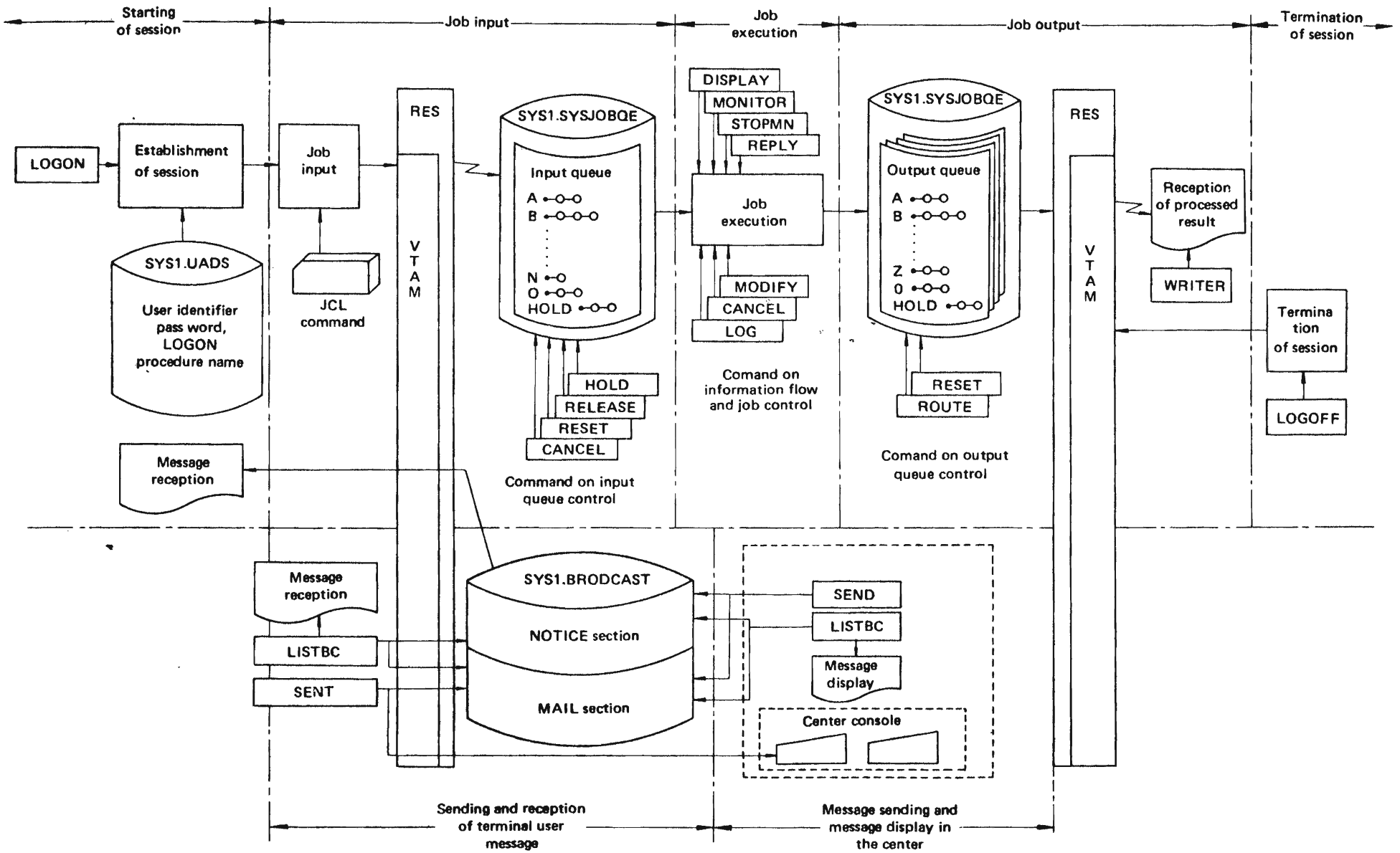


Fig. 3.2 Flow of processing in RES

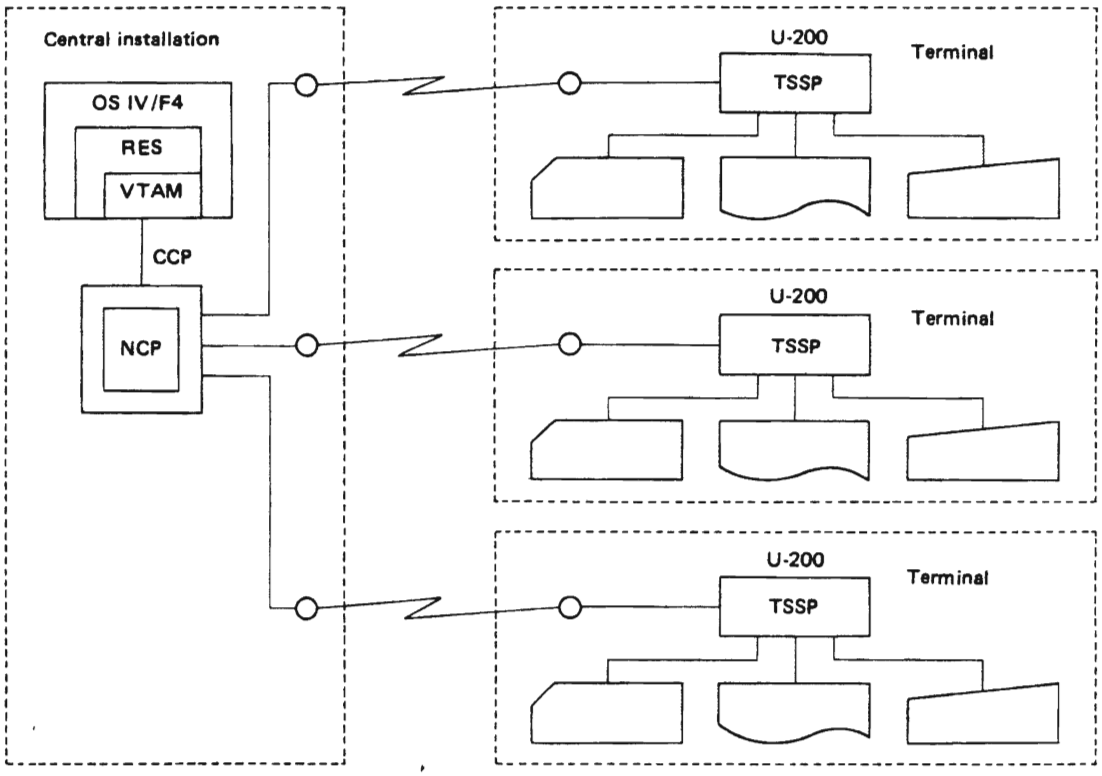


Fig. 3.3 Typical RES configuration

3.2 LOGON AND ENTERING JOBS

3.2.1 Starting a Session

As outlined in Section 3.1.3, a remote operator must enter a LOGON command after establishing a communications link between his terminal and a central OS IV/F4 installation. In addition to the name of this terminal and its associated password(s), the operator must furnish the name of the procedure for the RES reader and writer he wishes to manage this session. These parameters are stored centrally for all terminals in the user attribute data set (SYS1.UADS).

During LOGON for this terminal, any waiting broadcast messages are sent to it by RES; the system broadcast data set is described in Section 3.5.1. If a terminal is capable of accessing the AIM and TSS subsystems at the central installation, the remote operator must indicate their operational parameters during RES logon. As described in Section 3.1.3, it is unnecessary to logon if the terminal accesses RES via a dedicated line.

User attributes

RES accesses the **user attributes data set** (SYS1.UADS) when logging on remote terminals to determine their processing attributes and usage authorizations. Each member of SYS1.UADS (a partitioned data set) contains the password, default LOGON procedure name, and other attributes corresponding to this terminal identification, which is the name of the member. RES merges parameters of the LOGON command with those of the corresponding PDS member to define how this session is to be managed:

- Authorized terminal(s) for each remote user.
- Authorization to omit the LOGON procedure name from LOGON commands.
- How job outputs may be routed.
- Upper limit for job prioritles (PRTY parameter of JOB statement).

Identification and passwords

Corresponding to each terminal identification are an arbitrary number of passwords, as outlined in Section 3.1.3. To each password corresponds an arbitrary number of LOGON procedure names, as shown in Fig. 3.4.

As each session starts, RES verifies that the LOGON procedure name and password are valid for this terminal. If a particular terminal is permitted to omit the LOGON procedure name, RES supplies a default name previously generated for this terminal.

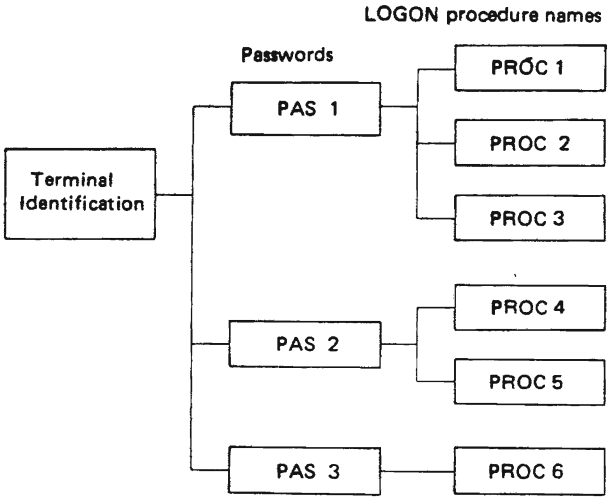


Fig. 3.4 Structure of identifications and passwords

Readers and writers
Remote operators cannot directly issue START commands to RES readers and writers. Hence, associated procedures are cataloged at the central installation and invoked automatically when a session starts, according to the procedure name furnished in the LOGON command (or the default procedure name).

Status commands
Either a central or remote operator can issue a DISPLAY command to show which terminals are currently active. By issuing a MONITOR command, the central operator can continually display which terminals are logging on and off.

3.2.2 Submitting and Controlling Jobs

While entering/receiving jobs, a remote operator can issue the following commands:

- A STOP command halts the associated RES reader.
- A CANCEL command deletes the indicated job, whether it is being entered, already enqueued for execution, executing, or awaiting output processing.
- A RESET command changes the class and/or priority of a job (or class of jobs).
- A HOLD command holds a job (or class of jobs) from executing (i.e., moves it to the input hold queue).
- A RELEASE command releases a held job.
- A DISPLAY command furnishes the remote operator with information about all jobs awaiting execution, executing, or awaiting output processing at the central installation.

After all jobs have been submitted at a particular remote terminal, the remote operator need not continue the session. If he logs off, he can subsequently log on and receive any outputs prepared by OS IV/F4 up to that point (or while RES transmits these outputs)

3.3 PROCESSING JOBS

Jobs submitted remotely are processed exactly like those submitted centrally to OS IV/F4. During a session, a remote operator can issue the following commands to control jobs:

- CANCEL deletes a specified job, whether being entered, awaiting execution, executing, or awaiting output processing
- REPLY replies to queries transmitted to this terminal by the central installation
- MODIFY transmits a parameter to a currently executing job controlled by this terminal
- MONITOR displays messages at the terminal whenever jobs submitted from this terminal start or complete execution
- STOPMN stops the dynamic job-status display requested by a prior MONITOR command.

3.4 PROCESSING JOB OUTPUTS

RES furnishes essentially the same output facilities as JES, described in Section 2.7 above.

3.4.1 Output Classes

SYSOUT data are written onto spool data sets by processing programs, controlled by entries on the

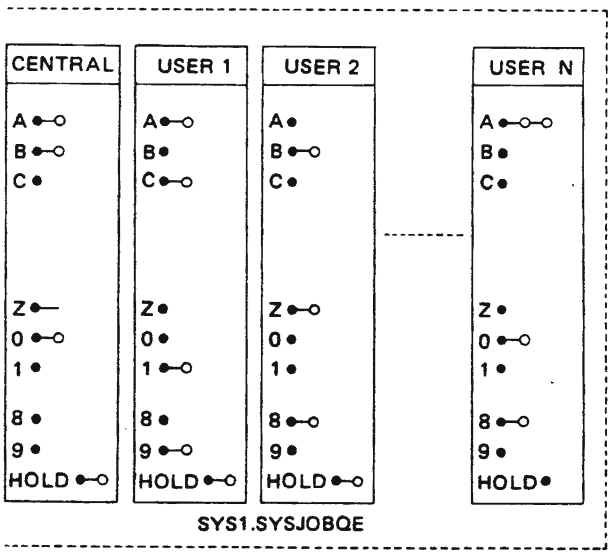


Fig. 3.5 Output queue structures

system job queue (SYS1.SYSJOBQE). Corresponding to each terminal is the same queue structure as for the central installation: SYSOUT queues by class, plus an output hold queue. Hence, if a total of N terminals are defined to RES — irrespective of the number simultaneously active — JES and RES together define N+1 output queue structures, as shown in Fig. 3.5.

Remote operators define which SYSOUT classes have what priority levels for transmission to their terminals, using WRITER commands to start these RES writers and MODIFY commands to modify classes and/or their priorities.

3.4.2 Routing Outputs

Outputs from each remotely-submitted job can be returned to the submitting terminal (the default destination), the central installation, or another terminal. Likewise, outputs from centrally-submitted jobs can be directed to remote terminals. Two facilities are available for routing jobs:

- DEST parameter on a DD statement
  The job submitter merely codes the terminal identification where he wants one or more SYSOUT data sets sent as DEST-parameter values on corresponding DD statements. The central installation has a routing identification of "LOCAL".
- ROUTE command
  The job submitter can optionally furnish a ROUTE command with his JCL, indicating to which remote terminal — or the central installation (LOCAL) — his entire job output should be routed. This is in contrast to the DEST parameter, which only routes individual SYSOUT data sets. A central/remote operator can issue a ROUTE command from his console to redirect outputs from one job (or from a class of jobs, or from all jobs destined for one terminal) to another terminal or to the central installation.

3.4.3. Limitations on Rerouting Outputs

RES verifies that requests to reroute outputs are properly authorized by comparing **destination control values** (integers between 0 and 255) of the source and destination for each job. The values are set by the central operator, usually based on default values stored in the SYS1.UADS data set. A ROUTE command is valid and effective only if the destination control value for the new destination (central installation or a terminal) is at least as large as the destination control value of the issuer of the ROUTE command.

For example, Fig. 3.6 shows the destination control values for a network of four terminals accessing a central installation.

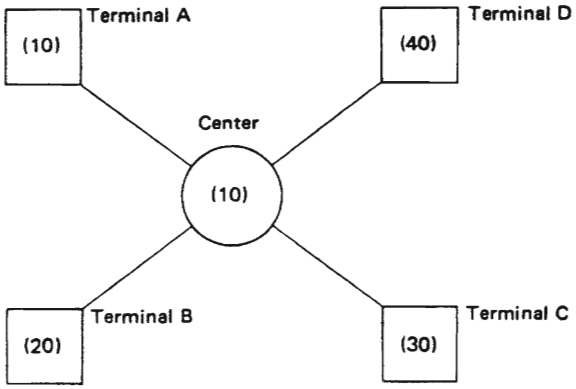


Fig. 3.6 Example of destination control values

Terminal A can route its outputs to the central installation or any other remote terminal. Terminal B can route its outputs to terminals C and D; etc. Outputs from centrally-entered jobs can be routed to any terminal.

3.5 CREATING AND RECEIVING MESSAGES

Certain messages are exchanged between the central operator and remote operators. Other messages are originated by the central operator for display to users on their SYSOUT listings.

3.5.1 Broadcast Data Set (SYS1.BRODCAST)

The broadcast data set contains messages exchanged by operators. Since all terminals of a network are not necessarily active when a message is prepared for them, it is necessary for RES to hold such messages on DASD at the central site until each terminal has had the opportunity to receive these messages. SYS1.BRODCAST is divided into sections for NOTICEs and MAIL. Messages are entered into SYS1.BRODCAST by SEND commands issued by central/remote operators.

- **NOTICE section**
  This contains messages to be displayed to all remote operators, identified by message numbers. A central operator can add or delete messages from this section by SEND commands; remote operators can only display messages, not add or delete them.
- **MAIL section**
  This contains messages for individual terminals entered by central or remote operators. Each set of messages is segregated from those for other terminals. When a terminal logs onto RES—or issues

a LISTBC command subsequently—any accumulated messages from its subsection of MAIL are displayed and then deleted from MAIL.

3.5.2 SEND Command

SEND commands can be issued by central or remote operators to communicate with one another. If the destination terminal is active, it can receive messages immediately. Otherwise, messages are handled in three different ways according to a SEND parameter:

- **Unsent messages are deleted**
  The sender is notified that his message was not sent successfully.

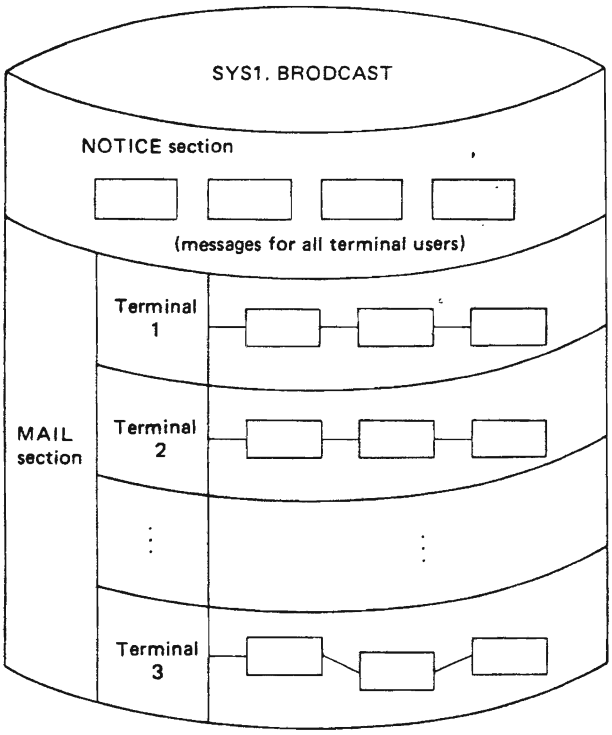


Fig. 3.7 Structure of SYS1.BRODCAST

- **Unsent messages are stored in SYS1 BRODCAST**
  When the destination terminal next becomes active, RES sends these messages from the MAIL section of SYS1.BRODCAST.
- **All messages are stored in SYS1.BRODCAST**
  Hence, these messages are received only when the next session starts at the destination terminal or when its operator issues a LISTBC command.

All messages directed to the central installation are immediately sent.

The central operator can create a message for all terminals by a SEND command directed to the NOTICE section of SYS1.BRODCAST. He can also display or delete any message in either section.

3.5.3 LISTBC Command

Remote operators use LISTBC commands to display relevant messages in the broadcast data set. A remote operator can choose whether to display NOTICE messages, MAIL messages, or both. The central operator can display part/all of the entire broadcast data set. (However, when he displays MAIL messages for a particular terminal, these messages are not deleted, as they are when displayed by the operator of that terminal.)

3.6 CENTRAL OPERATIONS

3.6.1 Generating RES

During OS IV/F4 system generation, various RES parameters and options are defined, including attributes for each terminal which are stored in the system parameters library (SYS1.PARMLIB). Using the ACCOUNT utility program, each installation defines the system broadcast (SYS1.BRODCAST) and user attributes data sets (SYS1.UADS).

3.6.2 Starting and Stopping RES

The central operator starts RES with a START command after initiating VTAM. Terminals can thereafter access this system until the central operator issues a STOP command to RES, which takes effect after all RES readers and writers conclude their current activities.

3.6.3 Creation and Maintenance of RES System Data Sets

The central operator issues ACCOUNT commands followed by various commands to creat, display, and maintain elements of the SYS1.BRODCAST and SYS1.UADS data sets. The commands available are as follows:

ADD command
- Enters the identification for a new terminal.
- Adds passwords and LOGON procedure names to existing terminals.

CHANGE command
- Changes attributes for an existing terminal.

DELETE command
- Deletes a terminal identification.
- Deletes passwords and LOGON procedure names, either for a specific terminal or for all terminals.

LIST command
- Lists terminal identifications, passwords, and LOGON procedure names for the entire SYS1.UADS data set.
- List passwords and LOGON procedure names for a particular terminal.
- List all terminals using a specific password, and corresponding LOGON procedure names.
- List all terminals using a specific LOGON procedure name, and corresponding passwords.

LISTIDS command
- Lists all terminal identifications in SYS1.UADS.

SYNC command
- Create a new SYS1.BRODCAST data set based on the information in SYS1.UADS.

END command
- Last command of a sequence following ACCOUNT.

3.7 RES COMMANDS

A remote operator can inquire about the status of the central installation as well as send it messages and define how jobs are to be entered and processed. Certain central-site commands are not authorized for remote operators, such as START, HALT, and MOUNT. Likewise, certain RES commands are appropriate primarily for remote operators:
- LISTBC.
- LOGON.
- LOGOFF.
- ROUTE (also useful to central operator).
- SEND (also useful to central operator).

Table 3.1 OS IV/F4 operator commands

Command name	OS IV/F4 commands with RES parameters	RES commands	OS IV/F4 command only for central operators
START	X		
STOP	X	X	
CANCEL	X	X	
HOLD	X	X	
HALT			X
RELEASE	X	X	
RESET	X	X	
DISPLAY	X	X	
REPLY	X	X	
MOUNT			X
UNLOAD			X
VARY			X
SET			X
MODIFY		X	
WRITER	X	X	
LOGON & LOGOFF		X	
WRITELOG			X
SWITCH			X
MONITOR	X	X	
STOPMN	X	X	
MSGRT			X
CONTROL			X
SWAP			X
DUMP			X
LISTBC	X	X	
LOGON	X	X	
LOGOFF	X	X	
ROUTE	X	X	
SEND	X	X	

CHAPTER 4 DATA MANAGEMENT

4.1 OUTLINE OF DATA MANAGEMENT

With the diverse expansion of computer applications and development of more powerful computer systems, I/O units are being treated as common resources of a computer system separated from any specific processing programs. As a result, there is a need for common control of read/write operations performed by I/O units. The term **data management** is used to collectively denote these control programs. Data management programs, together with job, task, and recovery management programs constitute the OS IV/F4 control program.

The main functions of data management are given below:

- Control the generations of data sets (a namedset of related records).
- Blocking/deblocking records.
- Processing of volume/data set labels.
- Detect and process operation errors of I/O units.
- Minimize reprogramming because of changed I/O units.
- Centralized control over the location of data sets.
- Allocation of direct access storage device (DASD) area
- DASD protection.
- Exclusive control over data sets.
- Processing of data sets with various organizations.

Several categories of input/output devices may be controlled by the data management facilities of the operating system as follows:

- magnetic tape device.
- direct access storage device (disk pack device, magnetic drum device).
- unit record device (line printer, card reader, card punch, paper tape reader and paper tape punch).

An illustration of the different capabilities of the data management system are depicted in Fig. 4.1.

The indexed sequential access method (ISAM), which is widely used on third-generation systems, is processed in OS IV/F4 by the virtual storage access method (VSAM). VSAM is provided with an ISAM interface routine which is transparent to the ISAM application programmer. For a description of this capability see Chapter 5, Part 2 of this manual.

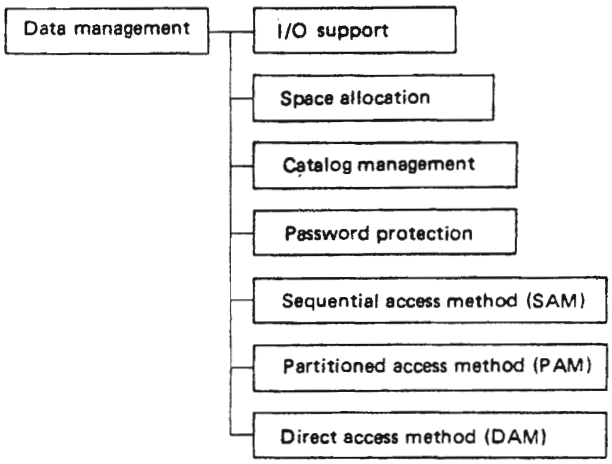


Fig. 4.1 Data management system

4.1.1 I/O Units

The main specifications of the 4 types of storage devices are given in the Tables 4.1 to 4.6.

Table 4.1 Magnetic tape device specifications

Device name \ Attributes	Recording density (BPI)	Tracks	Transfer speed (KB/sec.)
F610A$_1$	1600	9	320
F610A$_2$	900/1600	9	160/320
F610A$_3$	556/800	7	111/160
F611A	1600/6250	9	200/781
F611E	800/1600	9	100/200

Table 4.2 DASD specifications

- KL=length of each key field
- DL=length of each data field

Attributes / Device name	Device type	Tracks/ cylinder	No. of cylinders	Max. track capacity (bytes)	Volume capacity (megabytes)	Total block length (including inter and interblock gaps)	
						Keyd blocks	Unkeyed blocks
F479B Disk drive	Removable packs	19	808	13,030	200	191+KL+DL	135+DL
F4785 Disk drive	Removable packs	19	404	13,030	100	191+KL+DL	135+DL
F6625A Drum	Fixed-head	8	126	14,660	15	289+KL+DL	198+DL

Table 4.3 Line printer specifications

Attributes / Device name	Alphabet size (characters)	Print speed (lines/minute)	Print positions	Remarks
F650D	48 108	2,000 1,060	132/136*	150 optional positions
F651D	16 62	2,400 800	132/136*	Without Kana alphabet
F651E	19 109	1,600 800	132/136*	With Kana alphabet
F649A	36 62	900 630	136	Without Kana alphabet
F649B	19 109	900 370	136	With Kana alphabet

* Under switch control

Table 4.4 Card reader specifications

Attributes / Device name	Reading speed (cards/min)	Remarks
F668D	2000	
F671D	1250	Mark reading mechanism optional
F670B	600	Mark reading mechanism optional

Table 4.5 Card punch specification

Attributes / Device name	Punching speed (cards/min)	Remarks
F690D	250	Printing mechanism and punch/read mechanism are optional
F688B	91	

Table 4.6 Paper tape specifications

Attributes / Device name	Type	Reading/ punching speed (ch/sec.)	Remarks
F749F	Read	600/1200	6/8 channels, fixed
F749D	Read	300/600	6/8 channels selectable
F766A	Punch	200	6/8 channels selectable

4.2 PROGRAMS AND DATA SETS

The work involved in programming is considerably reduced because of the data management capabilities of the OS IV/F4 operating system. Data storage and retrieval is handled easily by data management programs. In addition, the necessary parameters for transferring data between memory and an external storage device need not be provided until the time of program execution. This feature,

called **device independence**, permits the programmer to use different storage devices without making permanent program changes.

4.2.1 Linkage Between Programs and Data Sets

Before the programmer can process any data sets, he must first provide the control program with detailed parameters required for storing and retrieving the data. This set of control information is referred to as a **data control block** (DCB). The DCB is initially constructed in the processing program by a DCB macro instruction. Specifications which can be defined in a DCB macro instruction are data set organizational format, block length, etc. However, the DCB constructed through a DCB macro instruction usually contains only a part of the parameters that is required for the processing of a data set. The other sources available to complete the content of the DCB at execution time are:

- DD job control statement
  Information not defined in a DCB macro instruction may be coded in the DCB parameter of the DD job control statement.
- Data set label
  During the opening of an existing data set, information in the data set label is read and stored in the DCB area of the program.
- Modification through DCB exit
  DCB information may also be provided by default options assumed in the OPEN macro instruction and by the user program with the DCBD macro instruction or a DCB exit routine.

Fig. 4.2 illustrates the linkage between programs and data sets.

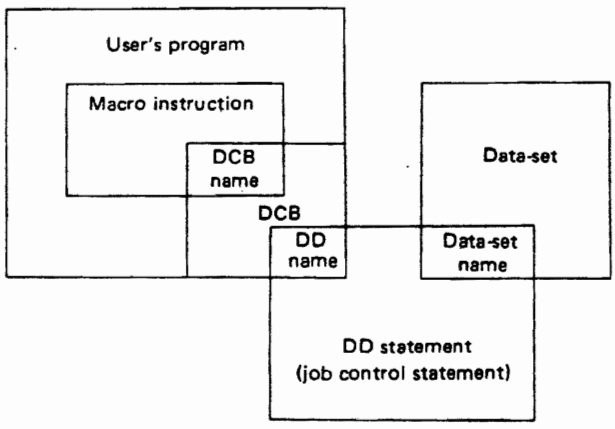


Fig. 4.2 Linkage between program and data sets

4.3 VOLUMES AND DATA SETS

As stated previously, the following I/O units can be utilized in an application program operating under the OS IV/F4 system:

- Magnetic tape device
- Direct access storage device (magnetic disk, magnetic drum)
- Unit-record device (line printer, card reader, card punch, paper tape reader and paper tape punch)

The medium onto which data is recorded by a magnetic tape unit and direct access unit is called a **volume.** The concept of a volume is not applicable to other devices, such as the unit-record device, display unit, and special I/O units. The volumes associated with a magnetic tape unit and direct access unit are called a magnetic tape volume and direct access volume respectively. A volume may contain multiple data sets, or a single data set may extend to 2 or more volumes.

4.3.1 Volumes

The media unit for magnetic tape (reel) or direct access storage (pack) is called a volume. Volumes may be described in terms of data sets, operating mode, access mode and management mode.

The following relationships may exist between volumes and data sets:

- single data set/single volume.
- single data set/multiple volumes.
- multiple data sets/single volume.
- multiple data sets/multiple volume.

Volumes can be classified into the according to operating mode:

- Private volumes
  Volumes reserved for specific jobs only.
- Public volumes
  Volumes that may be used commonly by any number of jobs.
- Storage volumes
  Same as public volumes, but which are demounted after their last use in a job step.
- Scratch volumes
  Magnetic tape volumes which are used when a nonspecific volume request is made and the data set is temporary.

Volumes can be classified into the following types according to access mode:

- Random access volumes
  These are volumes onto which records may be read or written without consideration of sequencing. These volumes can only reside on DASD devices.

- Sequential access volumes
  The read/write operations must be performed sequentially within the data set starting at the beginning (tape volumes may be read in reverse starting at the end).

Volumes can be classified as follows according to management mode:

- Standard volumes
  These are volumes having established management mode and recording formats for data sets. This type of volume can receive the services offered by the data management program.
- Non-standard volumes
  These are volumes having no fixed mode of management or no fixed recording formats for data sets. The data sets can have formats which can freely be altered by the user according to his needs.

4.3.2 Data Sets

A **data set** is defined as a named collection of related data for processing by a computer. It is composed of records, a single record being the unit of data for processing by a computer program.

Records and blocks

The unit of data physically recorded on an I/O

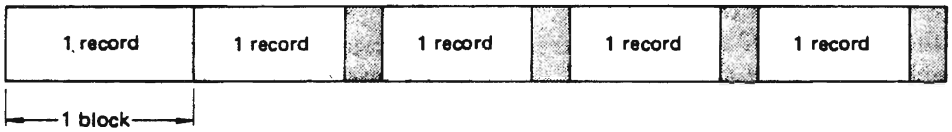


Fig. 4.3 Unblocked records

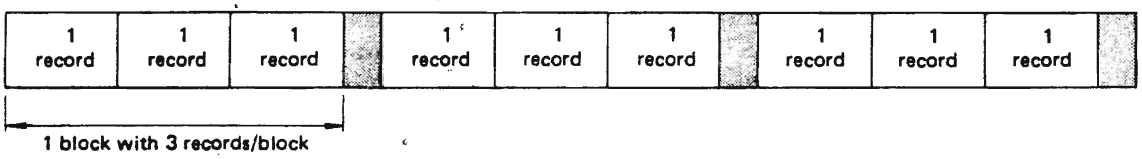


Fig. 4.4 Blocked records

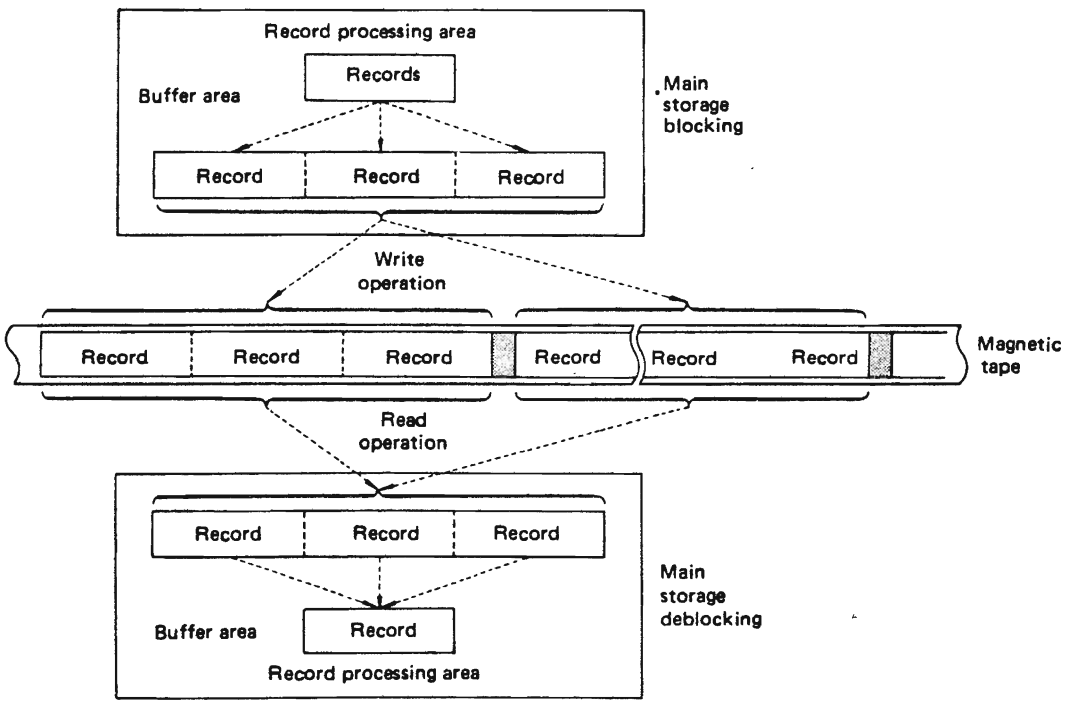


Fig. 4.5 Blocking and deblocking

device is called a **block.** When two or more records are included in one block, the records are said to be **blocked.** The number of records which constitute one block is called the **blocking factor.** When each record occupies an entire block, it is called an **unblocked record.** Fig. 4.3 and 4.4 illustrate the concept of records and blocks.

The process of concatenating records into 1 block for transfer to a storage medium is called blocking.

Conversely, the process of transferring one record at a time from the blocks on the recording medium to main storage is called deblocking.

Fig. 4.5 illustrates the concept of blocking and deblocking by using a magnetic tape example.

Record format

There are four kinds of record format depending on whether the length of the records in the data set are fixed or not. All the records in each data set must have the same format.

- Fixed length record (F-format)
  Records with this format have equal length.
- Variable length records (V-format)
  Records of different length contained in a single data set are called **variable length records.** The length of each record is contained in a **record descriptor word** (RDW) at the beginning of the record. Similarly, the length of each block is contained in a **block descriptor word** (BDW) at the beginning of the block. Fig. 4.6 indicates the relationship of a variable length record to a block.

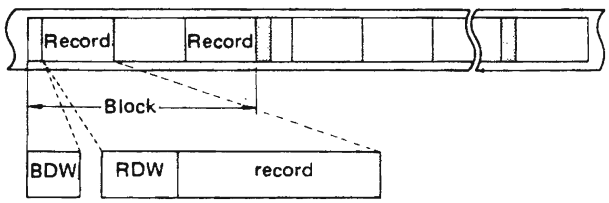


Fig. 4.6 V-format record

- Variable spanned records (VS-format)
  Variable spanned records are records whose length is greater than the length of their block. Fig. 4.7 depicts the variable length spanned record format.
- Spanned records are divided into units called segments. A word (segment description word) containing the length of a record segment is located at the beginning of the record. Multiple segments may constitute one record; therefore, the total record length can exceed the maximum block length (maximum 32,760 bytes).
- Undefined length record (U-format)
  Undefined length records cannot be classified in any of the above categories: F-format, V-format, or VS-format. Undefined length records may not be blocked under data management control. In addition, these records do not contain any indicator of record length. Thus the user must perform all blocking and deblocking within the program without the aid of data management processing. Fig. 4.8 shows the record to block relationship for U-format records.

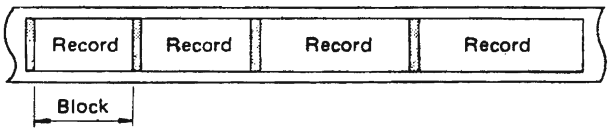


Fig. 4.8 Undefined length records

Control character

A **control character** is the specification made at the beginning of a record to indicate a carriage control channel when the data set is printed or for stacker selection control when the data set is punched.

Although the character is part of the record in storage, it is never printed or punched. However, it does require a one-byte space in the buffer area. Therefore, the determination of buffer size should

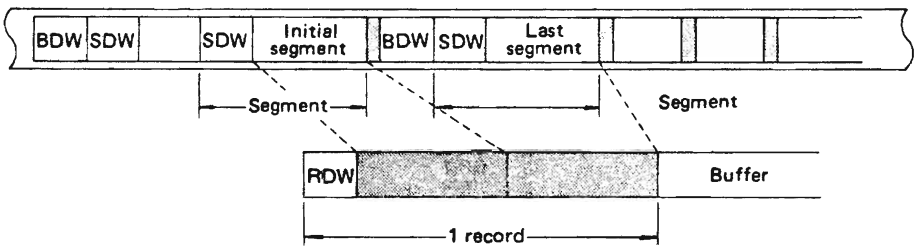


Fig. 4.7 VS-format record

include this control character space requirement. If the immediate destination of the record is a device, such as a disk, the does not recognize the control character, the system assumes that the control character is the first byte of the data portion of the record. If the destination of the record is a printer or punch and the user has not indicated the presence of a control character, the system regards the control character as the first byte of data.

4.3.3 Magnetic Tape Volumes

Because data sets on magnetic-tape devices must be organized sequentially, the operating system does not require space allocation procedures comparable to those for direct access devices. When a new data set is to be placed on a magnetic-tape volume, the user must specify the data set sequence number if it is not the first data set on the reel. The operating system positions a volume with standard labels, American National Standard labels, or no labels so that the data set can be read or written. If the data set has nonstandard labels, provisions for volume positioning must be made in nonstandard label processing routines in the user program. All data sets stored on a given magnetic tape volume must be recorded in the same density.

Each data set and each data set label group on magnetic tape that is to be processed by the operating system must be followed by a tapemark (see Fig. 4.9). Tapemarks cannot exist within a data set. When the operating system is used to create a tape with standard labels or no labels, all tapemarks are automatically written. Two tapemarks are written after the last trailer label group on a volume to indicate the last data set on the volume. On an unlabeled volume, the two tapemarks are written after the last data set.

When the operating system is used to create a tape data set with nonstandard labels, the delimiting tapemarks are not written.

If the data set is to be retrieved by the operating system, those tapemarks must be written by the user's nonstandard label processing routine. Otherwise, tapemarks are not required after nonstandard labels since positioning of the tape volumes must be handled by installation routines.

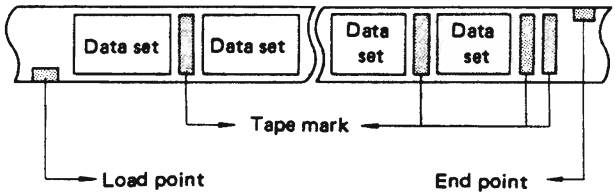


Fig. 4.9 Data sets on magnetic tape volumes

Types of label format

The types of labels which can be processed on magnetic tapes are stated below:

- Standard label.
- ANSI label.
- Standard user label.
- Non-standard label and no label.

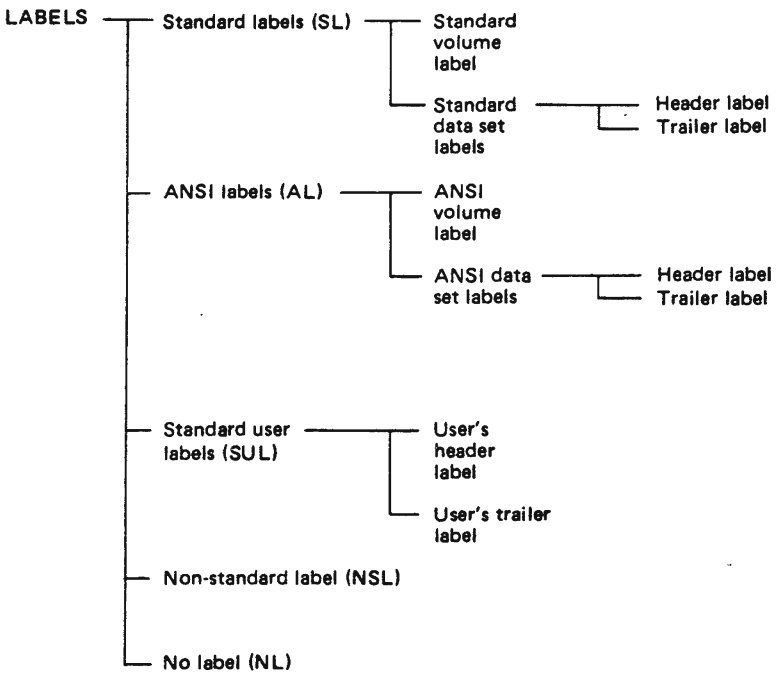


Fig. 4.10 Summary of label types

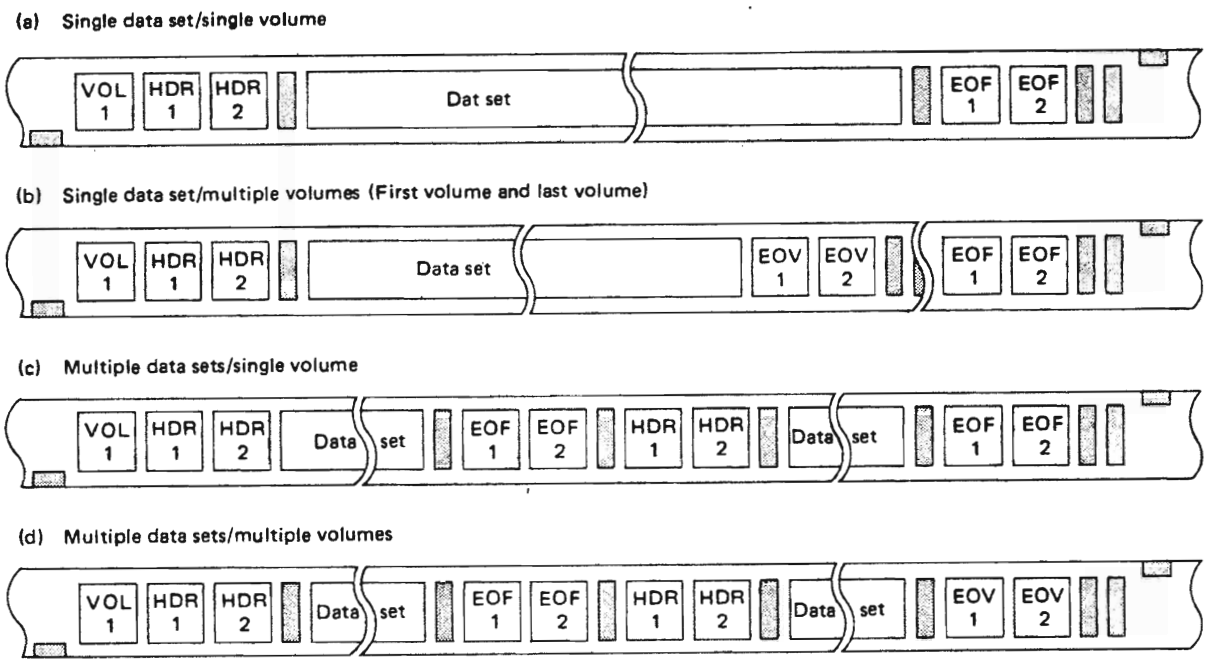


Fig. 4.11 Standard label configuration of magnetic tape

Each of the above types can be further classified into volume labels which identify the volume and indicate the attributes of the volume, and data set labels which identify data sets and indicate the attributes of data sets contained in their respective volumes. Fig. 4.10 illustrates the entire label system adopted here.

Standard label format

The labels commonly produced and processed by data management are called **standard labels.**

The standard label configuration depends on the data sets contained in the volume. The possible combinations are depicted in Fig. 4.11.

- Standard volume label
  The standard volume label (VOL1) occupies the initial data block of the volume and has a block length of 80 bytes. It contains the attributes of that volume such as the volume serial number, owner's name, and so on.
- Standard data set label
  Standard data set labels exist as a special initial data set preceding the user's data set and a special end data set following the user's data set. The initial label is called the header label (HDR) and the label following the user's data set is called the trailer label (EOF). The EOF may be of two types: trailer label for a volume with one data set, and trailer label for multiple volumes with one data set.

The system of standard data set labels is shown in Fig. 4.12.

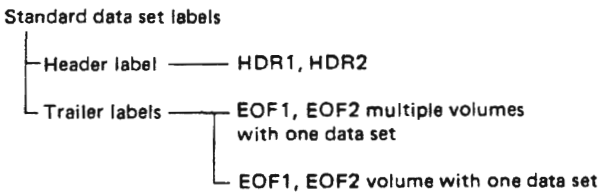


Fig. 4.12 Summary of standard data set labels

Non-standard labels and No label

- Non-standard labels
  Non-standard labels require special processing by the user program. Its contents are optional and may include items like: number of data sets, data set arrangement, or number of tape marks, tape mark position, and so on. Its contents including the position of check-head, etc. is the responsibility of the user.
- No label
  This type refers to volumes with absolutely no labels at all. The tape contains only data sets and tape marks. Therefore, all data set information must be contained in the DD statement or DCB for the data set.

The data on magnetic tape volumes can be in either EBCDIC or ASCII. ASCII is a seven bit code consisting of 128 characters. It permits data on magnetic tape to be transferred from one computer to

another even though the two computers may be products of different manufacturers.

Data management support of ASCII and of American National Standard tape labels is such that data management can translate records on input tapes in ASCII into EBCDIC for internal processing and translate the EBCDIC back into ASCII for output. Records on such input tapes may be sorted into ASCII collating sequence.

4.3.4 Direct Access Volumes

Direct access volumes are used to store executable programs, including the operating system itself. Direct access storage is also used for data and for temporary working storage. One direct access storage volume may be used for many different data sets, and space on it may be reallocated and reused.

A **volume table of contents** (VTOC) is used to account for each data set and available space on the volume.

Each direct access volume is identified by a volume label, which is stored in track 0 of cylinder 0. The user may specify up to seven additional labels, located after the standard volume label, for further identification.

The VTOC is a data set consisting of data set control blocks (DSCBs) that describe the contents of the direct access volume. The VTOC can contain seven kinds of DSCBs, each with a different purpose and a different format.

Each direct access volume is initialized by a utility program before being used on the system. The **initialization program** generates the volume label and constructs the table of contents.

When a data set is to be stored on a direct access volume, the user must supply the operating system

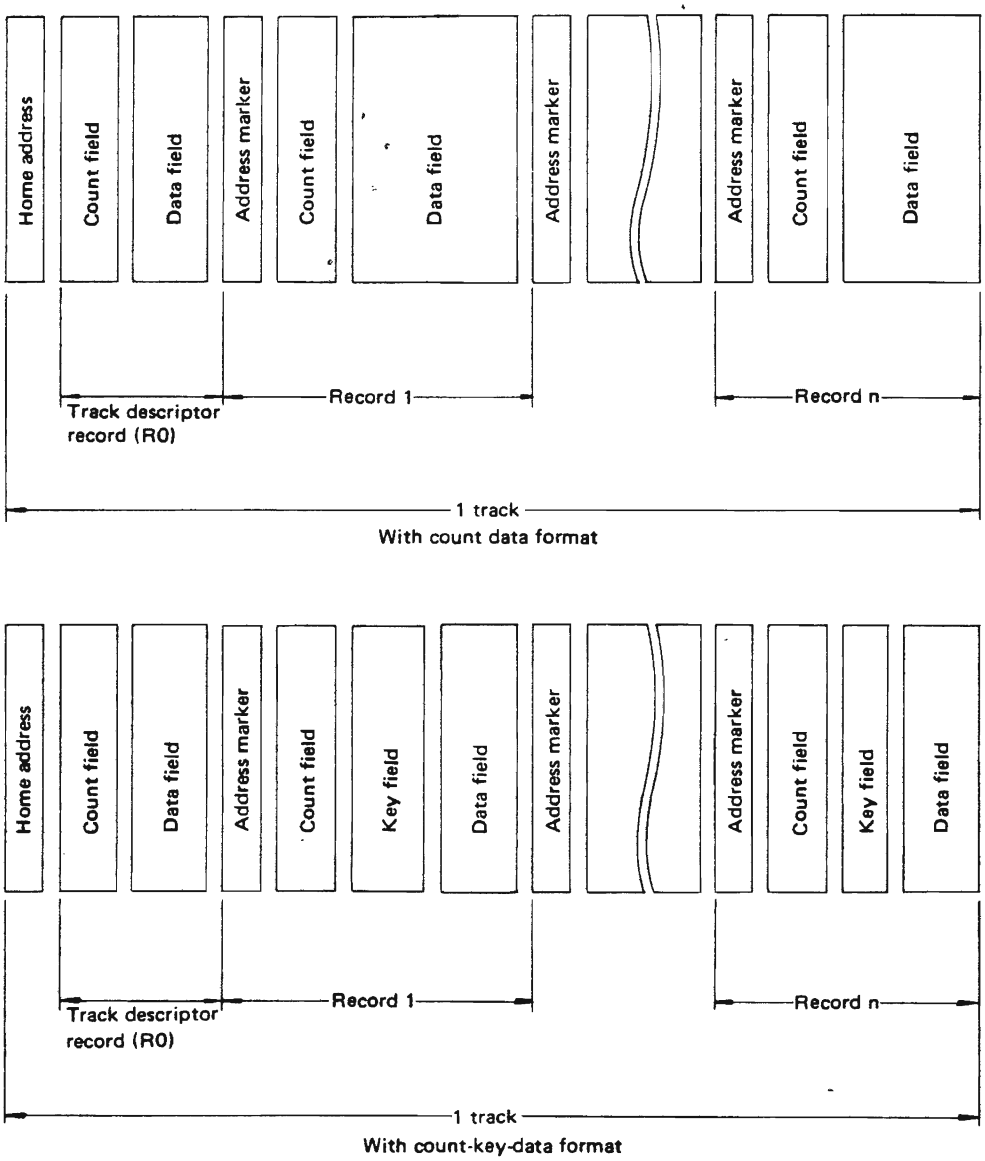


Fig. 4.13 Track formats

with the amount of space to be allocated to the data set, expressed in blocks, tracks, or cylinders. Space allocation can be independent of device type if the request is expressed in blocks. If the request is made in tracks or cylinders, the user must be aware of such device considerations as cylinder capacity and track size.

Track format

Information is recorded on all direct access volumes in a standard format. In addition to device data, each track contains a track descriptor record (capacity record or R0) and data records.

As shown in Fig. 4.13, there are two possible data record formats – count-data and count-key data – only one of which can be used for a particular data set.

In addition to device data, the count area contains eight bytes that identify the location of the record by cylinder, head, and record numbers, its key length (0 if no keys are used), and its data length.

If the records are written with keys, the key area (1 to 255 bytes) contains a record key that specifies the data record by part number, account number, sequence number, or some other identifier. In some cases, records are written with keys so that they can be located quickly.

DASD structure

Direct access volumes served by the data management program are composed of IPL records (1, 2), volume label, VTOC (volume table of contents) and data sets.

Fig. 4.14 below shows the structure of a direct access volume.

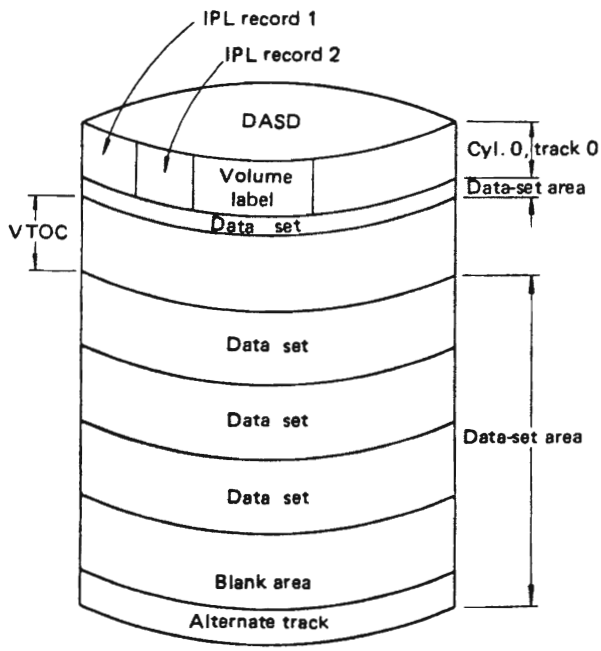


Fig. 4.14 Structure of direct access volume

Initial program loader (IPL) records

IPL records (1, 2) consist of an eight byte count field, four byte key field, and data field.

The data field is 24 bytes long. The IPL records for the system volume (volume on which the operating system resides) contains a program to load the control program into main storage.

Volume label

This label indicates the overall attributes of the volume (volume serial number, owner's name, VTOC address, etc.). This information is placed in the **volume label** at the time of its initialization. The volume label and IPL records are located at cylinder-0, track-0.

VTOC

The VTOC (volume table of contents) consists of tables containing device information, data set characteristics and other control information. Each of these tables is called a DSCB (**data set control block**). Each data set on a direct-access volume has one or more DSCBs to describe its characteristics. The DSCB is 140 bytes, consisting of a 44 byte key and a 96 byte data portion.

There are seven types of DSCB according to usage, as given below:

- DSCB0
- DSCB1
- DSCB2
- DSCB3
- DSCB4
- DSCB5
- DSCB6

After initialization, the relationship between the volume label and VTOC are as shown in Fig. 4.15.

To accomodate various categories of information about the volume and the data sets on it, the 140 byte blocks are formatted in different ways. DSCB formats 1 through 4 are designed for data set information; DSCB formats 5 and 6 describe the available or shared space.

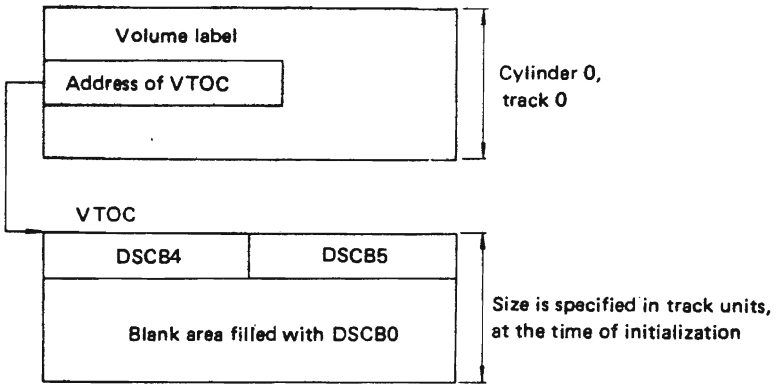
A single DSCB4 is located at the beginning of the VTOC. It is followed by at least one DSCB5. If there are any more DSCB5, they are chained from the first DSCB5.

If there are any DSCB6s, they are chained from the DSCB4. For every data set on the volume there is a DSCB4. A DSCB2 would also normally be used for data sets with index sequential organization, however, since OS IV/F4 does not support ISAM, reference to DSCB2 is possible only to allow for compatibility. DSCB's are found by using a search (equal) command with an argument of the DSNAME operand; they are not chained to one

another nor to the DSCB4. If the data set has more than three extents, a DSCB3 is chained from the DSCB1.

User labels, if used, occupy the first extent described by DSCB1. This extent, a separate one for each data set, is one track long; the labels from 80 byte data segments.

These characteristics are summarized in Table 4.7.



Note: VTOC area consists of 'n' tracks which form one continuous EXTENT. Its position is kept recorded in the volume label, the position of the VTOC can be anywhere on the DASD

Fig. 4.15 Volume label and VTOC

Table 4.7 Characteristics of DSCB

DSCB #	Conditions for existence	Chaining to other DSCB	Main contents
1	One DSCB is always constructed for each data set In addition, one is prepared for each volume in the case of a multivolume data set.	To DSCB3 when the number of EXTENTS is greater than 3. To DSCB2 in the case of IS organization.	Describes the attributes up to three extents of a data set.
2	One DSCB required in the index area of a volume when the data set is characteristics of IS organization.	To DS CB3 when the number of EXTENTS is greater than 3.	Characteristics of IS organization data sets.
3	One DSCB is constructed when the data set has more than 3 EXTENTS.	None	Describes remaining EXTENTS in the data set; up to 13 additional EXTENTS.
4	One DSCB is prepared at the time of volume initialization.	To DSCB6 when a shared cylinder allocation is made.	Describes the VTOC data set.
5	One DSCB is prepared at the time of volume initialization. More than one may be constructed as needed.	To the next DSCB5 when the unused EXTENTS on the volume exceed 26.	Describes the amounts of unused area of the volume.
6	One DSCB is constructed for shared cylinder allocation.	To next DSCB6 when shared EXTENTS exceed 26.	Describes the extent of space that are being shared by two or more data sets.
0	These are the unused areas in the VTOC. They are filled with binary zeros.	None	Binary zeros. This is available VTOC space.

The DSCB pointer process is depicted in Fig. 4.16.

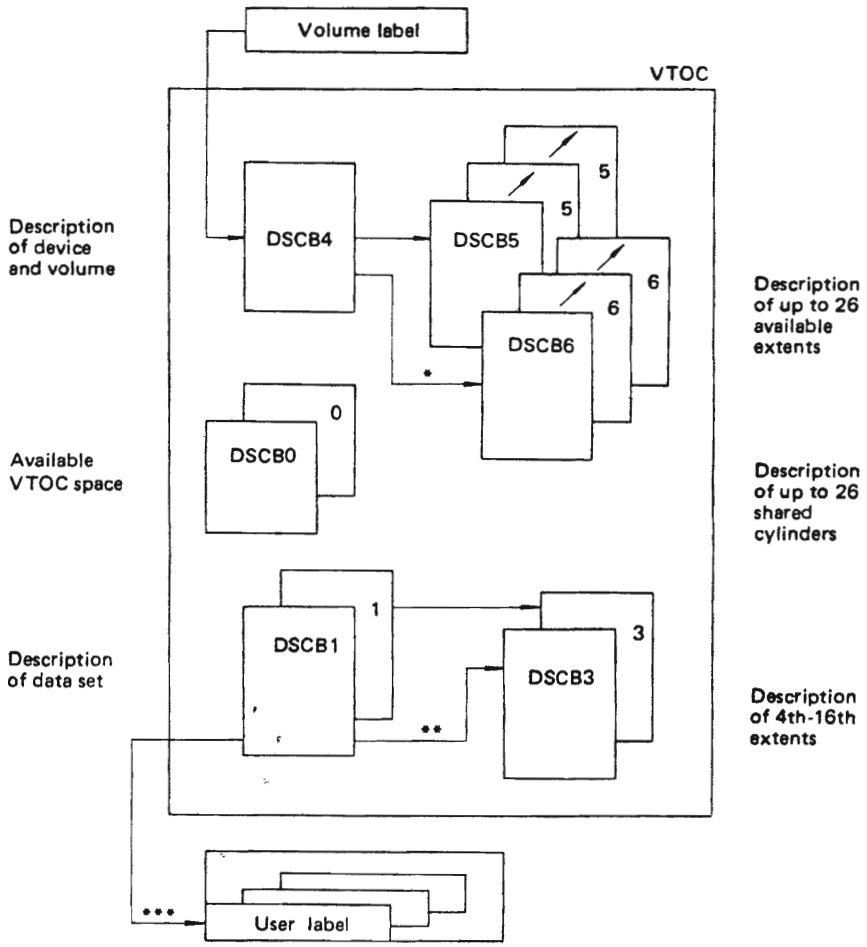


Fig. 4.16 DSCB concatenation

Record format and block format

Record and block formats on a direct access volume which can be handled by the data management program are given below.

- Fixed length records/unblocked.
- Fixed length records/blocked.
- Variable length records/unblocked.
- Variable length records/blocked.
- Variable spanned records/unblocked.
- Variable spanned records/blocked.
- Undefined length records.

Organization of data sets

The data management program can process datta sets with the following organizations:

- Physical sequential organization (PS).
- Partitioned organization (PO).
- Direct organization (DO).

For the specific format of data sets, refer to the respective access method (see data set access method, Section 4.6).

Error track

At the time of initialization by a system utility program, a read/write check is made to identify normal tracks and any error tracks. If an error track is detected, an error procedure is followed which permits the direct access volume to be used while compensating for the error track. The action taken depends on the device type.

- Disk pack device
  If an error is detected on a track of a disk pack at the time of initialization, it is linked with some other track (alternate track). This linkage makes it possible for the channel control program (EXCP) to access the auxiliary track during a read/write operation.
- Magnetic Drum Unit
  If an error track is detected on this unit at the time of initialization, the head is shifted to a spare track using hardware functions instead of linking the error track to an auxiliary track by a system utility.

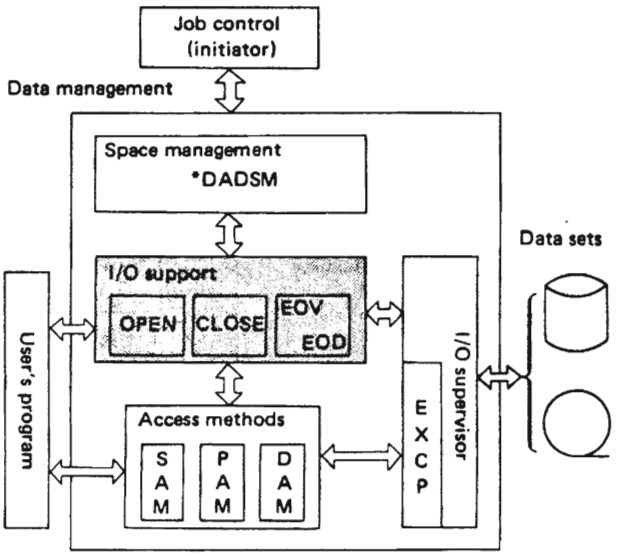
These procedures make the processing of error tracks transparent to the user. That is, the user need not be worried about error tracks during program operation.

4.4 INPUT/OUTPUT

I/O support is the center of the data management program. It supports I/O operations by providing procedures which allow for different access methods (SAM, PAM, DAM). The fuction of I/O support can be readily classified into the following:

- **Open function** . . . Essential preprocessing performed before actually accessing the data sets.
- **Close fuction** . . . Postprocessing of data sets.
- **EOV/EOD function** . . . Processing at the end of a volume or data set.

In addition to processing associated with specific access methods, there are some basic operations carried out by I/O support which are common to all access methods. Fig. 4.17 indicates the position occupied by I/O support relative to other data management facilities.



* DADSM: Direct access device space management

Fig. 4.17 Position of I/O support in data management

4.4.1 Open Function

When a program is assembled, the various data management routines required for I/O operations are not completely assembled until the DCBs are initialized for execution. To accomplish initialization an OPEN macro instruction must be issued. After all DCBs have been completed, the system ensures that all required access method routines are loaded and ready for use and that all channel word lists and buffer areas are ready.

The DCB information is collected and structured by merging together DCB data from the user's program, DD statement and data set label.

An explanation of the DCB merging steps depicted in Fig. 4.18 is:

- Prepare DCB by issuing DCB macro instruction.
- Store DCB information from the DD statement into JFCB (job file control block) on DASD.
- Store the information given in data set label of the input data set into a JFCB field that had not already been filled.
- Store DCB data contained in JFCB into the corresponding field of DCB that had not already been filled.
- Make modifications in the contents of the DCB by making use of a DCB merge exit routine.
- If the data set to be processed is opened for output the DCB data is stored in the JFCB.
- If the data set to be processed is opened for output a data set label is also prepared.

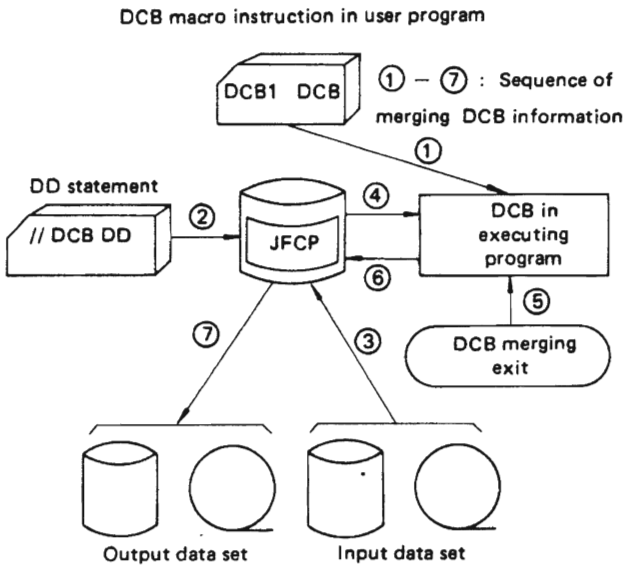


Fig. 4.18 DCB merging

4.4.2 Close Function

When I/O operations for a data set are completed, the user should issue a CLOSE macro instruction to return the DCB to its original status, handle volume disposition, create data set labels, complete writing of queued output buffers, and free virtual and auxiliary storage.

Specifically, the following actions may be performed by data management during the execution of the CLOSE macro instruction:

- Data set labels (trailer label) are constructed for magnetic tape volumes. For direct access

volumes, modification of information in the data set label (DSCB1) is completed.
- Positioning of magnetic tape volumes is carried out by specifying a parameter in the CLOSE macro instruction or through DISP parameters in the DD statement.
- After a data set has been closed, the DCB can be used for another data set. It can also be used for the same data set after any appropriate parameters have been changed. For example, a data set can be used as an output data set and then as an input data set during the same job step.
- A request for release of unused space to space management can be made if such a release of unused DASD spaces had not been indicated through the SPACE parameter in a DD statement (i.e., RLSE had not been specified as a SPACE parameter).

4.4.3 EOF/EOD Function

The EOV (end of volume)/EOD (end of data set) functions pass control automatically to the data management routines when the end of volume/end of data set conditions are indicated.
- EOV function
  When **end of a volume** (EOV1, 2) is detected while processing an input data set, the data management routine switches processing to the next volume.

  For an output data set, this data management function automatically continues the write operation on another volume of the same type. The output of records is continued after preparing a data set label (EOV1, 2) for the present volume and another data set label (HDR1, 2) for the next volume. If no such volume is available, the user's job is terminated.
- EOD function
  This function is applicable only for input processing. When the **end of data** set is encountered during input processing (i.e., when EOF1 and EOF2 are reached), control is transferred to the EOD exit provided in the users program.

4.4.4 Exits to Special Processing Routines

The DCB macro instruction can be used to identify the location of:
- an end-of-data processing procedure.
- a routine that supplements the operating system's error recovery routine.
- a list that contains addresses of special exit routines.

Special processing routines can be provided in the user's program to perform specific functions such as label processing, blockcount check, etc. The control program will transfer control to the user's program at different processing stages, like OPEN/CLOSE/EOV etc. depending on parameters set in the DCB macro instruction.

Setting of exits
The user provides for special processing exits by preparing processing routines and specifying their type and initial address in the DCB of the application program. Control is transferred to the particular processing routine at execution time based on this information.

Fig. 4.19 illustrates the relationship which exists between the DCB and the special processing routines.

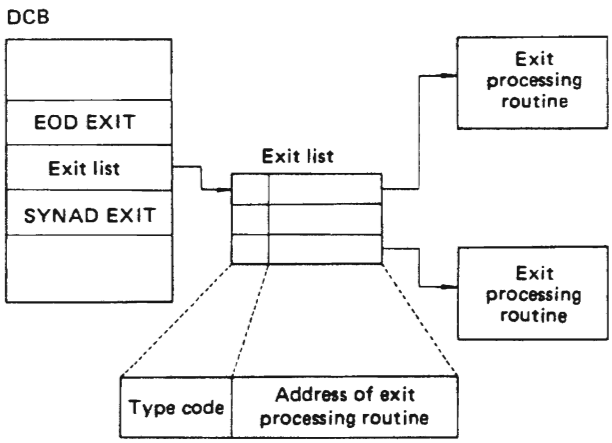


Fig. 4.19 Relationship between DCB and exit processing

Types of exits
Table 4-8 summarizes the exits that can be specified implicitly by providing the address of an exit list in the DCB.

4.5 BUFFER MANAGEMENT

The operating system provides several methods of buffer acquisition and control. Each **buffer** (virtual-storage area used for intermediate storage of input/output data) usually corresponds in length to the size of a block in the data set being processed. When using the queued access technique, any reference to a buffer actually refers to the next record (buffer segment).

The user can assign more than one buffer to a data set by associating the buffer with a buffer pool. A buffer pool must be constructed in a virtual storage area allocated for a given number of buffers of a given length. A **buffer pool** consists of a BCB (buffer control block) which shows its characteristics and an optional number of buffers. (See Fig. 4.20 below).

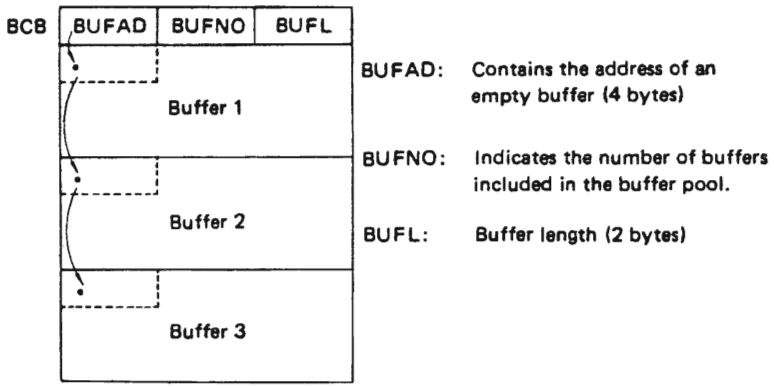


Fig. 4.20 Structure of buffer pool

Table 4.8 Types of exits and their functions

Type of exit routine	Type code*	Processing and functions
User's input header label	01	Process standard user input header labels
User's output header label	02	Process standard user output header labels
User's input trailer label	03	Process stnadard user input trailer labels
User's output trailer label	04	Process standard user output trailer labels
DCB merge	05	Process DCB exits
End of volume	06	Porcessing at the end of a volume
JFCB	07**	Specify storage address where JFCB is to be read-in.
213 ABEND	09**	Not provided
User's totalling computation	OA**	Specify the address of totalling computation
Block count	OB	Processing after unequal block count comparison by end of volume routine
Input trailer label delay	OC	Delay the processing of input trailer label from the time of EOD closing time.
Non-standard input trailer label delay	OD	Delay the processing of non-standard input trailer label, from the time of EOD closing time.
FCB image	10**	Define FCB images when opening a data set or issuing a SETPRT macro.
ABEND	11	Process ABEND exits
QSAM parallel GET	12	Specify the address of PDAB
Last entry item	80	Indicates end of parameter list

* Indicates hexadecimal 1 byte codes
** Indicates exit processing routine not necessary

4.5.1 Reservation and Releasing of Buffer Pools

The number of buffers assigned to a data set should be a tradeoff against the frequency with which each buffer is referred. A buffer that is not referred to for a relatively long period of time may be paged out. If this were allowed to happen to any considerable degree, it could result in a greater number of buffers actually decreasing throughout.

The type of requirement of buffer pool differs as follows, according to the adopted access method.

- QSAM
  A buffer pool is necessary here since buffer management is carred out by the control program.
- BSAM
  Since buffer management is the user's responsibility, in this method a buffer pool is not always necessary. However, buffer management work can be simplified by keeping a buffer pool reserved. Macro instructions may be used to secure or return buffers.

Reserving of buffer pool

There are three methods which may be used to reserve a buffer pool as given below. The user may select any one.

- Automatic construction at the time of a data set open operation.
- Reserving by GETPOOL macro instruction.
- Compiling a buffer pool using a BUILD macro instruction.

Releasing of buffer pool

The reserved buffer pool can be released by issuing a FREEPOOL macro instruction. Timing considerations applicable to the FREEPOOL macro instruction differ with each method of access.

- QSAM
  With this access method, the timing issue associated with the FREEPOOL macro instruction differs with each buffering method (see Section 4.5.3 Method of Buffering). When using the queued access technique, a data set must be closed

first. When using exchange buffering, the buffer pool must not be released until all the data sets have been closed.

- BSAM/BPAM/BDAM
  In this method, FREEPOOL macro instruction may be issued after all the buffers contained in the buffer pool are no longer needed.

4.5.2 Acquiring and Returning of Buffers

The method for acquiring and returning buffers to the buffer pool differs as follows according to the access method used.

- QSAM
  In this method, buffer management is done by the control program. The control program retrieves buffers from the buffer pool and assigns them to channel programs.
- BSAM/BDAM/BPAM
  In this method, buffer management is done by the user. Macro instructions which are provided for buffer management are GETBUF/FREEBUF. These macro instructions are useful when multiple DCBs make common use of a buffer.

4.5.3 Method of Buffering

When OSAM is used as the access method, the user program cannot directly control the buffers. In this case data management routines control the buffers. However, the following two buffer control macro instructions can be used.

- RELSE macro instruction
  By issuing this macro instruction, the input buffer is released and records from the next input buffer can be processed.
- TRUNC macro instruction
  By issuing this macro instruction, the current output buffer is truncated (that is, a short block length is constructed for output) and additional records are prepared in the next output buffer.

Two methods of buffering are available to the user when QSAM is the access method—simple and exchange. Each of these buffering techniques are described below.

- Simple buffering
  Here, the buffer is divided into small units, called buffer segments. In **simple buffering** the buffer segments are lined up contiguously in the buffer area. The buffer is allocated to a data set at the time of opening and is eventually returned back to the buffer pool when that data set is closed. The buffer pool, which the control program automatically reserves at the time of a data set opening, is released upon closing of that data set.
- Exchange buffering
  With this method, the buffer segments in the buffer are not necessarily contiguous in virtual storage. Moreover, since the buffer segments are not always associated with the same data set, the buffer structure does not remain constant.

  Generally, in **exchange buffering,** input data sets and output data sets share the same buffers. The transfer of records is eliminated by exchanging input buffer segments with output buffer segments and/or with a working area of processing program. In this way the program processing efficiency is improved. The buffer reserved at the time of a data set opening is not returned back to its buffer pool. Because some other data sets may be utilizing a buffer segment within the buffer, the buffer may not be returned back to its pool even when the data set is closed. Fig. 4.21 illustrates the simple buffering and exchange buffering methods.

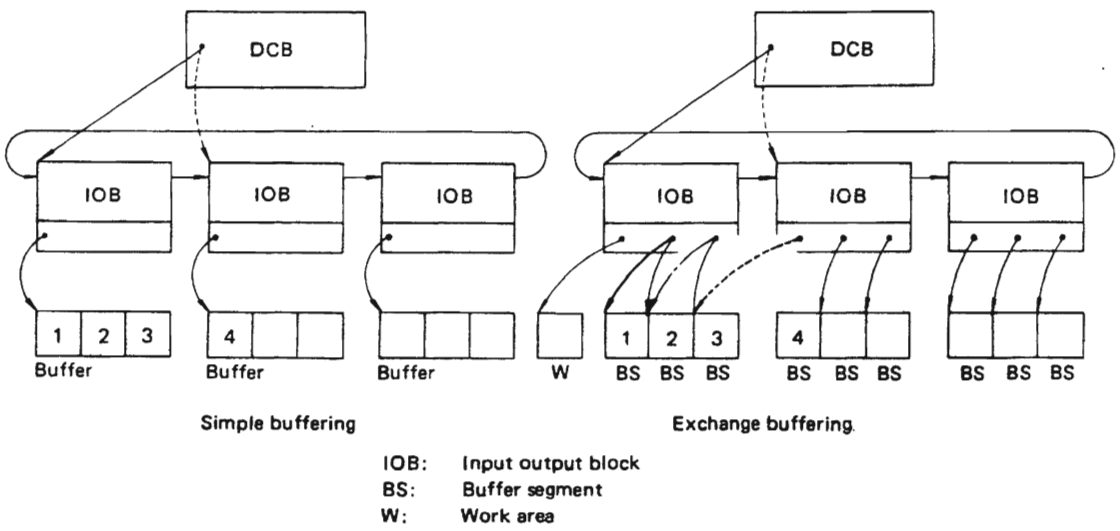


Fig. 4.21 Simple buffering and exchange buffering

For simple buffering, each record must be physically moved from an input buffer segment to an output buffer segment. It can be processed within either segment or work area.

In exchange buffering, the work area which contains records to be processed can be used as the buffer area of the processing program (substitute mode). Thus, there is no need to move the record for output.

4.6 DATA SET ACCESS METHOD

4.6.1 Access Technique

Two access techniques are available for processing data sets: queued and basic access. With the **queued** technique, data set processing is done in record units and the blocking/unblocking of records is carried out by the data management program. In **basic access** technique, data set processing is done in block units and the blocking/deblocking of records is performed by the user whenever necessary.

4.6.2 Data Set Organization and Access Technique

Different data set organizations can be handled in combination with the above two access techniques. Table 4.9 indicates the valid access techniques for each of the data set organizations.

Table 4.9 Access techniques and data sets

Data set organization \ Access technique	Queued technique	Basic technique
Sequential data sets	QSAM	BSAM
Partitioned data sets	QSAM	BPAM, BSAM
Direct data sets	–	BDAM, BSAM

QSAM : Queued Sequential Access Method
BSAM : Basic Sequential Access Method
BPAM : Basic Partitioned Access Method
BDAM : Basic Direct Access Method

4.6.3 Access Method Characteristics

Characteristics of access methods are shown in Table 4.10

4.7 PROCESSING A SEQUENTIAL DATA SET

4.7.1 Structure of a Sequential Data Set

A data set processed according to the physical sequence in which blocks are recorded is called a

Table 4.10 Characteristics of access methods

Access method \ Organization	Sequential organization		Partitioned organization	Direct organization
	QSAM	BSAM	BPAM	BDAM
Main access macros	GET, PUT, PUTX	READ, WRITE	READ, WRITE, BLDL FIND, STOW, NOTE POINT	READ, WRITE
I/O access and program synchronization	Executed by data management program	CHECK	CHECK	WAIT, CHECK
Record format	F,V,U,D format	F,V,U,D format	F,V,U format	F,V,U format
Buffer pool construction method and macros used for it	BUILDRCB, BUILD GETPOOL. Constructed automatically upon opening	BUILD, GETPOOL. Constructed automatically upon opening*	BUILD, GETPOOL. Constructed automatically upon opening*	BUILD, GETPOOL. Construced automatically upon opening*
Buffering method and buffering macros used	RELSE, TRUNCH. Simple/exchange buffering automatically conducted by system	GETBUT/ FREEBUF	GETBUF/FREEBUF	GETBUF/ FREEBUF/ FREEDBUF
Transfer mode (work area – buffer)	Shift, data positioning or substitute mode	–	–	–

* Buffer pools are automatically reserved during OPEN processing by the control program executing the BUILD, BUILDRCD, and GETPOOL macro instructions of QSAM.

sequential data set. Data sets on magnetic tape, paper tape, card and printer equipment are all sequential. The **sequential access method** (SAM) is used to access sequential data sets. Also, data sets with nonsequential organization on direct access devices can often be referenced in sequence with SAM. See Table 4.11.

Table 4.11 Sequential data set/device type attributes

Item \ Device names	I/O device	
	Magnetic tape unit	Direct access unit
Type of volume	Multivolume	Multivolume
Data set concatenation	Max. 255 data sets	Max. 255 data sets
User label	Usable	Usable

Volume Structure

- Magnetic tape volume

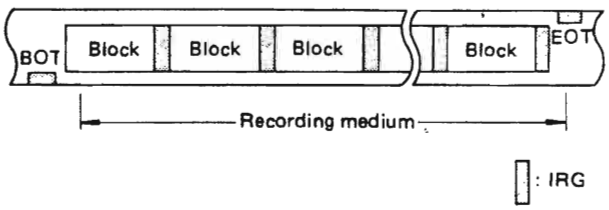


Fig. 4-22 Data sets on magnetic tape volume

Because of its physical characteristics, magnetic tape volumes can only store sequential data sets. Refer to Fig. 4.22.

- Direct access volume

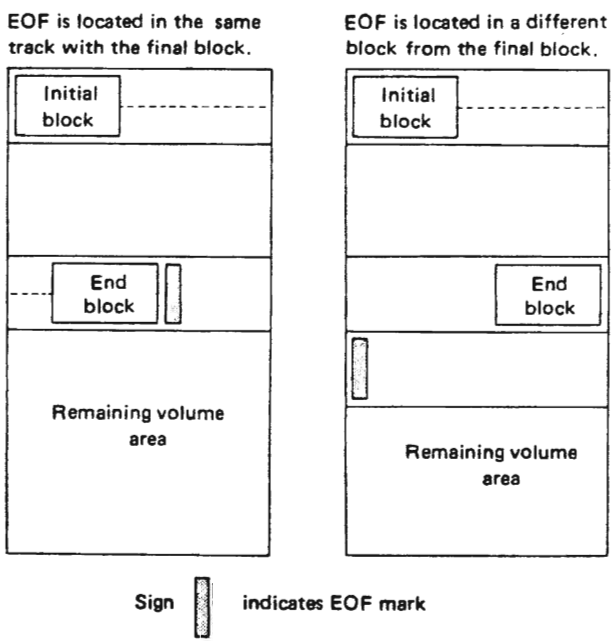


Fig. 4.23 Data sets on direct access volume

Sequential data sets on direct access volumes are constructed in the allocated data set area without requiring any system areas to describe the data set. At the end of every data set an EOF (end of file) mark is written. See Fig. 4.23.

Record format

Data management routines can process sequential data sets with the following four types of record formats:

- F-format (fixed length record).
- V-format (variable length record).
- U-format (undefined length record).
- D-format (variable length ASCII record).

Recording formats of unit-record device

The term unit record device is the collective name given to line printer/card reader/card punch units. The purpose of each of these units, and the associated record format/record length which they can handle is described below.

Line printer unit

The line printer unit can print the following three types of record formats:

- Fixed length (F-format).
- Undefined length (U-format).
- Variable length (V-format).

Because each line of print corresponds to one record, the record length should not exceed the length of one line on the printer (133 or 136 characters). For variable-length spanned records, each line corresponds to one record segment; the block size should not exceed the length of one line on the printer.

Card reader unit

The card reader unit can read the following types of record formats and character codes respectively.

- Record formats:
  - —Fixed length (F-format).
  - —Undefined length (U-format).
  - —Variable length (V-format).
- Character codes:
  - —EBCDIC mode.
  - —Column binary mode.
  - —Simple EBCDIC mode.

Card punch unit

The card punch unit can punch the following types of record formats and character codes respectively.

- Record formats:
  - —Fixed length (F-format).
  - —Undefined length (U-format).
  - —Variable length (V-format) including spanned records.
- Character codes:
  - —EBCDIC mode.
  - —Column binary mode.
  - —Simple EBCDIC mode.

Paper tape reader unit
The paper tape reader can read the following types of record formats:
- Fixed length (F-format).
- Undefined length (U-format).

4.7.2 Sequential Access Method

Two types of sequential access methods are available under data management – QSAM (which provides vides services at the record level) and BSAM (which provides services at the block level).

QSAM macro instructions
QSAM is equipped with the following macro instructions to perform processing in record units:
- GET
  Record read-in processing.
- PUT
  Record write-out processing.
- PUTX
  Record revise processing.
- RELSE
  Move buffer pointer to the next input buffer, ignoring the remainder of the present buffer.
- TRUNC
  Write a short block on the recording medium, ignoring the remainder of the present buffer.
- BUILD
  Prepare a buffer pool.
- BUILDRCD
  Reserve a buffer pool and record area.

The GET/PUT macro instructions have four main variations according to the user's program work area and the method of data transfer between I/O buffers.
- Move mode
  Commands of this mode shift data between the user's program work area and I/O buffers are described in Fig. 4.24.
- Data mode
  This mode is available to handle variable length spanned records. If this mode is used for input, only the data portion of the variable length spanned record& from the input buffer (that is the record portion excluding BDW and SDW) is transferred to the user's program work area. During output, this mode allows data from the user's program work area to be segmented and subsequently the record segments sent to the output buffer after having added BDW and SDW to the beginning of each record segment. Fig. 4.25 illustrates the movement of data after issuing GET/PUT macro instruction with the data mode.

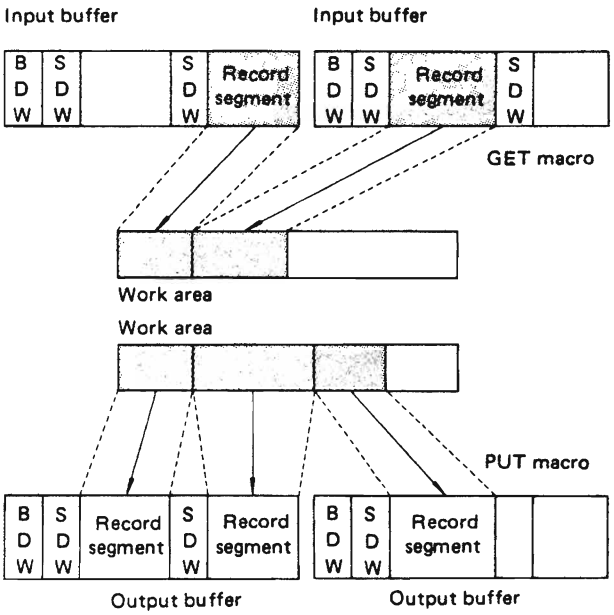


Fig. 4.25 GET/PUT macro instructions with data mode

- Locate mode
  When I/O macro instructions with the locate mode are used, a record work area is not necessary in the user's program. The I/O itself is used as the work area. The user's program determines the address of the I/O buffer from a general-purpose register. Fig. 4.26 indicates how the I/O buffer is pointed to after issuing a GET/PUT macro instruction with locate mode.

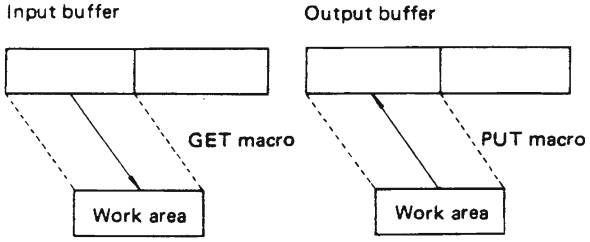


Fig. 4.24 GET/PUT macro instructions with move mode

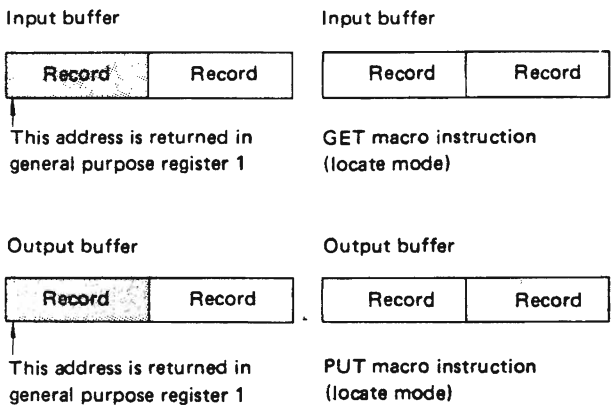


Fig. 4.26 GET/PUT macro instructions with locate mode

- Substitute mode
  In this mode, the addresses of the I/O buffer and the work area in the user's program are interchanged in general purpose register 1. No transfer of data is performed. Fig. 4.27 illustrates the interchange of the I/O buffer address and the work area address after issuing a GET/PUT macro instruction with substitute mode.

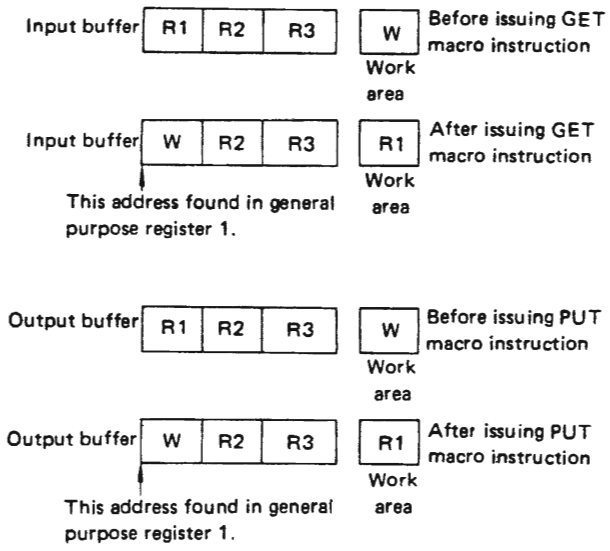


Fig. 4.27 GET/PUT macro instruction with substitute mode

Two processing modes of the PUTX macro instruction can be used in conjunction with a GET locate macro instruction.

- Update mode
  In this mode, records from a data set are read by the GET macro instruction, processed and then written out to the same record position from which it was read. This mode is convenient for updating data sets.
- Output mode
  In this mode, records are read by the GET macro instruction from one data set, updated and transferred to a new data set.

BSAM macro instructions

BSAM is equipped with the following macro instructions to perform data management in block units:

- READ
  Block read-in processing.
- WRITE
  Block write-out processing.
- CHECK
  Check the end of I/O operations of the READ/WRITE macro instructions.
- BSP
  Backspace one block on the magnetic tape or direct access volume being processed.
- CNTRL
  Control of magnetic tape and unit record devices.
- POINT
  Position to specified blocks on a volume.
- NOTE
  Requests the address of the present block on the volume.

4.7.3 Optional Functions of the Sequential Access Method

Functions independent of I/O device type

1) Parallel input processing

QSAM parallel input processing may be used to process two or more input data sets concurrently. This eliminates the need for issuing a separate GET macro instruction to each DCB processed. The get routine for parallel input processing selects a DCB with a ready record and then transfers control to the normal get routine. If there is no DCB with a ready record, a multiple WAIT macro instruction is issued.

Parallel input processing provides a logical input record from a queue of data sets with equal priority. The function supports QSAM with input processing, simple buffering, locate or move mode, and fixed, variable, or undefined length records. Spanned records, track-overflow records, dummy data sets, and SYSIN data sets are not supported.

Parallel GET function has been designed to shorten the waiting time during input processing in the case where requests for input from multiple data sets are issued simultaneously. This is accomplished by extracting records from the data set whose input is completed earlier.

2) Chained scheduling function

This function accelerates the input/output operations required for a data set. Prior to completing an earlier I/O request, a series of separate I/O requests are issued to the computer system as one continuous operation. The I/O performance is improved by reduction in both the CPU time and the channel start/stop time required to transfer data within virtual storage.

3) User's totalling function

When creating or processing a data set with user labels, control totals for each volume of the data set may be developed and stored in the user labels. For example, a control total that was accumulated as the data set was created can be stored in the user label and later compared with the total accumulated during processing of the volume. User totaling assists the programmer by synchronizing the control data the user creates with records physically written on a volume. For an output data set without user labels, the programmer can also develop a control total that will be available to the end-of-volume routine.

I/O device dependent functions

1) Direct search function

This function is used to accelerate input operations required for a data set on DASD. Direct search reads in the requested record and the count field of the

second record. This allows the operation to get the next record directly, along with the count field of the following record.

2) Check function (write operation)
After a record is transferred from main to secondary storage, the system reads the stored record (without transferring data) and verifies that the record was written correctly.

3) Track overflow function
The amount of used space on a volume can be reduced by using the track overflow option in the DD statement or the DCB macro instruction associated with a data set. A block that does not fit on the track is partially written on that track and continued on the next track.

4) RPS function (rotational position sensing)
This function minimizes the channel time spent waiting for a revolution of the volume on a direct access unit.

Function unique to magnetic tape units
Data management provides the following special services:

1) Read backward function
This function is used to read a magnetic tape backwards. In the case of BSAM, several macro instructions have been provided to position the tape (POINT/CNTRL macro instructions).

2) Padding function
The possibility of error is higher when writing blocks on a magnetic tape of very small length. To reduce this source of error, a function is available to add padding characters to short blocks and write them onto magnetic tape.

3) Handling of format-D code data sets
This function is used to convert/reconvert format-D code data sets (convert format-D codes to standard EBCDIC codes and vice-versa). A prefix is added to the data blocks showing the attributes of the block when the format-D code is used for data sets.

4) Handling of 7-track magnetic tape units
This function is used to perform code conversion between EBCDIC and BCD data sets. The conversion is done by the magnetic tape unit itself by utilizing a special hardware mechanism.

5) OS IV/F2 checkpoint record bypass function
Magnetic tape data sets which have been prepared by a OS IV/F2 system may contain checkpoint records which are of no use under OS IV/F4 data management. Therefore, in data management, OS IV/F2 checkpoint records must be tested and bypassed before proceeding with data set processing.

6) REF function (reduce error recovery)
This function is utilized in the data management program to control the repetitions of error recovery processing when an I/O error has occurred during data transmission. If an error exit is specified in the DCB macro instruction, the user routine is given control in the event of an uncorrectable error.

Functions unique to the line printer unit
The functions of data management related to control of the line printer unit are as follows:

1) Line feed control
This function control the advance of the line printer sheets.

2) Universal character set mechanism
This mechanism enables the processing program to specify the character type which the line printer can print.

3) Form control buffer (FCB)
Form Control Buffer has the software specifications which control skipping of lines when printing is being done on a line printer unit such as the F650D or F651D/E.

Functions Unique to the Card Punch Unit
Data management provides the following special services:

1) Specifiable Punching Modes
- EBCDIC mode.
- Digit binary mode.
- Simple EBCDIC mode.

2) Specifiable stacker
The output stacker is program-selectable.

Special data seets
Special services are provided for the following data sets:

1) SYSIN/SYSOUT data sets
SYSIN/SYSOUT data sets are input/output streams which are stored on an intermediate storage device until there is a convenient time for processing.

2) Dummy data sets
By specifying DUMMY in a DD statement it is possible to carry out a simulated (dummy) processing of data sets. Dummy processing is the apparent processing performed when an I/O request is received from the processing program, however no processing is done.

3) Handling of F690D card punch unit
This device is equipped with card read, card punch,

and card print functions. The user can select any of these functions by specifying appropriate parameters in the DCB of the program. Any combination of the three functions may be selected.

4.7.4 Volume Switching

If multiple volume data sets are required, automatic volume switching is accomplished by the end-of-volume routine.

When an end of volume condition occurs, the present volume is automatically closed and the open operation for the new volume is performed.

Volume switching situations

- Magnetic tape volume
  During input when EOF is detected and during output when the present volume is filled (EOV is detected).
- Direct access volume
  During input when present volume has to be changed because the EOD is detected on a multi-volume data set and during output when unused space on the present volumes makes it impossible to further allocate space.

Processing of current volume when swtching

- Processing of data set label.
- Check of block count (for input processing of magnetic tape only).
- Processing of user's label.
- Position processing (for magnetic tape only).
- Space allocation (for direct access volume only).

Processing of new volume when switching

- Check of whether correct volume has been mounted or not.
- Processing of data set label.
- Processing of user's label.
- End of volume processing for the old volume.

4.8 PARTITIONED DATA SET AND PARTITIONED ACCESS METHOD

A sequential data set can be processed (read, write, etc.) by specifying its name and volume. All processing must include open and close operations. When processing multiple sequential data sets, open and close processing is performed separately for each data set. Moreover, since the devices for processing of these data sets may be different for each, processing efficiency can be adversely affected. A degree of efficiency can be achieved if similar types of data sets with the same attributes and same general purpose are assembled at one location on the same device so that a common open/close processing for the entire group of data sets can be performed collectively. The concept of a partitioned data set has been developed in order to permit this procedure. The partitioned access method is used to access to a partitioned data set.

Constraints imposed on a partitioned data set are given in Table 4.12.

Table 4.12 Constraints imposed on a partitioned data set

Medium	Direct access volume
Volume configuration	Single volume
Data-set concatenation	Max. up to 255 data sets
User's label	Not permitted

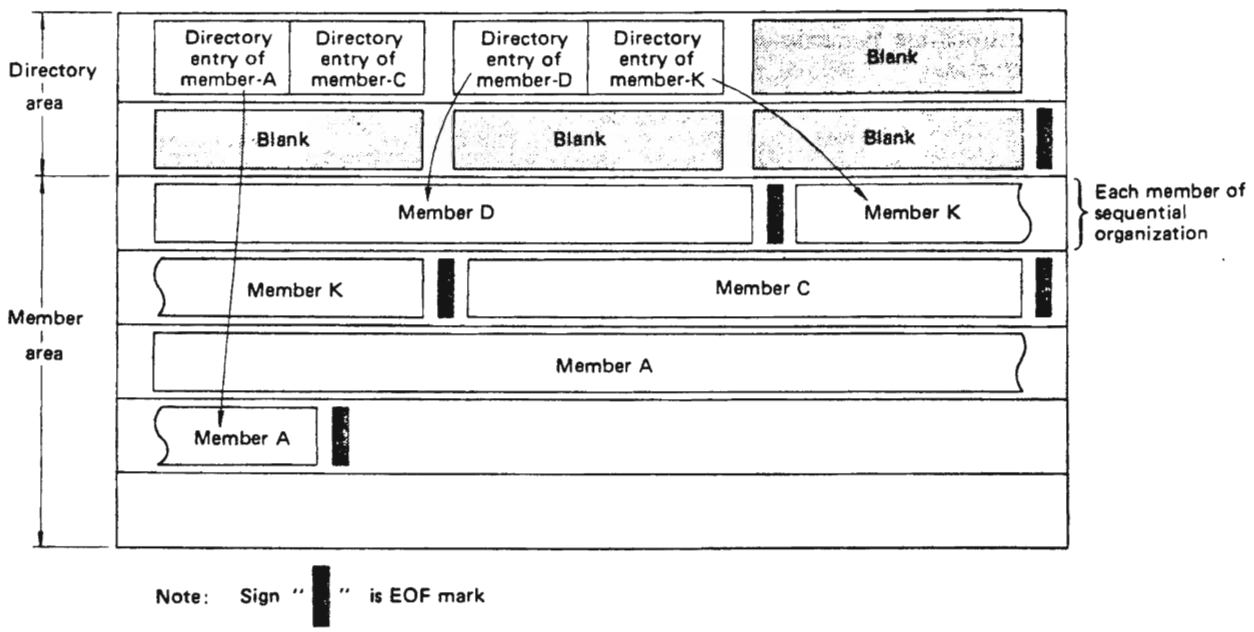


Fig. 4.28 Structure of a partitioned data set

4.8.1 Partitioned Data Set Structure

A partitioned data set residing on a direct access volume is made up of independent data groups and data management information related to those data groups. Each of these data groups is called a member; the organization of each member is sequential. This method of organization is designed to process similar sequential data sets in a uniform manner. The space allocated to the members is called the **member area.** The portion of the data set which contains information regarding each member is called the **directory.**

Fig. 4.28 shows the structure of a partitioned data set.

Structure of the directory

The directory is a data set composed of several data blocks which are called directory blocks. Each directory block is composed of several records called directory entries containing information about members. An EOF mark follows the last physical directory block. The structure of directory is shown in Fig. 4.29.

Some of the information about members, which is contained in the directory entries, includes member names, initial address (relative track address) of members, etc. Every directory block contains the maximum number of directory entries in alphabetical order of the member names.

4.8.2 Partitioned Access Method

The partitioned access method has been developed for processing partitioned data sets (BPAM: basic partitioned access method). BPAM permits recording and deleting members from the directory and reading or writing of records within members. Several macro instructions are available in BPAM, such as, macro instructions to retrieve and store information in the directory and macro instructions to retrieve/store member records from the member area.

Macro instructions unique to partitioned data sets

Before a member of a partitioned data set can be processed, the volume must be positioned to the member. This is done randomly using information from the directory. The following macro instructions are available to support this operation.

- BLDL
  Retrives information from the directory.
- FIND
  Positions the volume to specified members.
- STOW
  Performs inserting, deleting, updating and replacing of member information in the directory.

Before issuing any of the above macro instructions, the partitioned data set must be opened, and previous I/O requests regarding that partitioned data set must be complete.

Macro instructions to retrieve/store records

I/O macro instructions for retrieving and storing records within a member are identical to the macro instructions used in the basic sequential access (BSAM).

- READ
  Reads blocks within a member.
- WRITE
  Writes blocks within a member.
- CHECK
  Makes the processing wait for the end of an I/O operation.
- NOTE
  Acquires the relative track address of the previous block transferred.
- POINT
  Points to a block.

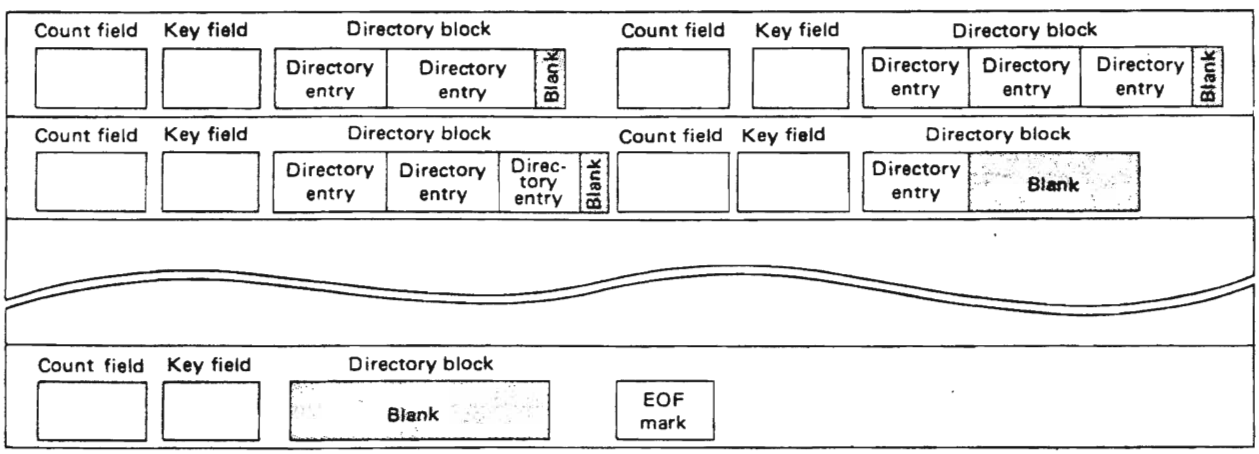


Fig. 4.29 Structure of directory area

4.9 DIRECT DATA SET AND DIRECT ACCESS METHOD

Often a data processing application requires the processing of records in an order different than the physical sequence in the data set. In a direct data set, there is a relationship between a control number of identification of each record and its location on the direct access volume. The relationship permits access to a record without an index search. Table 4.13 specifies some attributes of direct data sets.

Table 4.13 Attributes of direct data sets

Medium	Direct access volume
Volume configuration	Multi-volume possible
Data-set concatenation	Not possible
User's label	Allowed (however only on the 1st volume)

4.9.1 Direct Data Set Structure

A direct data set must reside on a direct access volume. It contains records arranged according to the user's specifications, similarly, the accessing sequence is arbitrary depending on the user's needs. A direct data set can be processed sequentially using either the queued access technique or the basic access technique.

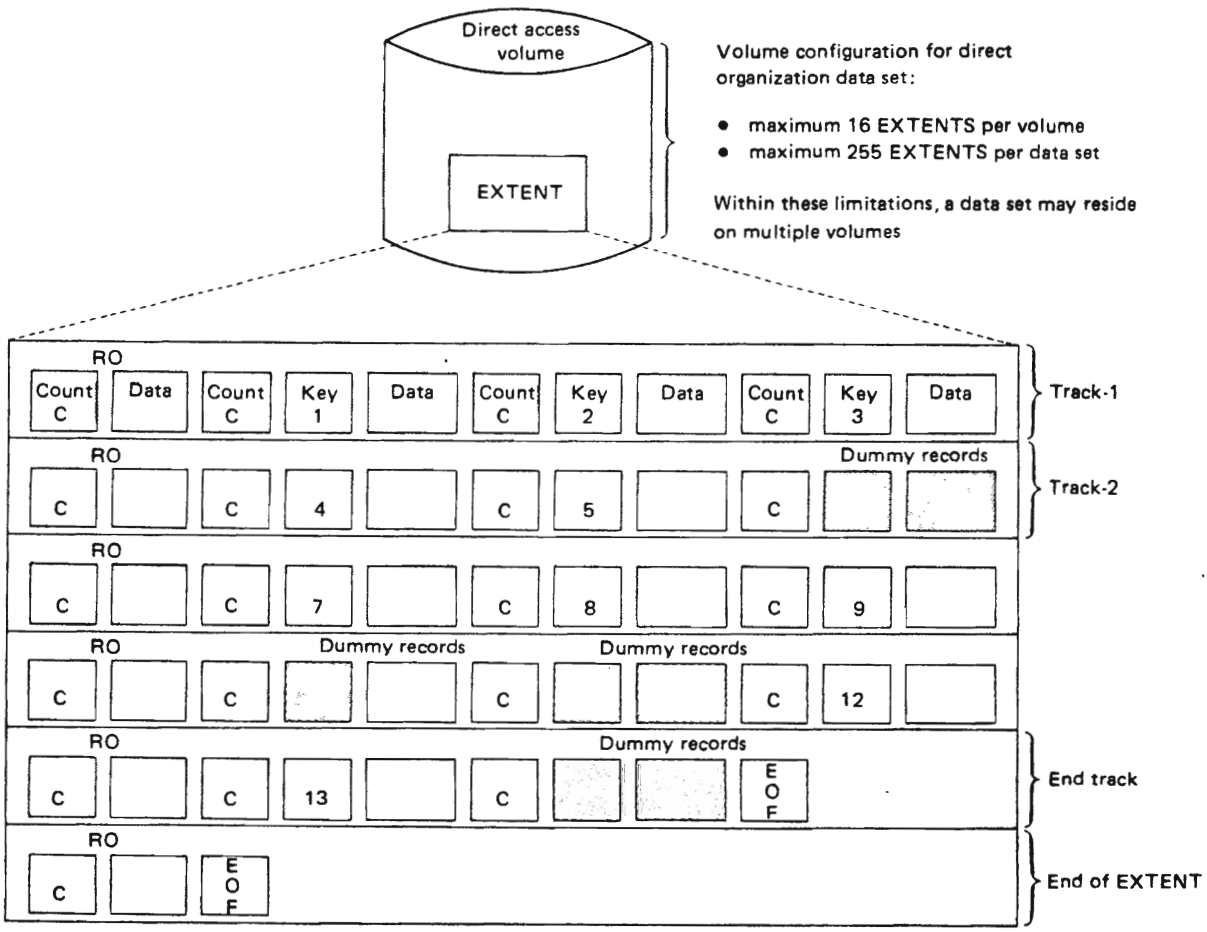
- A direct data set may consist of either a data field only or a key-field and data field.
- Records in the data field may be fixed length, variable length, or undefined length records.
- In the case of a direct data set with fixed length records space must be allocated on the basis of the range of keys rather than the number of records.

If format-F records with keys are being written, the key of each record can be used to identify the record. For example, a data set with keys ranging from 0 to 4999 should be allocated space for 5000 records. Each key relates directly to a location that can be referred to as a relative record number. The main disadvantage of this type of organization is that records may not exist for many of the keys even though space has been reserved for them.

The fact that a direct data set with variable or undefined record length has the maximum blocksize reserved for each block also results in unused space in a file.

Overall structure of direct data set

Fig. 4.30 shows the overall structure of a direct data set.



Records corresponding to keys 1–5, 7–9, 12, and 13 contain data, while records corresponding to keys 6, 10, 11, and 14 are dummy records.

Fig. 4.30 Structure of direct data set

Record format

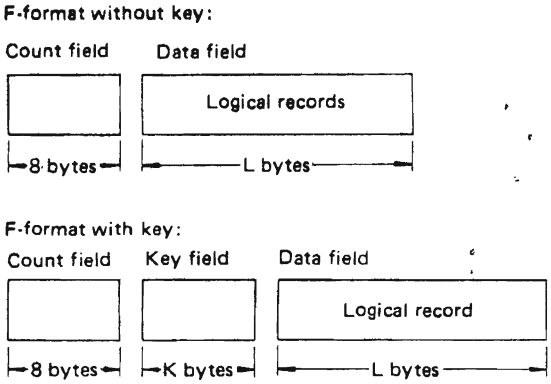
Record formats for direct data sets are the following types:

- F-format
  fixed length record format.
- V-format
  variable length record format.
- U-format
  undefined length record format.

These formats may include the blocking (B), track overflow (T), and spanned (S) options. The most common formats are the following:

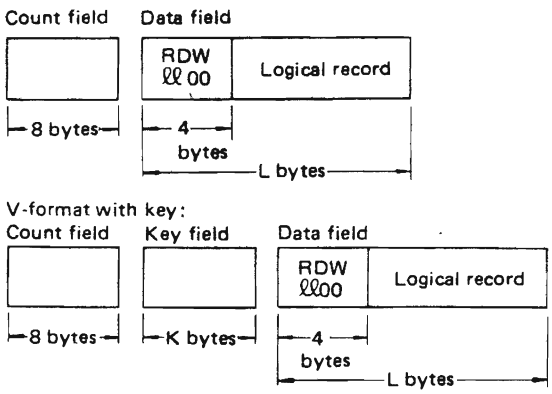
- F/FT
- V/VS/VBS
- U

Format F, V, and U are shown in Fig. 4.31, 4.32, and 4.33 respectively. Fig. 4.34 depicts the VBS Blocked variable length spanned) record format.



L: Record length, block length (1 record/1 block)
K: Key length

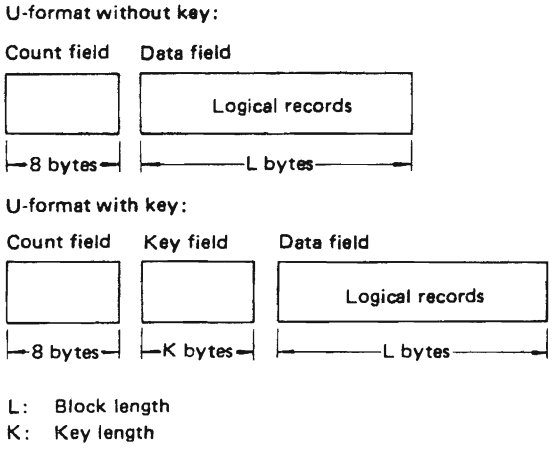
Fig. 4.31 F-format for direct data set



L: Record length, block length (1 record/1 block)
K: Key length
RDW: Record descriptor word

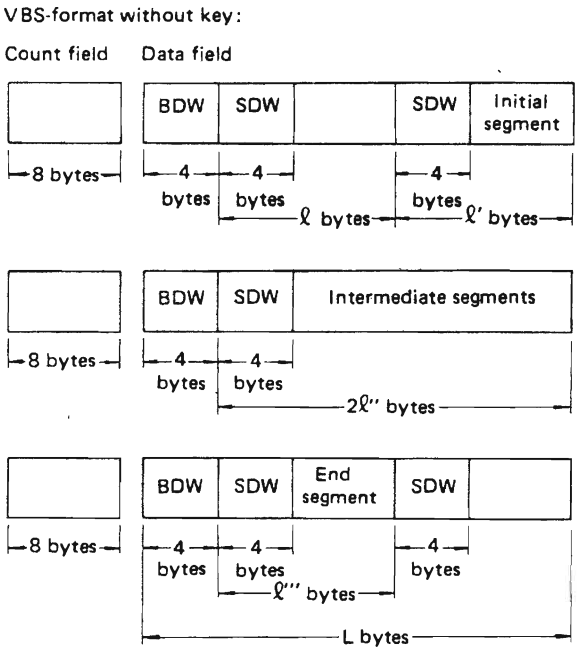
- 4-byte record descriptor word (RDW) exists at the beginning of each logical record.
- Initial 2 bytes of RDW (ℓ ℓ) . . . Length of the logical record (L)
- Next 2 bytes of RDW . . . Binary zeroes.

Fig. 4.32 V-format for direct data sets



L: Block length
K: Key length

Fig. 4.33 U-format for direct data set



BDW: Block descriptor word
- Initial 2 bytes of BDW . . . . Length of its segment (L bytes in this diagram)
- Next 2 bytes . . . . . Binary zeroes

SDW: Segment descriptor word
- Initial 2 bytes of SDW . . . . . Length of its segment (ℓ', ℓ'', ℓ''' bytes in this diagram)
- Next 1 bytes . . . . . Segment control code
  - Lower 2 bits
    - 00 . . . . Logical record which cannot be divided into segments
    - 01 . . . . Initial segment
    - 10 . . . . End segment
    - 11 . . . . Intermediate segment
- Next 1 byte . . . . . Binary zeroes

Fig. 4.34 VBS-format for direct data set

4.9.2 Direct access method

The BSAM DCB macro instruction must be used with the WRITE macro instruction to create a direct data set.

There are two methods of accessing a direct data set: by using the block address, or by the extended search technique using a block address as the starting point. There are three ways to specify a block address:

- Relative block address (nnn)
  In this method, the relative position of a block (or record) in the data set is specified with 3 binary bytes (nnn). This method can only be used in data sets with fixed length records.
- Relative track address (TTR)
  In this case, the block address is specified by a relative track number in the data set. That is 2 bytes for the track (TT) followed by 1 byte (R) for the record number (or block number).
- Absolute track address (MBBCCHHR)
  In this method, the address of desired block in the data set is specified as an 8 byte absolute track address (MBBCCHHR).

The primary macro instructions which can be used in the direct access method are the following:

- READ
  Reads in a block.
- WRITE
  Writes out a block.
- CHECK
  Tests for the completion of an I/O operation.

4.9.3 Optional Functions Utilized in the Direct Access Method

A large portion of the data set processing must be performed by the user when direct data sets are used. However, there are advantages to selecting an optimum layout for data sets. The following three optional functions are available to improve the processing efficiency.

Feed-back option
This option specifies that the system provide the address of the record requested by a READ or WRITE macro instruction. The feed-back option has 2 forms: one for creation of a direct data set, and the other for updating or making additions to a direct data set.

- Feed-back option for creating direct data sets.
  The function can be used only for creating direct data sets with variable length spanned records (VS). With the variable length spanned record format the address of the next block to be written is computed in advance. This task is facilitated by utilizing the feed-back option; address of the next block is returned to the area specified by the user in the form of a relative track address (TTR).
- Feed-back option for updating or making additions to direct data sets.
  In this case, feed-back is returned in the format of the addressing scheme used in the problem program (an actual or a relative address). When a WRITE or RELEX macro instruction is issued, the system will assume that the addressing scheme used for the WRITE or RELEX macro instruction is in the same format as the addressing scheme used for feed-back in the READ macro instruction.

Extended search option
Usually the READ or WRITE macro instruction is used for searching 1 track for a block or dummy block in a direct data set. By utilizing this option, multiple tracks can be searched by issuing a single READ or WRITE macro instruction. When searching for a dummy record, only one WRITE instruction is needed.

Dynamic buffering option
The user can either create and manage I/O buffers himself or use the capabilities of the data management program. Data management handles the management of I/O buffers through the use of the dynamic buffering option. With it the user can efficiently utilize buffers secured by the data management program as I/O areas.

After issuing a READ macro instruction with the dynamic buffering option, the system, before performing the I/O operation, retrieves a buffer from the buffer pool and then reads the I/O data into it. When a subsequent WRITE macro instruction is issued, the retrieved buffer is considered an output area for updating or adding data to the data set. The system then returns that buffer to the buffer pool after having written out its contents.

In case the user wants to return the buffer without issuing a WRITE macro instruction, he can do it by issuing FREEDBUF macro instruction.

Exclusive control option
This function has been incorporated for executing automatic exclusive control over a direct data set, after the user has issued READ or WRITE macro instructions to it, while adopting direct access methods.

The exclusive control option prevents a job step from accessing a block during updating by another job step. Therefore the block contents cannot be read until the current program issues a WRITE or RELEX macro instruction to the block.

4.10 CONCATENATION OF DATA SETS

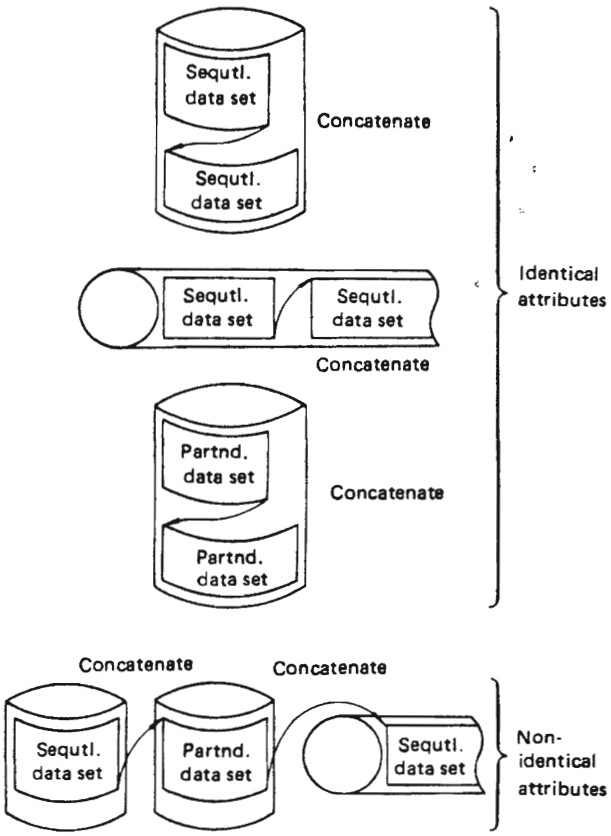
While processing similar multiple partitioned data sets or sequential data sets, it is sometimes convenient to treat them collectively as one data set. This capability exists in the data management program by using the data set concatenation function. Multiple

data sets which are processed consecutively by this function are called concatenated data sets. Concatenation may be of 2 types: concatenation of data sets with identical attributes and concatenation of data sets with different attributes.

In the former, the identical attributes of the concatenated sets are:

- record format.
- record length/block length.
- data set organization.
- device containing the data sets (DASD only).

Open or close processing for each data set is not necessary while concatenating data sets of identical attributes. When concatenating data sets of different attributes open and close processing is required. Fig. 4.35 shows an example of concatenation.



Note: In this case, only one member of the partitioned data set is concatenated.

Fig. 4.35 Concatenation of data sets

Only sequential data sets and partitioned data sets can be concatenated. And, as seen in Fig. 4.35, both concatenation of identical and non-identical sequential and partitioned data sets is possible.

Concatenation limits are specified as follows:

- Sequential data sets only; maximum 255.
- Partitioned data sets only; maximum up to 255 extents, where every data set contains not more than 16 extents. In other words, if every data set contained only 1 exetent, it would be possible to concatenate up to 255 data sets.
- Mixed sequential and partitioned data sets; since sequential access is used to process concatenated data sets, it is possible to concatenate up to 255 data sets. With this type of concatenation, because only 1 member of each partitioned data set may be concatenated, they are treated in the same way as sequential data sets. Positioning of the concatenated members of the partitioned data sets is done by the data management program.

An example of the job control statements necessary for concatenating data sets with non-identical attributes are given below. Here are three data sets; two with DSN equal to AAA and MTA and the third on card input. AAA resides on disk and MTA on tape.

```
//TRCF  DD DSN=AAA, UNIT=F479B, VOL=SER=000001,
//          DISP=(OLD, DELETE)
//       DD DSN=MTA, DISP=OLD, UNIT=F610K,
//          LABEL=(,NL)
//       DD *
         :
      Card data
         :
/*
```

4.11 SHARING AND EXCLUSIVE CONTROL OF DATA SETS

Generally, when multiple user's seek to process a data set, it is possible to adopt either the exclusive control method or sharing method.

Exclusive control of a data set

To use exclusive control, the DISP parameter of the DD statement is specified as OLD, NEW or MOD. The initiator, at the time of initiating the specified data set in the DD statement, keeps other requests waiting until the present user has completed processing of the data set.

Sharing of a data set

To permit sharing of a data set, the DISP parameter of the DD statement is specified as SHR. Multiple users can then simultaneously process the same data set.

Usually exclusive control is adopted when large numbers of requests for the same data set would exist if the data set were shared. Macro instructions

necessary in exclusive control are as given below:
- ENQ macro instruction.
- DEQ macro instruction.
- RESERVE macro instruction.

Additional reference information on these macro instructions is presented in Section 8.7.

4.11.1 Shared Use or Exclusive Control of a Data Set by Tasks Within a Job Step

This section describes the shared use or exclusive control of a data set by the multiple tasks which are generated, say, by issuing ATTACH macro instructions (see Section 8.3 Task Management) during a job step. However, the user must carefully understand the operation of tasks and the method of shared use of a DCB before he can access shared resources.

Shared use of a data set by one DCB
Fig. 4.36 shows an example of one DCB that is used commonly by two tasks for a shared data set.

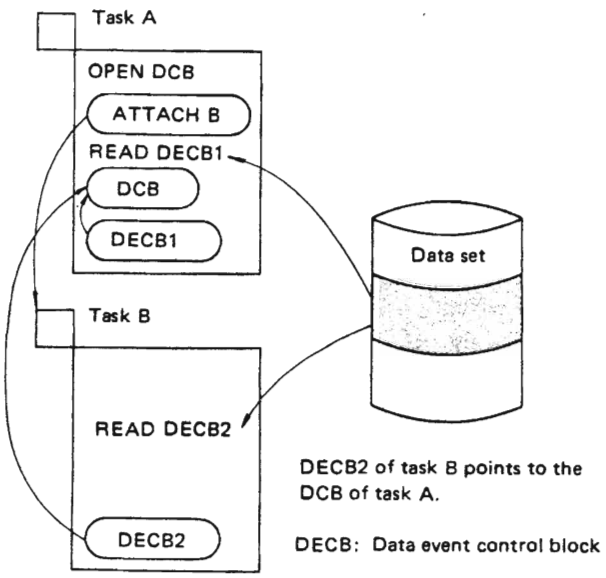


Fig. 4.36 Shared use of a data set with one DCB

In the figure:
- Task A initially opens a DCB.
- Next task B is generated by issuing an ATTACH macro instruction.
- READ macro instruction containing DECB1 designation is issued.
- During task B a READ macro instruction containing DECB2 designation is issued.
- DECB2 of task B points to the DCB of task A.

In this case, the user is generally required to execute exclusive control of the data set by issuing ENQ/DEQ macro instructions everytime he access the data set. Table 4.14 gives the macro instructions exclusive witth one DCB.

Table 4.14 Exclusive control macro instructions with one DCB

Access method / Processing mode	BSAM	QSAM	BPAM	BDAM
INPUT	ENQ READ CHECK DEQ	ENQ GET DEQ	ENQ READ CHECK DEQ	Not necessary
OUTPUT	ENQ WRITE CHECK DEQ	ENQ PUT/PUTX DEQ	ENQ WRITE CHECK STOW DEQ	Not necessary
UPDATE	ENQ READ CHECK WRITE CHECK DEQ	ENQ GET PUTX DEQ	ENQ READ CHECK WRITE CHECK STOW DEQ	ENQ READ CHECK WRITE CHECK DEQ

Shared use of data sets by multiple DCBs
An example of a data set being shared by more than one task and using multiple DCBs is shown in Fig. 4.37.

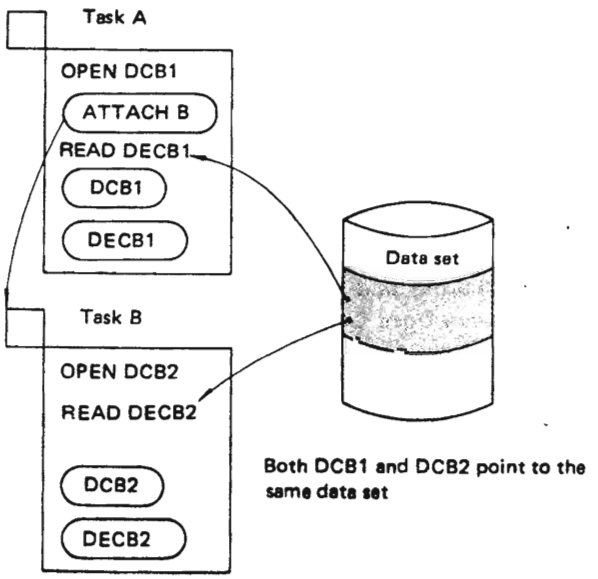


Fig. 4.37 Shared use of data sets by several DCBs

In the figure:

- Task A initially opens DCB1.
- Next, Task B is generated by issuing ATTACH macro instruction.
- READ macro instruction containing DECB1 designation is issued.
- During Task B, DCB2 is opened.
- Next, READ macro instruction containing DECB2 designation is issued.
- DECB1 of Task A points to DCB1, while DECB2 of Task B points to DCB2.
- And both DCB1 and DCB2 point out the same data set.

By utilizing this method, a user can freely access any record in the data set in the INPUT processing mode, without any concern of other users. In the OUTPUT processing mode, however, shared use of a data set is generally not possible. When the access method is BDAM, since the system automatically controls the sequence of accesses to records, shared-use is possible, both in INPUT as well as OUTPUT processing mode. In the UPDATE mode, the shared-use of a data set, except in the case of QSAM, is possible but is the user's responsibility. Table 4.15 gives the macro instructions for exclusive control when multiple DCBs are used.

Table 4.15 Exclusive control macro instructions with multiple DCBs

Access method / Processing mode	BSAM	QSAM	BPAM	BDAM*
INPUT	Not necessary	Not necessary	Not necessary	Not necessary
OUTPUT	Not feasible	Not feasible	Not feasible	Not feasible
UPDATE	ENQ READ CHECK WRITE CHECK DEQ	Not feasible	ENQ READ CHECK WRITE CHECK STOW DEQ	ENQ READ CHECK WRITE CHECK DEQ

* The user exercises exclusive control of data set himself without making use of BDAM's exclusive control function.

4.11.2 Sharing and Exclusive Control of a Data Set by Tasks from Different Jobs

This section describes the shared use and exclusive control of a data set by tasks which are included in the job steps of different jobs in a multijob system. For this type of processing, the user must code SHR in the DISP parameter on the DD statement for the data set.

To use exclusive control, the user must issue ENQ/DEQ macro instructions as given below and also specify the appropriate parameter in the DD statement for the data set. Fig. 4.16 gives the macro instructions for using exclusive control.

Table 4.16 Exclusive control units

Access method / Processing mode	BSAM	QSAM	BPAM	BDAM*
INPUT	Not necessary	Not necessary	Not necessary	Not necessary
OUTPUT	Not feasible	Not feasible	Not feasible	Not necessary
UPDATE	ENQ READ CHECK WRITE CHECK DEQ	Not feasible	ENQ READ CHECK WRITE CHECK STOW DEQ	ENQ READ CHECK WRITE CHECK DEQ

* The user exercises exclusive control himself without making use of BDAM's exclusive control function.

4.11.3 Sharing and Exclusive Control of a Data Set by Multiple Systems

This section describes the shared use or exclusive control of a data set by 2 or more computer systems. The data set must reside on a shared DASD (shared direct access storage device). Shared DASD refers to one or more DASD which can be shared and accessed by multiple systems simultaneously.

For exclusive control, the user must utilize the RESERVE macro instruction (see Section 8.7 Management of Serially Reusable Resources) and DEQ macro instruction. That is, exclusive control is the responsibility of the user. The system cannot execute exclusive control (BDAM's exclusive control function also is not available for shared DASD). Fig. 4.38 presents the concept of shared DASD. Table 4.17 gives the macro instructions for exclusive control of a data set by multiple systems.

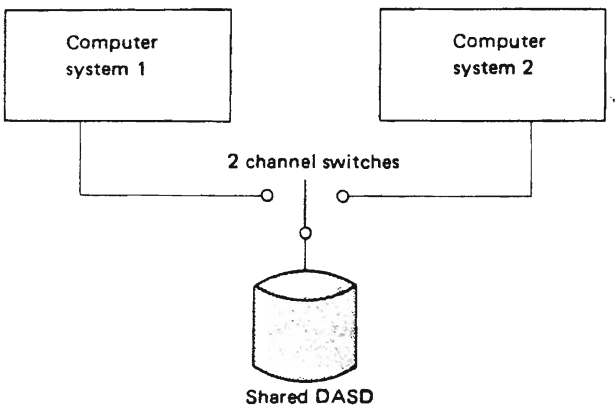


Fig. 4.38 Concept of shared DASD

Table 4.17 Exclusive control macro instructions

Processing mode \ Access method	BSAM	QSAM	BPAM	BDAM
INPUT	Not necessary	Not necessary	Not necessary	Not necessary
OUTPUT	Not feasible	Not feasible	Not feasible	Not feasible
UPDATE	RESERVE READ CHECK WRITE CHECK DEQ	Not feasible	RESERVE READ CHECK WRITE CHECK STOW DEQ	RESERVE READ CHECK WRITE CHECK DEQ

4.11.4 Deadlock from Exclusive Control

Deadlock is the nonoperational mutual wait state caused by competition among job steps for exclusive control of a data set. Fig. 4.39 illustrates the deadlock state.

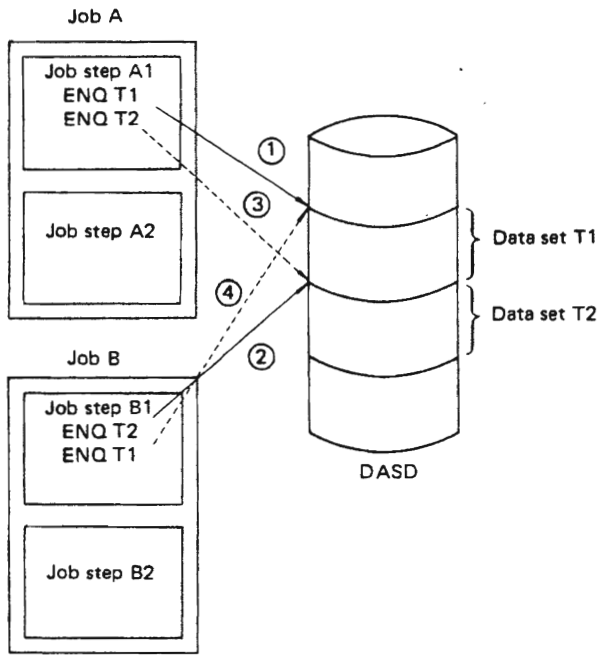


Fig. 4.39 Deadlock state

In the figure:

- Job step A1 locked data set T1 by issuing the ENQ macro instruction.
- Job step B1 locked data set T2 by issuing the ENQ macro instruction.
- Job step A1 issues the ENQ macro instruction to data set T2; however, it is kept waiting since T2 has been locked by B1.
- Job step B1 issued the ENQ macro instruction to data set T1; however, it is kept waiting since T1 has been locked by A1.

In the above case, both job steps A1 and B1 remain in the wait state, thus halting operation. This state is called the deadlock state. Users executing exclusive control (except for BDAM exclusive control), must be careful to avoid this type of deadlock.

4.12 SPACE MANAGEMENT

Space management is the function of managing requests for allocation or releasing of space on a direct access unit. Space management is performed by a control program called DADSM (direct access device space management). Below are some of the main functions of DADSM:

- space allocation for data sets;
- allocation of space extensions for data sets;
- releasing all space of a data set;
- releasing of unused space of data set.

4.12.1 Space Allocation

The request for allocation of space on a direct access volume may be made by parameters on the DD statement. The following type of parameters are available.

- SPACE parameter
  Specifies the general request for space allocation.
- SPLIT parameter
  Specifies the request for a split (divided) cylinder space allocation.
- SUBALLOC parameter
  Specifies the request for further suballocation of already allocated space.

An example of the SPACE parameter of the DD statement is shown in Fig. 4.40.

SPACE=({TRK / CYL / block length} (primary allocation [,secondary allocation] [,directory]) [,RLSE] [,CONTIG / ,MXIG / ,MLX] [,ROUND])

SPACE= (ABSTR, (primary allocation, relative track address in the volume [, directory]))

Fig. 4.40 DD statement parameters for requesting allocation of space

The unit of space allocation and space allocation method is described next by referring to the above subparameters.

General Space Allocation

The following are the units of space allocation:

- tracks.
- cylinders.
- block length (in bytes, however, the system, while actually allocating the space, allocates it in track units).
- Space in track units starting from the user's specified absolute track address (relative track address in the volume).

The methods of space allocation can be summarized as follows:

- CONTIG (Contiguous)
  With this parameter, available contiguous EXTENTS equal or larger than the requested space, are searched out and the requested space is allocated from them.
- MXIG (Maximum Contiguous)
  Here, available contiguous EXTENTS, greater than the size of the requested space are searched out, and from them, space from the EXTENT containing the maximum available space is allocated.
- ALX (All Extent)
  EXTENTS larger than the requested space are searched out. Up to 5 EXTENTS, starting from the largest available space, are allocated.
- No special space requirement specified:
  An EXTENT containing available space which can satisfy the need is searched out and allocated. However, if such an EXTENT cannot be found, space taken from up to 5 EXTENTS may be consecutively allocation.
- ABSTR (Absolute Track)
  The requested space, starting from the specified address (relative track address in the volume, except R0) is allocated.

An example of the various methods of space allocation is given in Fig. 4.41. The figure indicates the before and after space allocation corresponding to the given DD statement.

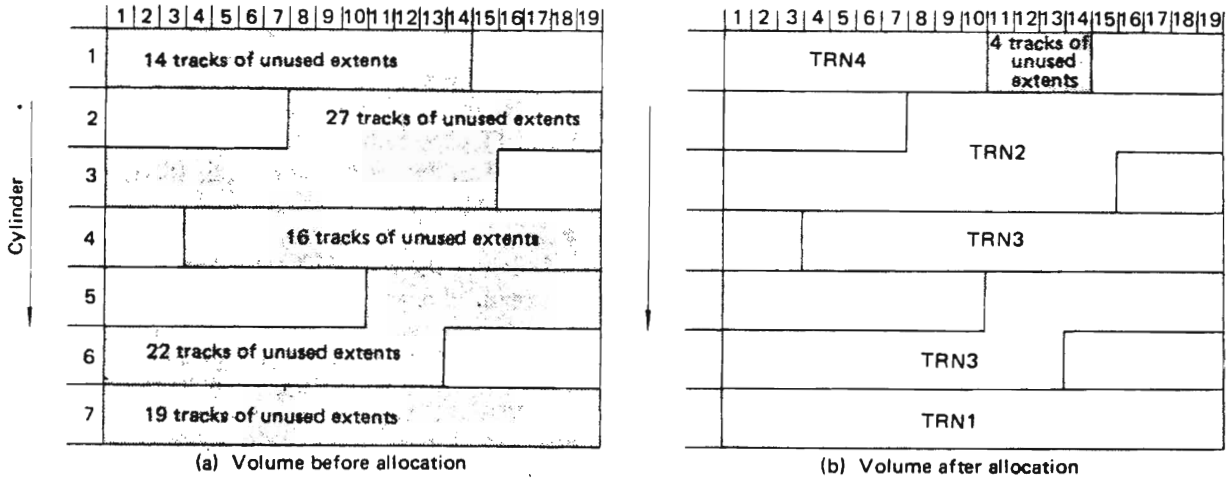
Allocation of split cylinder

This method allocates cylinder space from the same volume for shared use by a number of data sets, which are used in the same job step, thereby minimizing movement of the access arm. The following are the units for allocation of split cylinder.

- cylinders.
- block length in bytes (the system allocates space after conversion into cylinder).

When requesting split cylinder space the required space for the first data set may be specified exactly, and space for subsequent data sets may be specified as a percentage of the first. In response to this request, the system searches out an available EXTENT, equal to or larger than the requested space, and allocates the required space from it. Eventually, the allocated cylinders are divided between the data sets in the specified ratio.

The split cylinder space allocation (SPLIT parameter) cannot be made for partitioned data sets.




```
//SAT1  DD  DSN=TRN1, SPACE=(CYL, 1, CONTIG),.....
//SAT2  DD  DSN=TRN2, SPACE=(CYL, 1, MIXG),.....
//SAT3  DD  DSN=TRN3, SPACE=(TRK, 15,, ALX),.....
//SAT4  DD  DSN=TRN4, SPACE=(TRK, 10),.....
```

With direct access volume as shown in the figure (a), specifications through DD statements (c) will cause the volume to contain data as shown in the figure (b).

Fig. 4.41 Example of various methods of space allocation

The following DD statements are an example of a split cylinder request.

```
//STA1 DD  DSN=MAS1,
//            SPLIT=(5,CYL,8),...
//STA2 DD  DSN=TRN1, SPLIT=7,...
//STA3 DD  DSN=TRN2, SPLIT=5,...
```

The subsequent space allocation in response to the above request is depicted in Fig. 4.42.

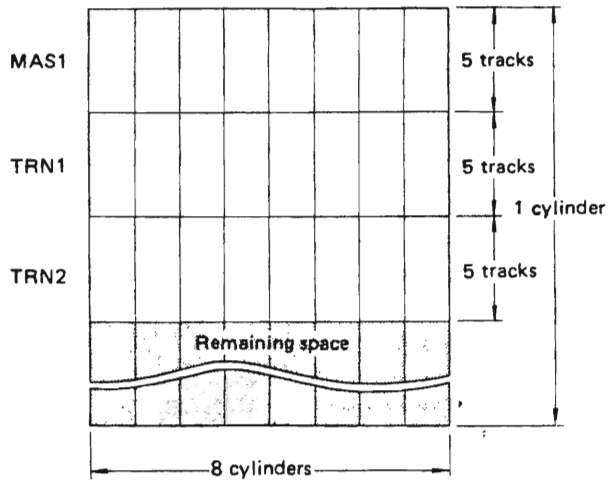


Fig. 4.42 Result of a split cylinder allocation

Suballocation

Suballocation of space is performed for multiple data sets in the available contiguous space on the same vvolume. With this method, the total space necessary for a job is requested collectively. Thus, every step in that job eventually requests for suballocation of already allocated space.

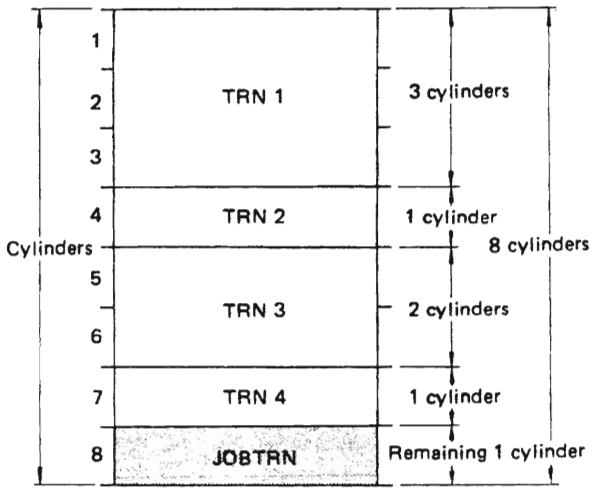
Since the space is continuous, improved access efficiency can be expected. Where optimum access efficiency is essential, this method will be found convenient. Units of space suballocation are the same as those which can be specified in the SPACE parameter (except ABSTR).

Suballocation can be specified in the SUBALLOC parameter of the DD statement and reference made to the DD statement of the first data set for which the space was secured. Therefore, space, starting with the first suballocated data set is allocated and any remaining space is allocated to the first data set (making the collective request). An example of suballocation is illustrated in Fig. 4.43.

```
//JJ1 JOB CLASS=A,...
//STEP1 EXEC PGM=PPTRN1
//STA1 DD DSN=JOBTRN, SPACE=(CYL, 8,, CONTIG),...
//STA2 DD DSN=TRN1, SUBALLOC=(CYL, 3,, STA1),...
//STA3 DD DSN=TRN2, SUBALLOC=(CYL, 1,, STA1),...
//STAP2 EXEC PGM=PP, TRN2
//STA4 DD DSN=TRN3, SUBALLOC=(CYL, 2, STEP1
          STA1),...
//STA5 DD DSN=TRN4, SUBALLOC=(CYL, 1, STEP1
          STA1),...
```

(a) JCL used to request suballocation

Space allocation performed (b) in response to the DD statements (a)



(b) Resulting suballocation of space

Fig. 4.43 Example of suballocation of space

4.12.2 Space Extension

If the primary space allocated to a data set runs short, and secondary space has been designated, DADSM automatically secures space extension. DADSM can manage a data set extending on multiple volumes. When this is done DADSM secures consecutive space extensions. The space extension onto another volume becomes necessary either when the unused space on the initially allocated volume is short or when the total EXTENTS of the data set exceed 16.

Specification for additional space can be done through SPACE, SPLIT, or SUBALLOC parameters.

Unit of additional space

The unit of additional space can be cylinders, tracks, or block length (bytes) depending on the initially used unit of space allocation.

Relationship between DISP parameter of DD statement and additional space.

- When the DISP parameter is specified as NEW, the space is extended directly by the amount specified in the secondary space allocation.
- When the DISP parameter is MOD or OLD, secondary space which was specified at the time of the initial space allocation or the secondary space specified in the current DD statement becomes the extension space. The latter type of secondary space allocation is effective only for the job step in which it is specified.

Method of space extension

For data sets or split cylinders, unused space is allocated on that volume equivalent to the secondary EXTENTS, which have been designated for that data set only. Thus, the additional cylinders do not get split between different data sets.

For suballocation data sets and all ordinary data sets, the method utilized is the same as that used at the time of primary space allocation.

4.12.3 Releasing of Unused Space

Data management provides for releasing unused space so that any unused space in the allocated space of a data set at the time of the close operation is released to the available space on the volume. This enables efficient use of volume space without any unnecessary space allocation. Moreover, the units of released space are the same as the space unit which was used at the time of initial allocation. Table 4.18 gives the space releasing boundary for the different space allocation units.

Table 4.18 The space releasing boundary for the different space allocation units

Space allocation unit		Space releasing boundary
Cylinder (CYL)		May be released starting from the boundary of the cylinder next to the last stored data.
Track (TRK)		May be released starting from the boundary of the track next to the last stored data.
Block length	ROUND specified	May be released starting from the boundary of the cylinder next to the last stored data.
	ROUND not specified	May be released starting from the boundary of the track next to the last stored data.

The following DD statement is an example of using the RLSE parameter to release unused space.

```
//STA1 DD DSN=TRN1,
//        SPACE=(CYL,10,RLSE,CONTIG),...
```

Space is released upon closing the above data set (TRN1) in the following manner as shown in Fig. 4.44.

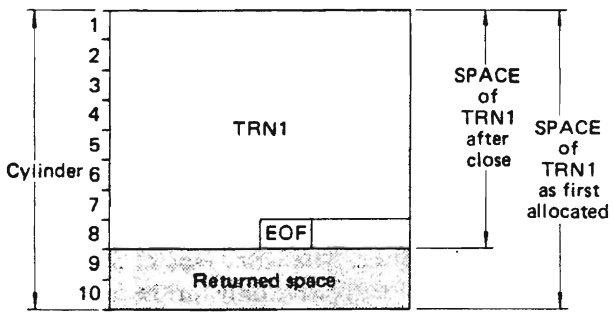


Fig. 4.44 Release of unused DASD1 space

4.13 CATALOG MANAGEMENT

Data sets residing on direct access or magnetic tape volumes can be managed by the user. However, the user must then be constantly aware of the data set names and their corresponding volume serial number.

Catalog management is the portion of data management designed to relieve the user from this tiresome work by performing a centralized and systematic type of data set management. In catalog management, data sets are managed by recording information (their names, volume serial number, device type) into a ledger where they are stored. This ledger, which is called the system catalog is also a data set itself. It resides on a direct access volume (system resident volume) with the name of "SYSCTLG". Parts of the catalog can exist on other volumes if necessary. Registering or deleting data sets from the system catalog is done by using the JSG PROGRAM utility program (see Part 3, Chapter 9, UTILITY) or by parameters in the job control statements.

By utilizing the catalog management function, any data set can be readily processed simply by specifying its name. Thus, the job control statements are simplified and time is saved during system operation.

4.13.1 Structure of System Catalog

The system catalog is recorded in the VTOC in the same way as other data sets. Its structure is the same as that of the directory of partitioned data sets. Each block is composed of several levels of index to manage the hierarchy of data set names. This type of block is called the index or index block.

The physical structure of the system catalog is designed for the catalog management functions; however the user need understand only its logical structure in order to take advantage of it.

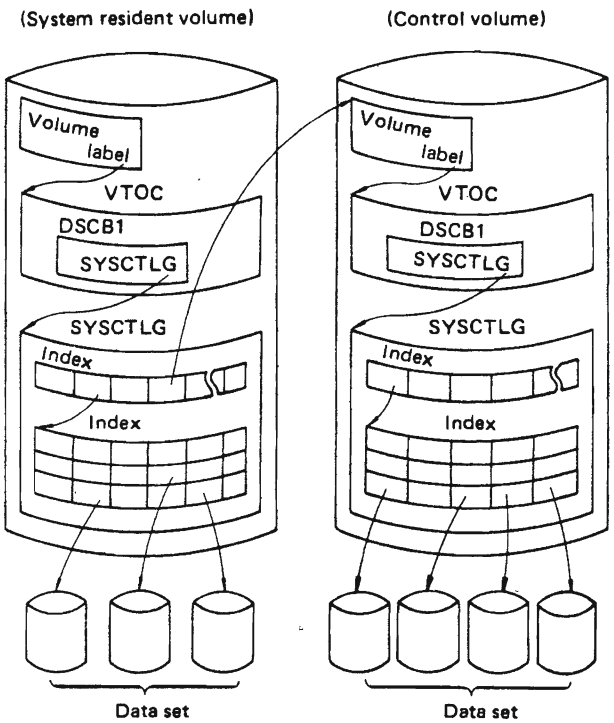


Fig. 4.45 Structure of the system catalog and its relationship with other control volumes

Although the system catalog generally exists on a system resident volume, part of it can be constructed on other types of volumes also. Any volume which contains part of the system catalog is called a control volume (CVOL). The control volumes must be kept logically linked with the system catalog contained on the system resident volume. It is also possible to link multiple control volumes mutually in succession. Fig. 4.45 illustrates the relationship between the system catalog on the system resident volume and the other control volumes.

Inside the system catalog are qualifier and simple names constituting the data set names. They are arranged in a uniform hierarchial structure with every qualifier pointing to a separate index block. The index block pointed to by the last qualifier is an index entry with volume information and simple names. An example of the qualified name structure is:

Here STAFF and SKILLS qualify the simple name MASTER.

As shown in Fig. 4.45, the system catalog is a data set of hierarchial physical structure. Another illustration of the scheme used to manage data sets inside a system catalog is shown in Fig. 4.46.

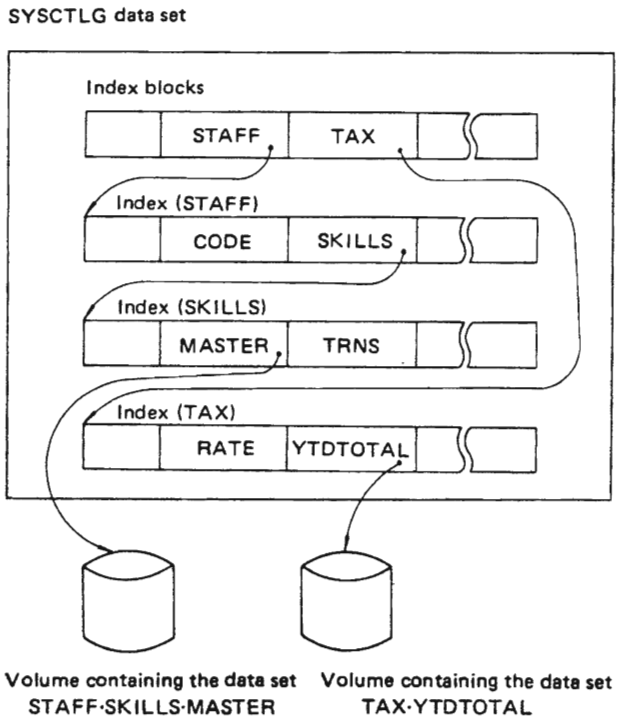


Fig. 4.46 System catalog and its data sets

4.13.2 Cataloging of General Data Sets

By cataloging of data sets is meant the recording information related to their location into the system catalog. This information includes the following items.

- data set name.
- volume serial number.
- device type.
- data set number (for magnetic tape volume only).

The user can catalog data sets (sequential/partitioned/ direct access data sets) into the system catalog using job control statements or a utility program.

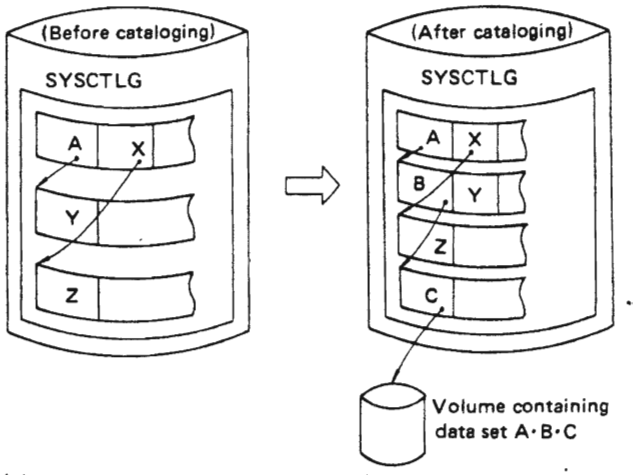
Cataloging of data sets by job control statements

If data sets being created are to be simultaneously cataloged, the user may do it by using a DD statement with the DISP parameter specified as CATLG. The user can also catalog data sets while accessing already existing data sets. However, the data sets will not actually be cataloged until the end of the job or job step.

Some examples of cataloging data sets are given in Fig. 4.47.

```
//STA1 DD  DSN=A·B·C, DISP=(NEW, CATLG), UNIT=F478B,
//            SPACE=(CYL, 3), VOL=SER=000001
```

(a) JCL for creating and cataloging a new data set



(b) Effect of cataloging data set A·B·C

```
//STA1 DD  DSN=A·B·C, DISP=(OLD, GATLG), UNIT=F478B,
//            VOL=SER=000001
```

(c) JCL for cataloging an old data set

```
//STA1 DD  DSN=A·B·C, DISP=OLD
```

(d) JCL for accessing a cataloged data set

Fig. 4.47 Creating and accessing cataloged data sets

Cataloging of Data Sets by Utility Program (JSGPROGRM)

With this method, information about existing data sets is given to the utility program, JSGPROGM, which performs the cataloging function.

4.13.3 Cataloging of Generation Data Sets

Some data sets are periodically created and updated; others are updated irregularly. It is convenient to catalog periodic data sets collectively and to update them at periodic intervals. For this purpose, data sets which are related with respect to the date of their creation are collectively called a **generation data group.** In catalog management, this group of data sets is managed under one name, the **generation data group name.** Moreover, data sets which belong to a generation data group are called generation data sets. Each of them being referred to by their group name and generation number. Generation numbers are of two types: absolute generation numbers and relative generation numbers.

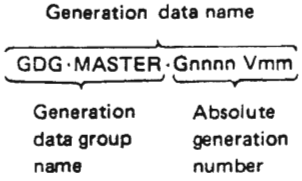


Fig. 4.48 Absolute generation number

Absolute generation numbers

An absolute generation number is used to identify a specific generation of a generation data group.

As shown in Fig. 4.48, an absolute generation number is expressed with 8 characters; 'nnnn' is the generation number expressed with any one of the 4 decimal digits from 0001−9999; and 'mm' is the version number expressed with any one of the 2 decimal digits 00−99.

The relationship between absolute generation numbers and the system catalog is shown in Fig. 4.49.

Relative generation numbers

As an alternative to using absolute generation and version numbers when cataloging or referring to a generation, a relative generation number can be used.

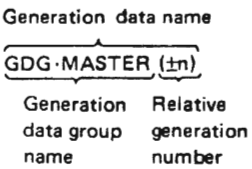


Fig. 4.50 Relative generation number

Any relative generation number can be expressed as shown in Fig. 4.50, where 'n' is the relative generation number expressed as any decimal number from 0−255. The relative generation number shows the relative position of the latest generation of data sets and is intended to be used for retrieving or

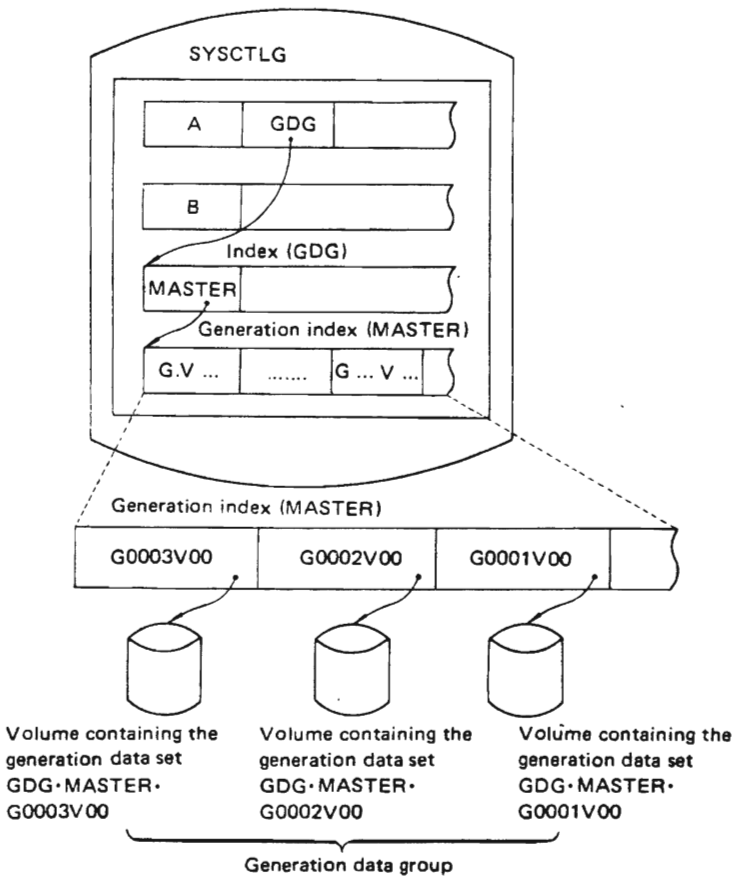


Fig. 4.49 Relationship between absolute generation numbers and system catalog

cataloging (described later) data sets. Fig. 4.51 shows the relationship between relative generation numbers and the system catalog.

In order to catalog a generation data group, the user must in advance, set up the generation data group name in the system catalog. For instance, in Fig. 4.49, and 4.51, the user must set index GDG and generation index MASTER prior to actual cataloging of the generation data sets. This can be done by using a utility program, JSGPROGM. The maximum number of generation data sets which can be cataloged in a generation index is 255 (the limit is specified in a parameter of JSGPROGM). The method of cataloging generation data sets is the same as cataloging of other data sets, that is, either by job control statements or by utility program (JSGPROGM).

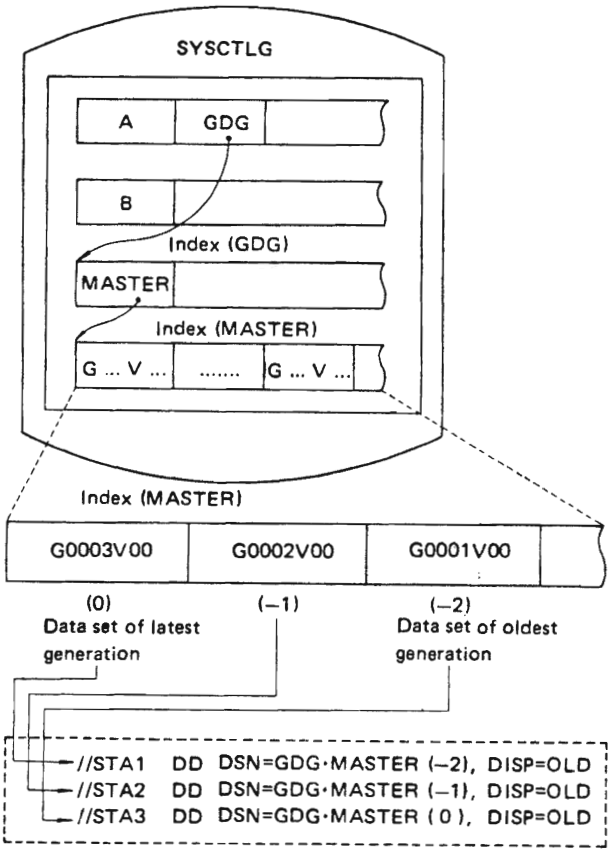


Fig. 4.51 Relationship between relative generation numbers and the system catalog

The cataloging of data sets by absolute generation numbers is depicted in Fig. 4.52.

Here, G0002V00 must be greater than the latest cataloged absolute generation number. If G0001V01 had been specified as the absolute generation number, and if G0001V00 already has been cataloged, it will get replaced by G0001V01.

The cataloging of data sets by relative generation numbers is depicted in Fig. 4.53.

In the example, given by Fig. 4.53, generation data set name GDG.MASTER. G0003V00 is assigned as the data set of latest generation, and is cataloged at the end of that job-step.

Upon cataloging the generation data set GDG.MASTER (+1) its relative generation number becomes (0), while the data set corresponding to the relative generation number (0) before cataloging (GDG.MASTER.G0002V00) becomes (−1). However, while extracting a generation data set cataloged as GDG.MASTER (+1) during the same job step, its relative generation number will have to be specified as (+1).

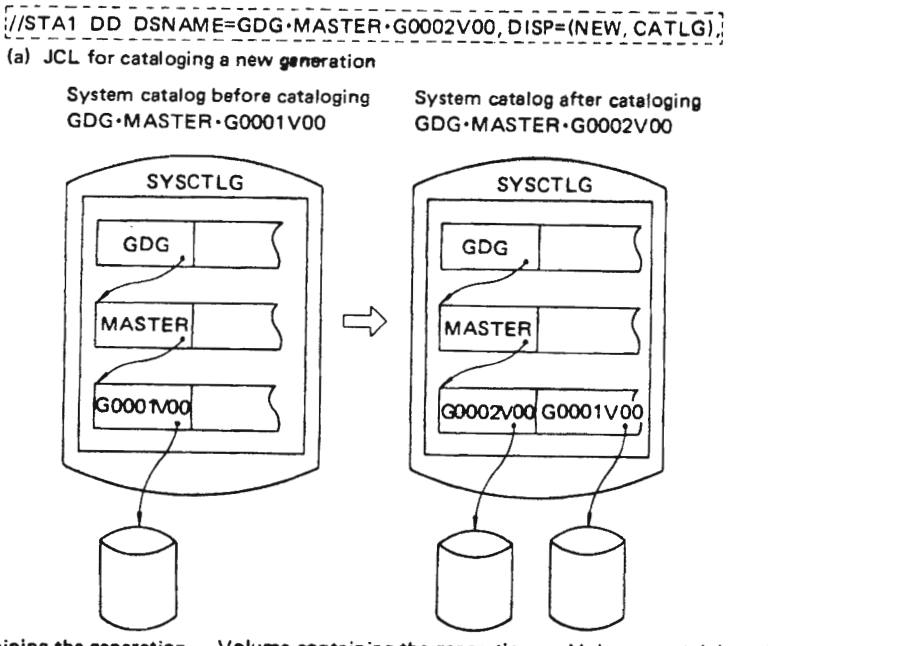
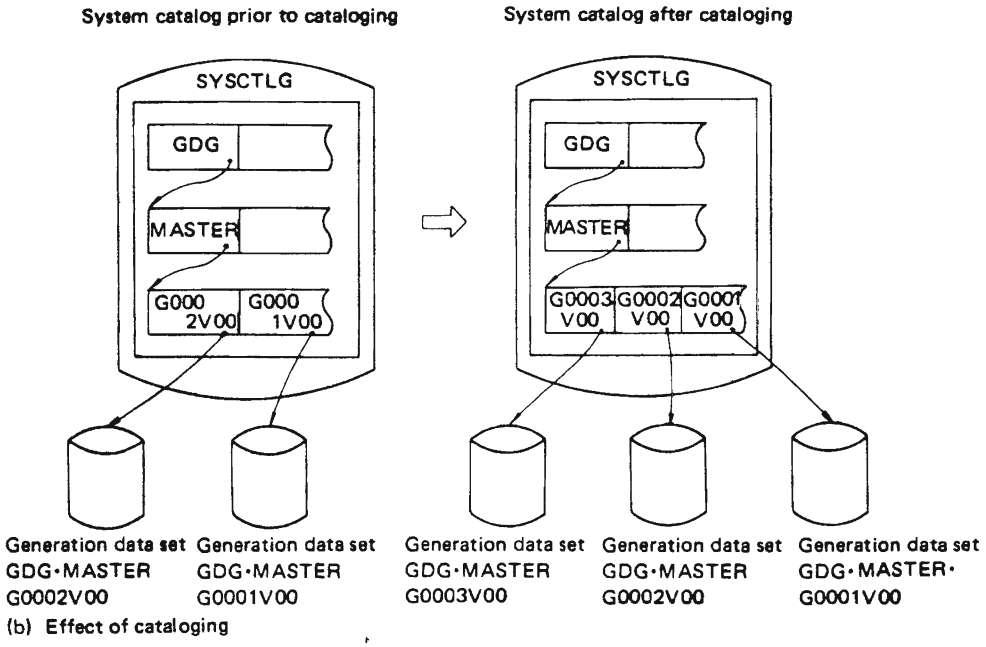


Fig. 4.52 Cataloging by absolute generation numbers

```
//STA1   DD  DSNAME=GDG·MASTER(-1), DISP=(NEW, CATLG)···
```

(a) JCL for cataloging a new generation



(b) Effect of cataloging

Fig. 4.53 Cataloging by relative generation numbers

4.13.4 Uncataloging Data Sets

Uncataloging a data set means deleting the corresponding volume information and any unnecessary index from the system catalog. Once a data set is uncataloged it cannot be accessed through catalog management until it is cataloged again.

Uncataloging can be done by two methods: with job control statements or with a utility program (JSGPROGM).

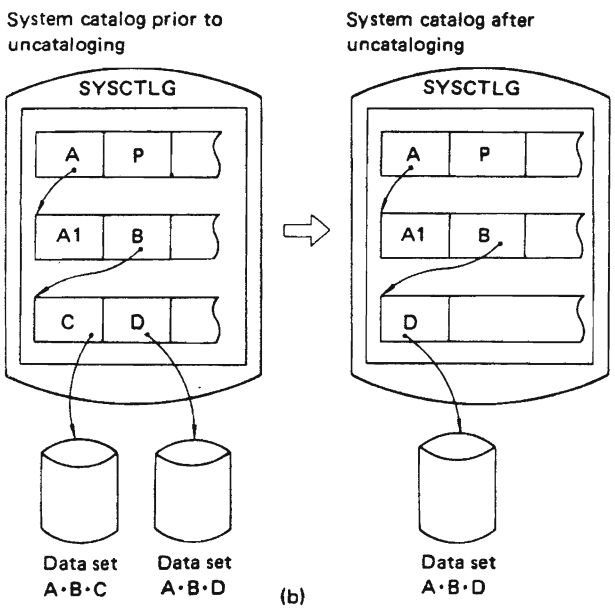
Uncataloging of data sets by job control statements
In this method, uncataloging is done by using a DD statement with DISP parameter specified as UNCATLG. For uncataloging a data set simultaneously with deleting it after completion of processing the user may specify DELETE for the DISP parameter on the DD statement. However, in this case, if the VOL parameter also has been specified in the DD statement, the data set will only be deleted and not uncataloged. In the former case, actual uncataloging is done at the end of that job or job step.

An example of the DD statement used to retrieve data set A. B. C. and uncatalog it at the end of the job or job step is given in Fig. 4.54.

Uncataloging of data sets by utility program (JSGPROGM)
The utility program, JSGPROGM, can be used to uncatalog data sets by removing the volume information and any unnecessary index information from the system catalog. Uncataloging is not complete until the utility finishees execution.

```
//STA1   DD  DSNAME=A·B·C, DISP=(OLD, UNCATLG)
```

(a)



(b)

Fig. 4.54 Uncataloging a data set

4.13.5 Uncataloging of Generation Data Sets

Generation data sets may be uncataloged using two methods: automatic uncataloging during catalog management and uncataloging by user's instruction.

Automatic uncataloging during catalog management
In catalog management, generation data sets are usually processed at periodic cycles. Since the max-

imum number of generation data sets which can be included in a generation data group is fixed (as specified by the utility program JSGPROGM . . . maximum of 255), if this number is exceeded, the oldest generation data set in the group is automatically uncataloged.

It is also possible to specify (via utility program JSGPROGM) that all the generation data sets in that group get automatically uncataloged upon exceeding the set number, and only the latest generation data set which is requesting to be cataloged at that time, gets cataloged.

The user can make either of these two arrangements by specifying a parameter in the utility program when the generation data group name is inserted in the system catalog. Thus, uncataloging will be done automatically without the user's giving explicit instructions.

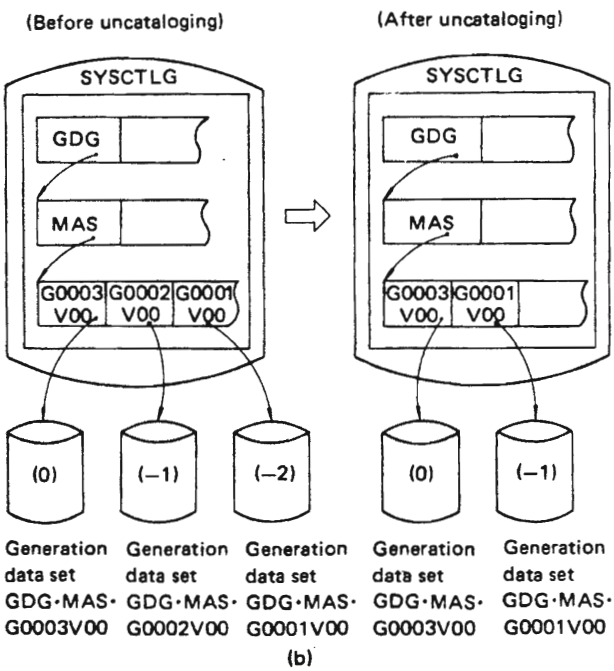
Uncataloging by user's instruction

If the user wishes to uncatalog a generation data set, he can do it by making use of job control statements during processing of a utility program (JSGPROGM) in the same way as for uncataloging other data sets.

An example of the uncataloging of a generation data set is given shown in Fig. 4.55.

```
//STA1 DD DSN=GDG·MAS(–1), DISP=(OLD, UNCATLG)
```

(a)



(b)

Fig. 4.55 Uncataloging a generation data set

4.14 PASSWORD PROTECTION

Password protection provides for the security of data sets stored on direct access and magnetic tape volumes. The user may designate passwords (each may have a maximum of eight characters) to prevent unauthorized access to his data set.

Password protection performs data set security checks to ascertain that only authorized users are accessing confidential data sets. The security check involves comparison, during open processing, of the password specified by the user. This is indicated by a protect-display byte in the tape label or data set control block.

Password protection can be requested when the data set is created by using the LABEL field of the DD statement. In addition to requesting password protection in the JCL, at least one record must be entered for each protected data set in a data set named PASSWORD that must be created on the system residence volume.

The password data set is maintained by the system through the use of the PROTECT macro instruction or the JSGPROGM utility program. The utility program is provided to add, delete, change, and print out entries in the password data set:

Two levels of protection options are available. The user can specify the options in the LABEL field of a DD statement with the following parameters:

- **PASSWORD** makes a data set unavailable for all types of processing until a correct password is entered.
- **NOPWREAD** indicates that a password is not required if only reading is to be performed; for other types of processing, such as deletion or output, a password is required.

If JSGPROGM or the PROTECT macro instruction is used to perform password protection for a data set on a direct access device, the system automatically updates the DSCB of the data set to indicate its protected status. However, the data set to which the password belongs must be mounted at that time.

Types of passwords and password protection method

The password can be entered in the password data set by using the utility program JSGPROGM. A data set may possess one or more passwords. The initial recorded password is called the control password; other passwords are called auxiliary passwords. The control password and auxiliary passwords should be specified for a data set at the same time.

Protect mode and protect display

The protect mode is the security level indicated in the password records of the password data set. The following types of protect mode can be specified.

- Protect password for all access transactions.
- Protect password for the write or delete operation, no password required for the read operation.

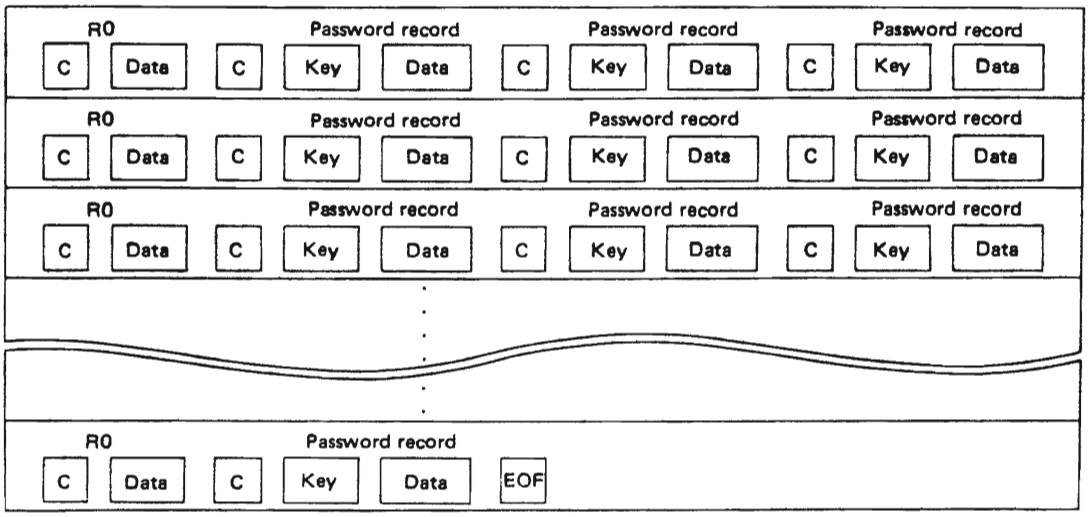


Fig. 4.56 Structure of the password data set

The protect display is the security indicator byte in the label of the data set to be protected. The user establishes the protect display by either the DD statement or the utility program JSGPROGM. During the opening operation, a data set which is protected by a password is checked for protect mode and password.

4.14.1 Structure of Password Data Set

The password data set (PASSWORD), which always exists on the system residence volume, is composed of password records. Each password record contains information about the data set which is to be protected. The PASSWORD data set organization is sequential and contains keys plus fixed length un blocked records. Fig. 4.56 shows the structure of a password data set.

Every password record consists of the fields given in Fig. 4.57.

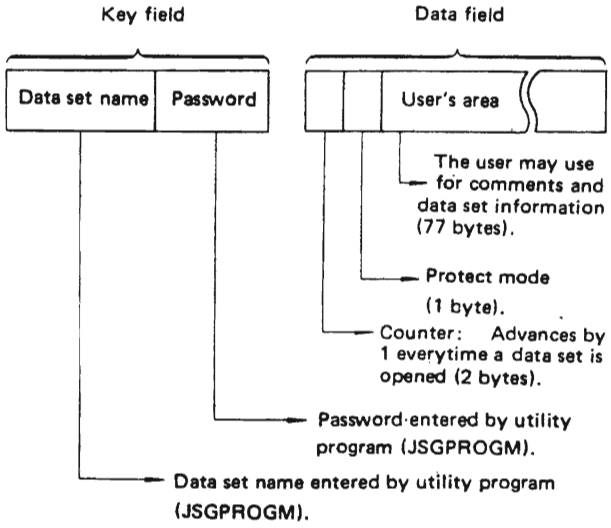


Fig. 4.57 Structure of password record

4.14.2 Password Protection and User's Identity Check

An outline of password protection and user's identity check is shown in Fig. 4.58.

Fig. 4.58 illustrates how password protection of a data set and user's identify check is performed during the opening of a data set TRN1. In the figure:

1. Initially, a check is made to determine whether the data set TRN1 (to be opened) is protected by a password or not. This is done by checking the protect-display contained in the data set label (DSCB1). If the display indicates protection is not necessary, open processing of the data set may be continued.
2. If the display indicates password protection, password management requests the user to input the password. It then checks whether the input password for the data set matchess with that of the password record of TRN1. If the input password is incorrect, it requests the user to reinput the password. If the input password is not correct a second time, the job is terminated.
3. If the input password matches, password management checks the password mode indicated in the password record of TRN1 (protect-display shown in DSCB1).

With these three steps password management determines whether or not the user is authorized to use the data set TRN1.

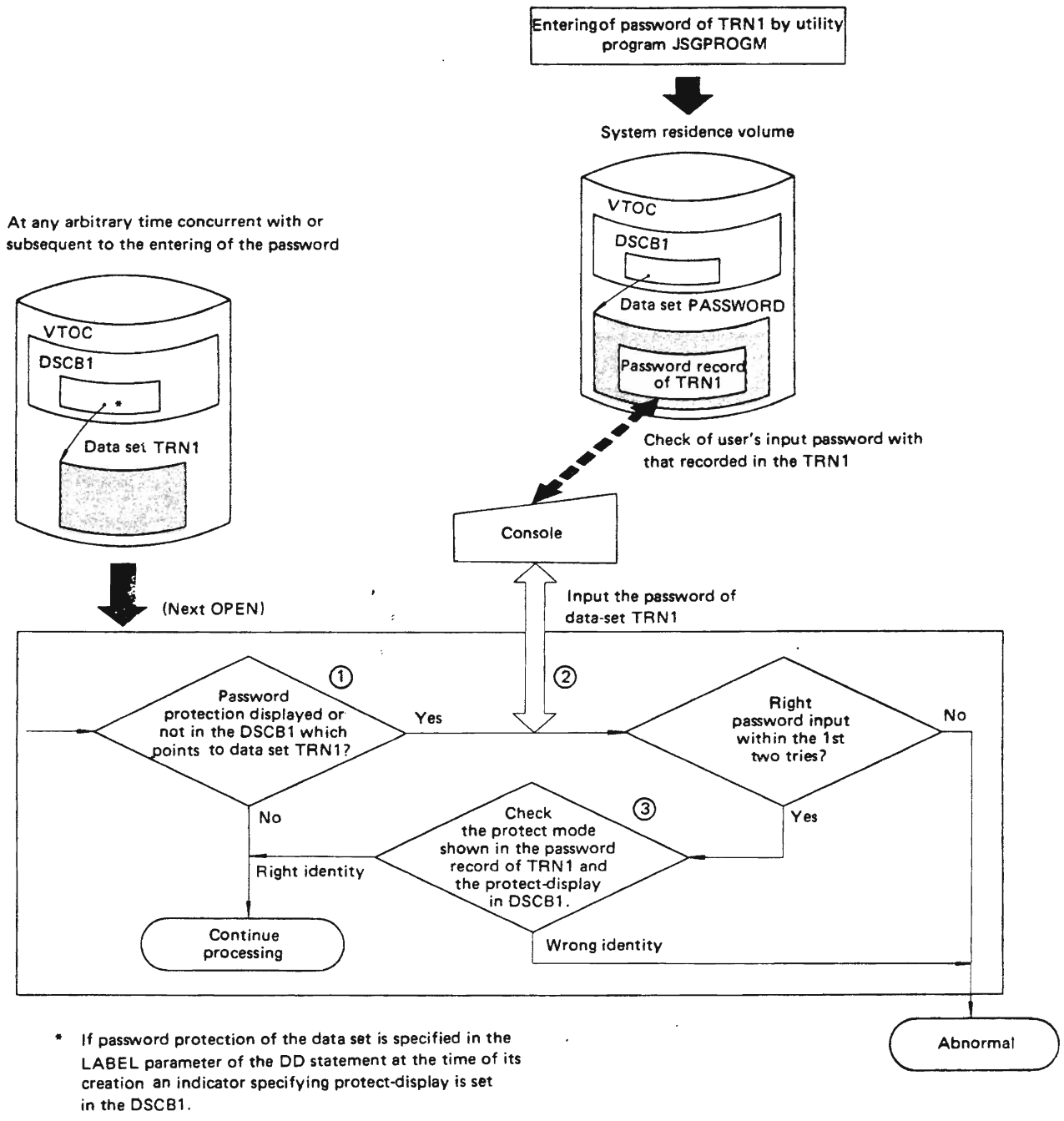


Fig. 4.58 Outline of password protection and user's identity check

Table 4.19 summarizes the method of password protection and user's identity check.

Protect-display indicated in the data set label is controlled by the LABEL parameter of the DD statement. The protect-mode indicated in the password record is controlled by the TYPE parameter of the ADD statement in the utility program JSGPROGM (see Part 3, Chapter 9 Utility).

4.15 EXCP

EXCP is a control program which enables the user to perform I/O operations with I/O devices which the data management program cannot handle or access data sets which cannot be accessed by usual methods. By utilizing EXCP, the user can directly control data sets to be transferred to I/O units according to their respective organization, as well as control the I/O units themselves. Fig. 4.59 shows the various relationships between EXCP users and EXCP.

4.15.1 EXCP—Usage and Processing

To use EXCP, the user must prepare some informa-

Table 4.19 Method of password protection and user's identity check

Protect-display inside data set label	Password input essential or not	Processing mode at the time of OPEN	Protect mode given in password record	Action
No password specification	Not essential			
Password	Essential	INPUT RDBACK	READ/WRITE *1	Continue processing
			READ *2	
		INOUT OUTPUT OUTIN UPDAT	READ/WRITE *1	
			READ *2	Abnormal end
No password read	Not essential	INPUT RDBACK		
	Essential	INOUT OUTPUT OUTIN UPDAT	READ/WRITE *1	Continue processing
			READ *2	Abnormal end

*1 Password protection during both READ and WRITE operations.
*2 Password protection is available for WRITE operation but not necessary for READ operation.

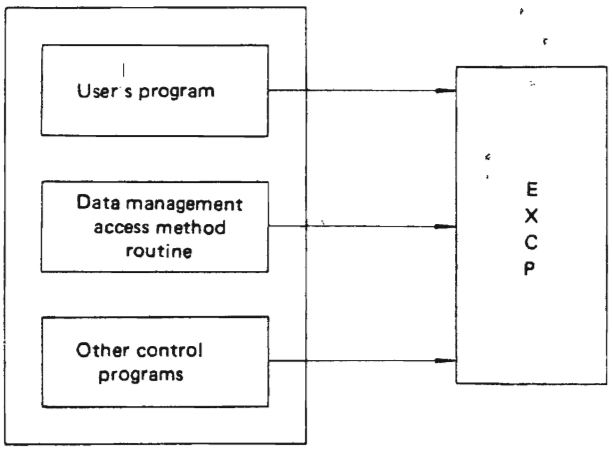


Fig. 4.59 EXCP users and EXCP

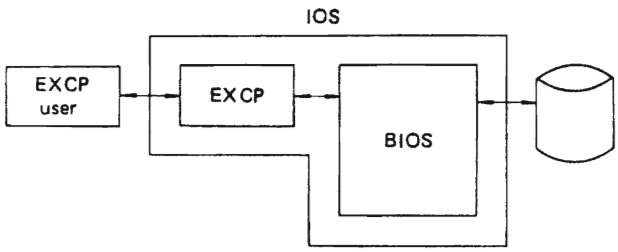


Fig. 4.60 Structure of IOS

tion for execution of the channel program in an IOB (I/O block). Subsequently, the user may issue an EXCP macro instruction to request data for input. Upon receiving this request, EXCP issues an SIO (start I/O) macro instruction to BIOS (basic I/O system) to request execution of the channel program. Channel program here means the program to perform I/O processing. It consists of a CCW (channel command word) which indicates the processing method. The channel program is interpreted by four separate operations before being executed.

BIOS, which performs all I/O operations, is the control program which forms the nucleus of IOS (I/O supervisor). Fig. 4.60 shows the relationship between EXCP, BIOS, and IOS.

Some precautions on the usage of EXCP: page folding must be avoided by providing in advance for the necessary page. The area can be:

- I/O buffer area.
- PCI appendage routine.
- Channel end appendage routine.
- CCW.

With EXCP the user can interrupt processing by EXCP. Also, if the EXCP user indicates a specific EXCP appendage routine, the EXCP transfers control to it during processing. EXCP appendage routines are described in the next section.

4.15.2 EXCP Appendage

In addition to the access methods provided by the operating system, an elementary access technique

called execute channel program (EXCP) is also provided. To utilize this technique, the user must establish a system for organizing, storing and retrieving data. Its primary advantage is the complete flexibility it allows in using the computer. However, if every attempt to give the system redundancy were made, its performance would be adversely effected.

The EXCP appendage interface has been provided for OS IV/F4 to control a part of EXCP operation by interrupting it during processing in order to perform several independent functions.

The user must provide an EXCP appendage routine, which links with the EXCP appendage interface of the OS IV/F4 system. This reenterable structure must be kept ready in LPA (link pack area). Because of its structure, the EXCP cannot acknowledge the EXCP appendage routine.

Types of EXCP appendage interface

Six types of EXCP appendage interface functions are available with EXCP:

- EOE (End of Extent).
- SIO (Start I/O).
- PGFX (Page Fix).
- PCI (Program Controlled Interruption).
- CE (Channel End).
- Abnormal end.

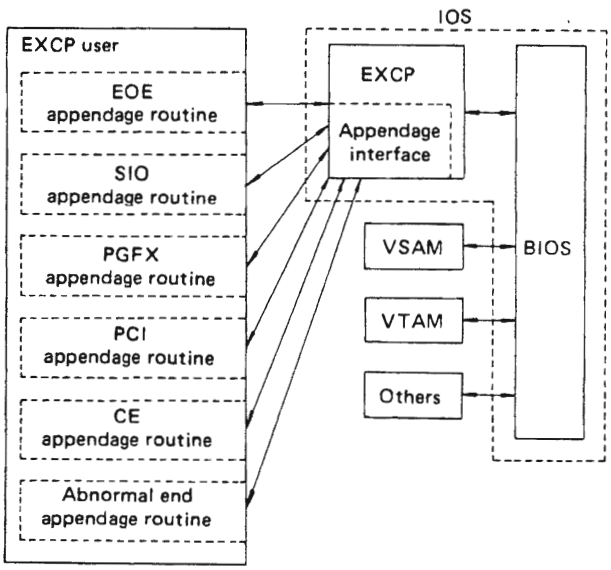


Fig. 4.61 Relationship between appendages and EXCP

The interrelationships of these appendages in OS IV/F4 are shown in Fig. 4.61.

Any appendage routines can be executed while protect key ZERO (see Section 8.2.3 Flow of Control) is set at supervisor mode.

Appendage routines must conform with the following rules:

- Any appendage routine should never issue a command which changes the system's operating mode (SVC command, LPSW command, etc).
- An appendage routine should not include logic which makes the processing wait for an end of I/O command execution.
- An appendage routine must not rewrite the area being used by the operating system.

EOE (end of extent) appendage interface

This appendage receives control at the time of EXTENT check by DASD. That is, in the case of DASD, EXCP checks the SEEK address before executing the SIO macro instruction. It then transfers control to this appendage if the SEEK address is found to be incorrect.

To handle such cases, the EXCP user must program this appendage routine to correct the erroneous address. Thus, when control returns to EXCP, it repeats the EXTENT check.

SIO (start I/O) appendage interface

This appendage receives control from EXCP before EXCP has started the transfer operations with the channel program and its data area.

Through this appendage the EXCP user can make changes in the channel program, make necessary changes in the addresses of data area, etc.

PGFX (page fix) appendage interface

This appendage is intended to notify the EXCP about which pages to fix by sending a fix list to it. This may be required to avoid page folds that may be generated while revising the channel program. The area in which the fix list is prepared is supplied by EXCP. Only after this area has been provided is control transferred to this appendage.

Here the EXCP user must structure a fix list which includes areas of CCW, buffers, PCI appendage, and CE appendage. There can be up to 10 EXTENT areas in the fix list. The EXCP fixes pages in the areas set by the PGFX appendage only.

PCI (program controlled interruption) appendage interface

This appendage receives control from the PCI exit routine which is set by EXCP during post-processing of BIOS. Here the EXCP user provides an appendage routine to make dynamic alterations in CCW and in the channel program.

The PCI interruption is generated when the PCI flag contained in CCW turns ON.

CE (channel end) appendage interface

This appendage receives control from EXCP in the following cases.

- When channel-end is reached after normal end of channel program.
- Upon recovering from an error by ERP (error recovery procedure) (see Section 7.2.7 ERP).

This does not refer to the channel-end of the channel program by retries, but to the end of ERP processing when the processing has temporarily been diverted to ERP.

The EXCP user provides an appendage routine taking into consideration unit exception cases (for example, if tape mark is detected upon issuing READ command to MT unit) and the processing of multivolumes.

Abnormal end appendage interface

This appendage receives control from EXCP in the following cases:

- If an error is generated during execution of the SIO macro instruction.
- If interruption caused by error status is generated at the start of I/O operation.
- If ERP cannot recover an error even after making several tries.
- When the seek address is not correct during EXTENT check (i.e., exceeds the limit) and EOF appendage has not been provided; or EOF appendage is provided but has made an abnormal end.

The user, by providing an appendage routine to perform a reinput/output operation in the case of different errors, can process his own independent errors also.

CHAPTER 5
VIRTUAL STORAGE ACCESS METHOD

5.1 OVERVIEW

The virtual storage access method utilizes direct access devices for sequential and direct processing. VSAM creates and maintains two types of data sets. One type is organized by a key field within each record and is called a **key sequenced data set.** Records are located using the key field and an index that points tp key fields and addresses of associated data. Key-sequenced records can also be located using their displacements from the beginning of the data set; each record displacement is called a **relative byte address (RBA).**

The other type of VSAM data set is organized by time of arrival of each record into the data set. This is called an **entry sequenced** data set. Records are located using the RBAs.

VSAM offers users two major advantages over sequential (SAM), indexed sequential (ISAM), and direct (DAM) access methods. One is its data organization techniques, which minimize data movement and provide device-independent data sets designed for long-term stability and for data base applications. The other is a set of service routines designed to facilitate data management; these routines initially define data sets, copy and print data sets, delete VSAM entries from a catalog, and provide full data set portability among small and large operating systems.

Conversion of data sets from ISAM or SAM format to VSAM format is another function of the service routines. An **ISAM interface program** is provided to map ISAM macro instructions into corresponding VSAM requests, so that most programs written using ISAM data sets can also be used with VSAM. In VSAM, access to data is controlled by assembler-language macro instructions.

In addition to major improvements, certain device-dependent calculations have been automated in VSAM tc minimize the programming effort required to change device types. For example, the calculation of optimum block size for a device is performed by OS IV/F4. VSAM also offers multiple levels of password protection and an exit for user written security routines to improve data set security.

For additional details on VSAM, the reader should consult the following publications:

- **FACOM OS IV/F4 VSAM Functions and Facilities.**
- **FACOM OS IV/F4 AMS Commands Reference Manual.**
- **FACOM OS IV/F4 VSAM Macro Instructions Reference Manual.**

5.1.1 VSAM Highlights

CPU and DASD efficiency

VSAM has many facilities to improve CPU efficiency, effective channel throughput, and job/transaction turnaround times, compared to SAM, ISAM, and DAM:

- skip-sequential processing, to eliminate continued index searches after locating the first record of a sequence.
- replicated track indexes, to reduce rotational delays on DASDs.
- long-term retention of index and data records in buffers to reduce the number of ovverall I/O operations.
- automatic insertion of empty space on tracks during creation of VSAM data sets, allowing for subsequent additions.
- reuse of space from deleted records, so that each track or block is compactly filled with active records.

Also, the following unique aspects of VSAM help it perform well:

- VSAM catalog is itself a VSAM data set.
- indexes are blocked.
- keys are **compressed** (sequences of keys reduced to minimum overall size).
- insertion of several new records within one track with a single track-long channel program.

Simple but useful modes of access

- Sequential access by key or RBA.

- Direct access by key or RBA.
- Skip-sequential access by key.

These modes can be intermixed freely after a data set has been opened, in contrast to ISAM. The user can switch from one mode to another as his application requires.

Integrated approach
All attributes of VSAM data sets and associated volumes are collected into the VSAM catalog, rather than being dispersed to non-VSAM catalogs, VTOCs, directories, etc. as in other access methods for DASDs. This approach helps simplify DD statements considerably.

Device independence
Each record in a VSAM data set is located by its byte address relative to the beginning of the data set. Hence, record and block numbers are not used to locate records. Since these numbers are quite device-dependent, VSAM achieves a much higher level of device independence than SAM, ISAM, etc.

Using the VSAM catalog and the AMS service program, it is simple to move VSAM data sets between installations.

Privacy and security
Passwords can be created at four different levels in VSAM, to safeguard individual fields and records as well as entire data sets. Users can be required to furnish authorization keys. Shared andd exclusive controls of data sets and individual records are fully supported.

Comprehensive service aids
In the **access method services** (AMS) service program are facilities for defining, changing, deleting, and displaying the VSAM catalog and selected

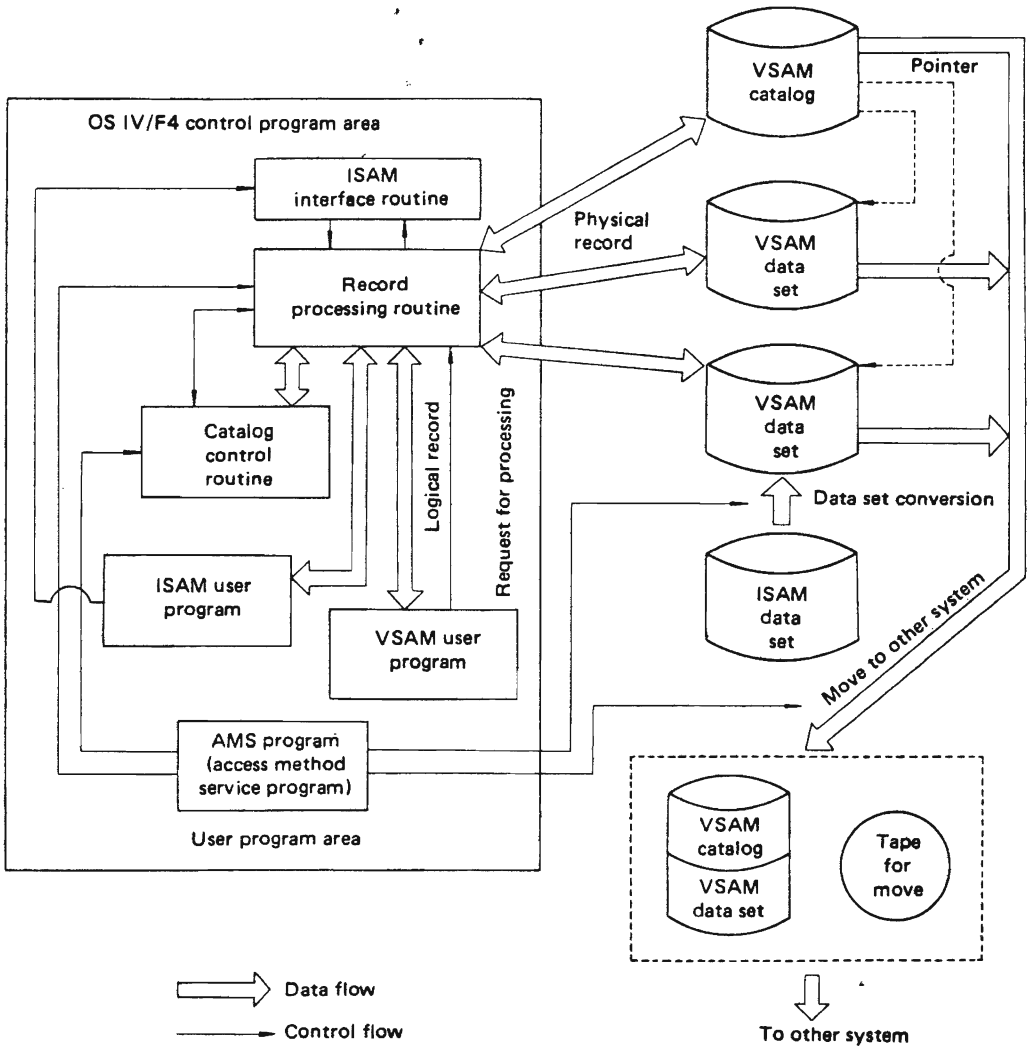


Fig. 5.1 Schematic diagram of VSAM

VSAM data sets. AMS will copy SAM, ISAM, and DAM data sets to VSAM format. It will create portable data sets, which contain relevant portions of the VSAM catalog as well as selected VSAM records. Finally, AMS will give powerful assistance to the user recovering VSAM data sets after a job failure or system failure.

5.1.2 VSAM Structure

Fig. 5.1 shows a schematic outline of VSAM. In the control program area (system and common areas of each address space) are VSAM routines for processing records, managing catalogs, and interfacing to programs with ISAM linkages. In the user area are the problem programs (using VSAM and/or ISAM linkages) and, as a distinct job step, AMS.

Record processing
These routines accept and validate requests for VSAM records from user programs, AMS, catalog control routines, and ISAM interface routines. These routines then process VSAM data sets and catalogs as requested.

Catalog control
This routine manages the VSAM catalog on a unified and centralized basis, handling the definition and any subsequent attribute modifications for each VSAM data set.

ISAM interface
This routine accepts and validates requests for ISAM-format records by user programs. It converts these requests to equivalent VSAM macro instructions, then services these requests appropriately. Hence, VSAM data sets can be created, accessed, and modified by assembler, COBOL, and PL/I programs originally written to use ISAM data sets.

VSAM user program
A VSAM user program requests processing of VSAM data sets with VSAM macro instructions.

ISAM user program
An ISAM user program can process VSAM data sets by linking to the ISAM interface routine. Previously any indexed sequential data sets must be converted into VSAM data sets using REPRO commands from AMS.

AMS (access method services)
AMS is a utility program for handling VSAM data sets. AMS is closely related to the catalog control routine. In other words, AMS processes data set information by referencing and changing catalog information. With AMS, a user can convert and indexed sequential data set to a VSAM data set. Also, he can move a VSAM catalog and/or VSAM data set to another OS IV/F4 system.

5.2 VSAM DATA SETS

5.2.1 Types

There are two types of VSAM data sets: key sequenced and entry sequenced. Each type may only be created on one or more DASDS.

Key sequenced data set
In a **key sequenced data set** (KSDS), records are arranged by ascending order of their keys. When a record is accessed by its key, an index is necessary; if a record is accessed by its relative byte address (RBA), an index is not used. Each index is a data set distinct from one containing data records. Key sequenced data sets automatically reuse space from which records have been deleted.

Entry sequenced data set
In an **entry sequenced data set** (ESDS), records are arranged in order of input; there is no index. All accesses are by RBA, and any new records are added to the end of the data set.

The principal differences between these two types are shown in Table 5.1.

Table 5.1 Differences between KSDS and ESDS

Type / Attribute	Key sequenced data set (KSDS)	Entry sequenced data sets (ESDS)
Sequence	Ascending by keys	Entry sequence
Index?	Yes	No
Access by——	Key or RBA	RBA
Position of free space	Each control interval/control area	End of data set
Reuse space?	Yes	No
Change RBAs by insertions/ deletions?	Yes	No

5.2.2 Structure

Common attributes of all VSAM data sets are as follows:
- VSAM data spaces.
- clusters.
- control areas.
- control intervals.
- physical blocks.
- records.
- relative byte addresses.

VSAM data spaces
VSAM data sets are not directly allocated by direct access storage device space management (DADSM), which allocates space for SAM, PAM, and DAM data sets in OS IV/F4. First, VSAM

acquires data spaces from which individual VSAM data sets are allocated. Thereafter, information on these spaces is entered into the VSAM catalog. VSAM data spaces can coexist with non-VSAM data sets on a volume.

In summary, VSAM data sets are allocated by VSAM from data spaces, based on information contained in the VSAM catalog.

Cluster

A cluster is a group of related VSAM data sets. For example, a KSDS actually comprises two data sets: the index, and the records themselves. These two data sets are collectively called a **cluster.**

An ESDS has no index, hence contains only one data set. Nonetheless, it is still considered to be a (trivial) cluster.

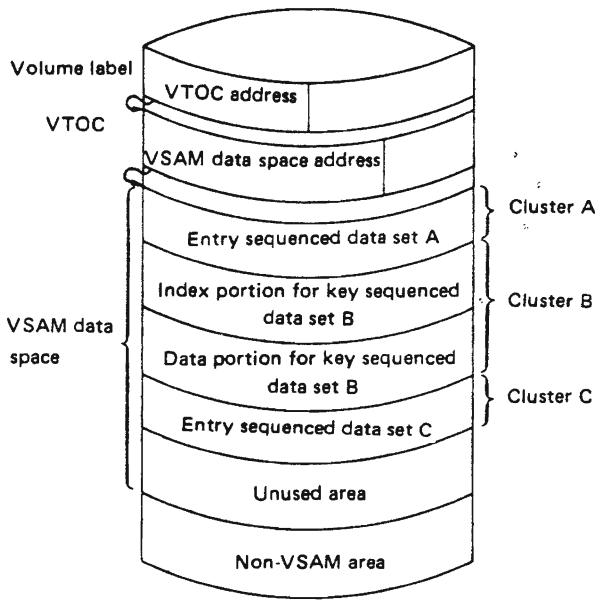


Fig. 5.2 VSAM data space and data sets

Control interval

VSAM controls transfers of data between I/O buffers and a VSAM data set; the unit of transfer is a **control interval.**

The length of a control interval is computed for each data set to be optimal for the DASD on which it is written, taking into consideration record lengths and I/O buffer lengths. Users can optionally select lengths for control intervals, with a maximum value of 32,768 bytes.

One or more records are contained in each control interval. One VSAM record can never extend over several control intervals.

The user can leave empty space in the control intervals of a KSDS to allow for future record insertions. Unused or reused space from deleted records is reutilized within each control interval.

Control area

A **control area** consists of multiple control intervals. The size of a control area is determined by its control-interval length, primary volume, secondary volume, space allocation, etc.

The maximum size of a control area is one DASD cylinder. One control area is the minimum unit for extending a VSAM data set.

When creating a KSDS, a user can allocate one or more empty control intervals for future record insertions within his control area.

Physical block

Each control interval contains one or more physical blocks. A physical block is the unit of transfer for actual I/O operations; VSAM blocks have a fixed length within each data set: either 512, 1024, 2048, or 4096 bytes.

VSAM selects the optimum block size for each data set, taking into consideration the lengths of records and control intervals. If the control interval is larger than one physical block, data are transferred between a control interval and I/O buffers by chains of physical blocks. Hence, the same control interval length can be maintained even if the data set is moved to a different type of DASD. The relationship will be described with reference to Fig. 5.3.

Suppose there exists control interval within an I/O buffer, as shown at the top of Fig. 5.3. The content of this control interval can be accommodated in one physical block. Thus, this control interval is divided into three physical blocks. In this case, records may extend over physical blocks, such as records 3 and 5 on Device 1.

Suppose this data set is shifted from Device 1 to Device 2, whose track length is shorter than that of Device 1. VSAM selects a different physical block length for Device 2, in order to better utilize DASD space. Thus, both the lengths and numbers of physical blocks change. Since the control-interval length remains constant, the user program sees no difference between processing the data set on Device 1 or Device 2.

Records

The VSAM record is the logical unit of data accessed by a program, just as for other access methods. The maximum VSAM record length is 32,761 bytes, the maximum length (32,768 bytes) of a control interval minus the minimum field (7 bytes) of necessary control information.

Relative byte address

Relative byte address (RBA) shows the relative position of a record from the start of the data set. Even if a data set contains many extents (or extends over many volumes), RBAs are determined in a consistent, device-independent way. VSAM converts RBAs into physical addresses on a device by considering the bytes/track, tracks/cylinder, and tracks/extent coefficients peculiar to this device and

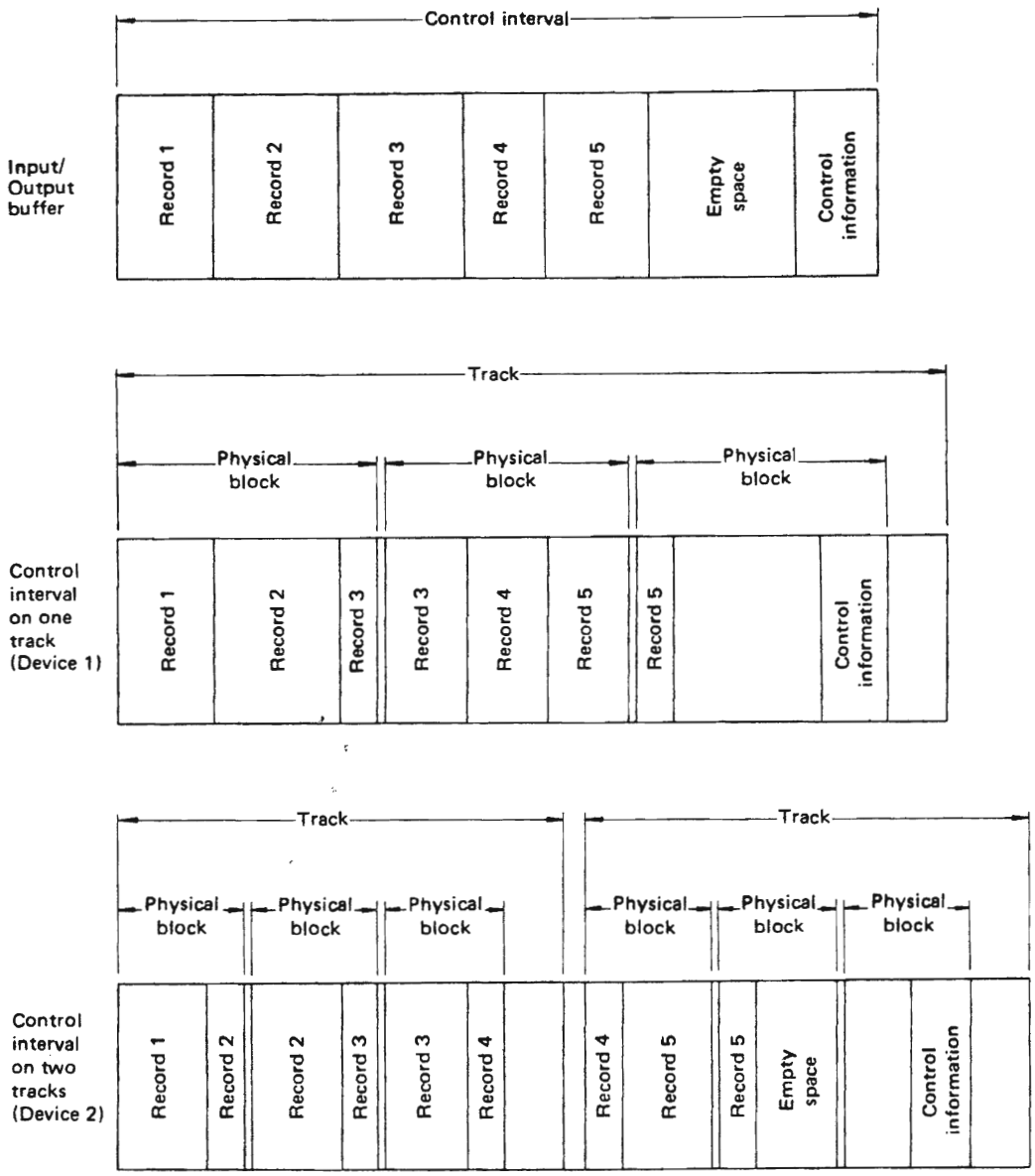


Fig. 5.3 Control intervals and physical blocks

data set. The user need not be conscious of physical addresses.

RBAs are calculated by VSAM when writing records into a VSAM data set. Users can access RBAs and thus achieve direct access to records. By processing records in sequence by their RBAs, records are accessed in sequence. Since an RBA is a four byte integer, one VSAM data set can have 2^{32} bytes — approximately 4.3 billion bytes.

5.2.3 Key Sequenced Data Sets

The internal structure of a key sequenced data set (KSDS) is quite similar to that of an indexed sequential data set.

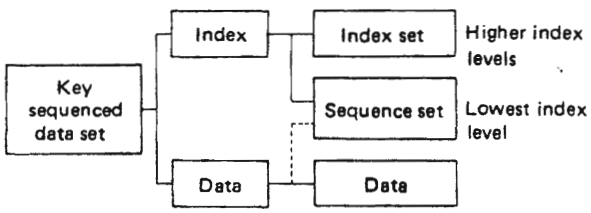


Fig. 5.4 KSDS structure

Structure

A KSDS is structured as shown in Fig. 5.4.

The data set contains an index (consisting of index levels) and data (consisting of records). The index and data are data sets independent of one other.

The **sequence set** is the lowest level of the index;

it can be optionally processed as data in order to increase access efficiency. The index levels of higher order than the sequence set is called the **index set.**

Fig. 5.5 shows an example of a KSDS.

Structure of the data portion of a KSDS

VSAM data can be at different levels such as control areas, control intervals, and data records.

Structure of a control interval

Each control interval normally contains the following three kinds of data, as shown in Fig. 5.6:

- groups of KSDS records.
- empty space.
- system control information.

When a KSDS is created, the user can plan to avoid clumsy future insertions by providing empty space on DASD tracks.

VSAM creates system control information at the right end of each control interval: a **control interval definition field** (CIDF) of four bytes, and a **record definition field** (RDF) of three bytes. The CIDF contains information about records and empty space within the control interval.

The RDF contains information about the record format: how it is placed within its control interval. RDFs do not necessarily match individual records. For example, when several consecutive records have the same length, their RDFs are combined to save space. RDFs enable VSAM to handle fixed-length and variable-length records in the same way. In Fig. 5.6, variable length records 1 to 3 are controlled by RDFs 1 to 3.

A KSDS utilizes space of deleted records or space released when records are shortened.

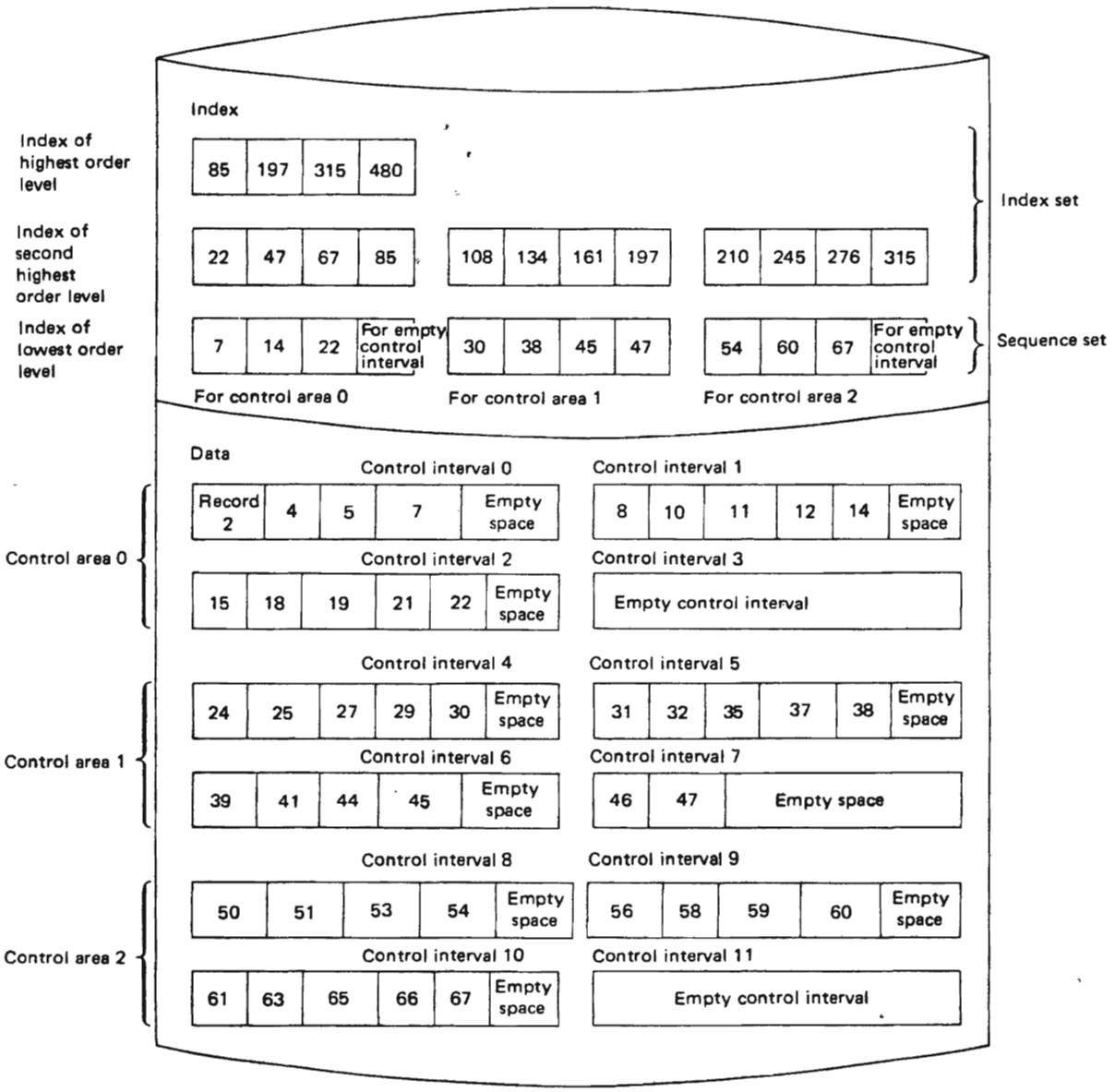


Fig. 5.5 Diagram of KSDS

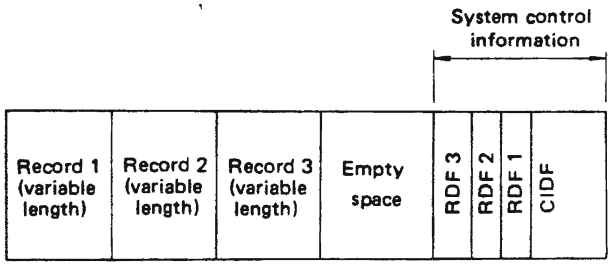


Fig. 5.6 Structure of control interval

Structure of a control area

A control area consists of multiple control intervals, as was shown in Fig. 5.5. It is also possible to leave empty space in a control area, just as in a control interval. By leaving an empty control interval within a control area, the user avoids rebuilding the control area later, which is likely to happen if he lengthens existing records or inserts records.

Structure of the index

The index controls records by keys, just like an ISAM data set.

Index levels

Index consists of a group of index records, including key values, pointers, and various kinds of control information.

A VSAM index normally has a tree structure of at least two levels. The highest level contains exactly one record. The lowest level of the index is called the **sequence set**, as shown in Fig. 5.7.

Each index record in the index set contains a vertical pointer to an index record one level lower. Each index record of the sequence set contains a horizontal pointer to the following index record of the same level, plus vertical pointers to all control intervals within one control area.

Structure of index records

Each index record has the format shown in Fig. 5.8. VSAM selects its size for each data set. An index record fills one control interval, thus having one RDF.

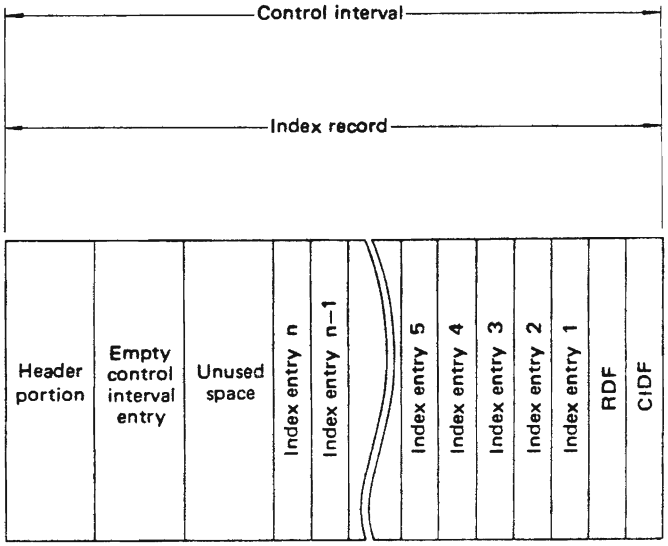


Fig. 5.8 Index record and control interval

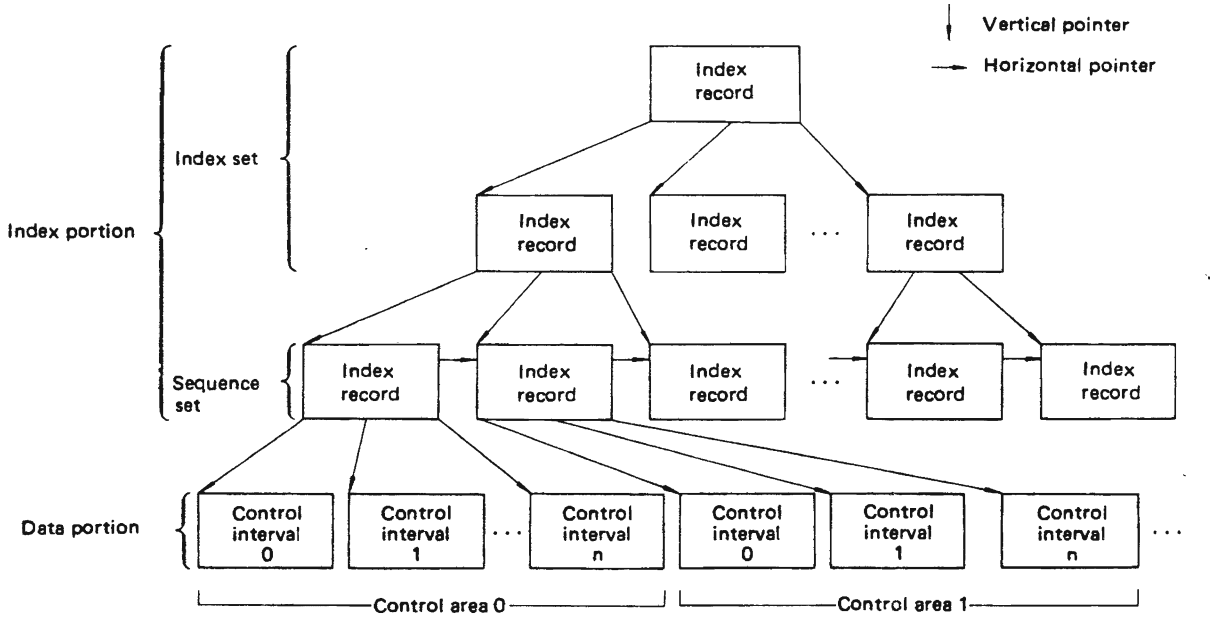


Fig. 5.7 Structure of index and data in KSDS

The header portion of an index record contains control information for this record. The next control interval entry exists only in the sequence set; otherwise, it is empty. The unused space is an empty space within the index record. If an index entry is added to this index record, it is entered into this unused space (if any).

An index entry has a different format, according to whether it is in the index set or the sequence set:

- Within an index record in the index set
  Key of the maximum index entry for the level immediately below, plus a pointer to the corresponding index record.
- Within an index record of the sequence set
  Key of the maximum record within the control interval, plus a pointer to this control interval.

Key compression

Each key within an index entry is equivalent to the key field of a particular record. However, keys are usually compressed to increase DASD space utilization and access efficiency.

Compressing keys consists of eliminating leading and/or trailing characters from consecutive keys represented by an index entry. Only characters necessary to uniquely identify each record remain after compression. As a consequence of key compression, the lengths of index entries vary, as shown in Fig. 5.9.

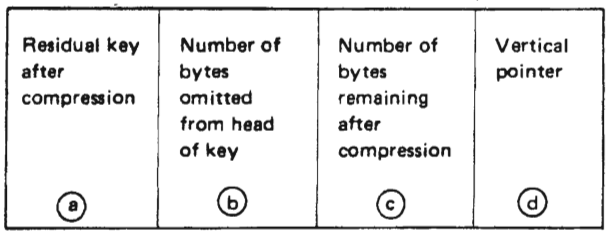


Fig. 5.9 Structure of an index entry

VSAM keys are subject to forward compression and backward compression. **Forward compression** omits leading characters from consecutive keys whose leading characters match. In Fig. 5.10, if key "1234557" immediately precedes key "1234567", the characters "12345" match; they are omitted from the key "1234567". The number of omitted characters is recorded in the index entry, "5" in this example.

After forward-compressing consecutive keys, VSAM compresses keys backwards as follows: if key "1234578" follows key "1234567", their sixth characters do not match; the seventh character, "7", is omitted from key "1234567".

In this way, after subjecting the preceding keys to forward and backward compression, only the character "6" remains from key "1234567". If this index entry pointed to the third control interval (i.e., relative number "2") within a control area, it would be written as [6|5|1|2] in the format of Fig. 5.10, requiring only four bytes rather than the original seven bytes.

Sectioning

VSAM automatically divides index entries into **sections** in order to increase the efficiency of index searches. The number of index entries in each section is approximately $\sqrt{N}$, where N is the total number of index entries at this level. For example, VSAM creates three sections of 5 entries, and two sections of 6 entries if N=27. Whenever a particular index entry is sought, instead of sequentially searching from the first index entry, VSAM compares its argument against the maximum key within each section. This technique enables VSAM to quickly locate the section containing the desired index entry, increasing search efficiency.

Options to increase index-search efficiency

With the DEFINE command of AMS, a user can request any/all of the following options for improving index-search efficiency:

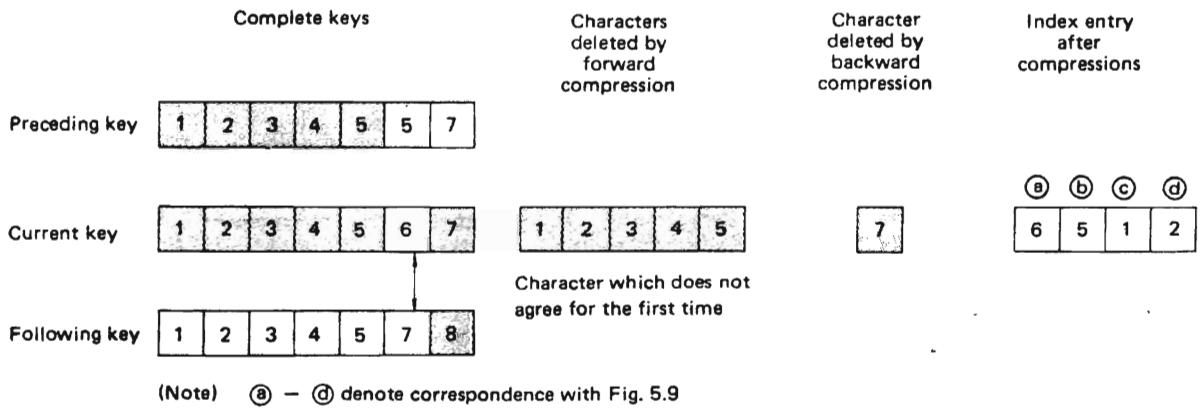


Fig. 5.10 Example of key compression

- creating the index and data portions of a KSDS on separate volumes.
- replicating index records.
- including the sequence set in the data portion.
- replicating the sequence set.
- reading part/all of the index into virtual storage.

Creating the index and data portions on separate volumes

This option improves retrieval times as follows:

- simultaneous access to index and data portions.
- potential for using a drum (very fast access, limited capacity) for the index.

Replicating index records

Fig. 5.11 shows how index records can be replicated. If the user selects this option, VSAM allocates a full track to each index record, replicating this record as many times as the track capacity permits. Rotational delays are minimized by this strategy; however, it increases the amount of DASD space required for an index, in order to improve response times and channel throughput.

Including the sequence set in the data portion

The user can request VSAM to create his sequence set within the data portion of his KSDS rather than in the index. This option can sometimes increase efficiency of searching the sequence set.

Replicating the sequence set

Index records are written onto the first track of the corresponding control area, replicated as much as the track can accommodate, as shown in Fig. 5.11. This technique minimizes time spent searching for an index record. Since the control interval containing the desired record is on the same cylinder, the DASD access mechanism need not be moved, improving overall DASD efficiency. This method is particularly effective for random or skip-sequential accesses to many records.

Reading an index into virtual storage

VSAM has a capability similar to that of ISAM for holding part/all of the master and cylinder indexes for an ISAM data set in main storage.

Using the DEFINE command of AMS, a VSAM user can assign a predetermined number of virtual-storage buffers to hold index records whenever this KSDS data set is accessed. These buffers are filled with currently-useful index records when VSAM commences processing the data set. If all buffers become filled, VSAM overwrites those index records (in virtual storage) least likely to be useful thereafter during this job step. With this approach, VSAM reads index records from DASD as infrequently as possible. Often, VSAM can load the entire index into virtual storage as the data set is being opened.

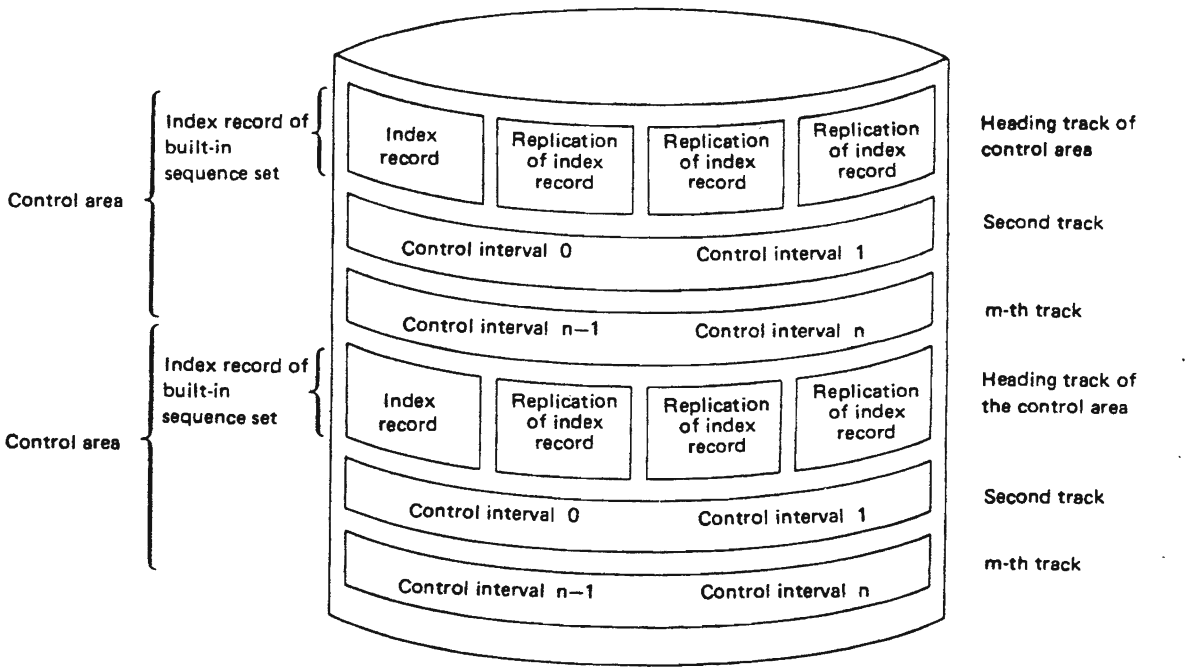


Fig. 5.11 Creating a sequence set in a data portion

5.2.4 Entry-Sequenced Data Sets

An ESDS has no index; its data portion has the same format as that of a KSDS, except as follows:

- records may not include logical keys.
- empty space cannot be preallocated to control intervals or control areas.

5.3 VSAM PROCESSING

VSAM provides a wide variety of access techniques for key-sequenced and entry-sequenced data sets.

5.3.1 VSAM Access Techniques Overview

There are five access techniques, as shown in Table 5.2.

Table 5.2 VSAM access techniques and data sets

Access by	Access type	Key sequenced data set	Entry sequenced data set
Key	Sequential	X	
	Direct	X	
	Skip sequential	X	
RBA	Sequential	X	X
	Direct	X	x

Also, system programmers can directly read, write, and update control intervals, although this practice is not usually necessary or recommended for ordinary users.

The VSAM user can shift from one access technique to another whenever necessary, without having to open and close his data set repeatedly.

Access processing

Sequential access processes records in their sequence of entry (ESDS) or their key sequence (KSDS).

For direct access, the user must furnish a key or RBA with each request; only after receiving his request does VSAM schedule a corresponding I/O request, allocate one or more buffers, etc. (In general, VSAM acquires an appropriate number of buffers when it opens the data set; it assigns these buffers to control intervals only when it receives specific record requests.)

In either case, VSAM deblocks records one by one if the user has requested move-mode processing, moving these records to the work areas he designates. If the user has requested locate-mode access, VSAM points general register 1 to each logical record. VSAM permits users to optionally perform their own synchronization of I/O with processing.

5.3.2 KSDS Processing

All VSAM access techniques can be used with KSDS, although most commonly access is by key. After deletion, insertion, or updating of a KSDS record, its RBA may change.

Sequential access by key

Records are processed sequentially in key order, even if the user does not furnish keys in his I/O requests.

- Reading records

  When a KSDS data set is opened, VSAM presents the record with the lowest key. If the user wants to start processing with another record, he must issue a macro instruction to position VSAM appropriately.

  Reading is performed by GET macro instructions.
- Deleting records

  After a record has been requested by a GET macro instruction for updating purposes, it can be deleted by an ERASE macro instruction.

  With ISAM, it was impossible to reuse space vacated by deleted records. With VSAM, a deletion creates useable empty space. Space reutilization is shown in Fig. 5.12. The reader should note that the RBAs for records "35" and "36" change after the deletion.
- Updating records

  After the user has retrieved a record by a GET macro instruction he can optionally issue a PUT macro instruction to update this record. He can change the length of a key-sequenced record. If he shortens its length, VSAM performs space reutilization; if he lengthens the record, the following records are moved back within their blocks, control intervals, etc. In either case, RBAs for all records after the updated record change.

  When updating, the user cannot change a key or its position within the record.

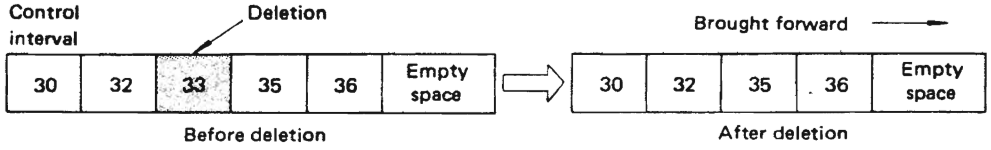


Fig. 5.12 Space reutilization

- Adding records

  Additions can either be insertions within the data set (between two existing records) or additions at the end of the data set.

  VSAM can insert several new records simultaneously into one control interval, using a highly-efficient set of I/O operations as follows.

  All read, insert, delete, and other operations are performed on the virtual-storage image of the control interval, which is then written to the DASD as shown in Fig. 5.13. Hence, VSAM does not require several different block-oriented I/O operations to insert new records onto a DASD track, as ISAM does; VSAM updates the in-memory image of the entire control interval, which it writes back to DASD with an efficient chained-block channel program.

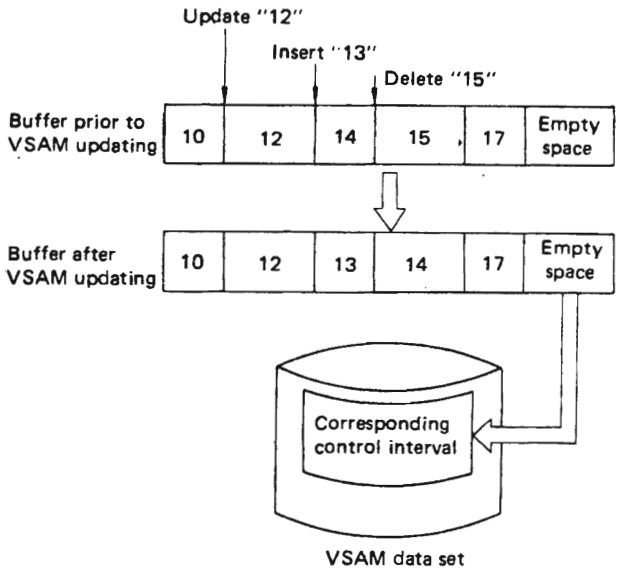


Fig. 5.13 Example of multiple simultaneous insertions/deletions

- Sectioning control intervals

  If adequate empty space is available in a control interval, insertion is a simple, efficient process; no sectioning of the corresponding control interval is required.

  Control-interval sectioning is a VSAM feature for introducing empty space at the end of a control interval to facilitate future expansion. A record whose key exceeds that of a record about to be inserted into a control interval is moved to an unallocated control interval within the same control area. Thereafter, the inserted record becomes the last record of the first control interval, as shown in Fig. 5.14. In this figure, the striped boxes indicated changed record areas, and the circled numbers denote pointers to control intervals.

  RBAs of records in the striped boxes of Fig. 5.14 have changed. After inserting record "21", the user can read sequentially by key as follows: 11, 14, 15, 17, 19, 21, 22, 24, 28, 30, 31. If he reads sequentially by RBA, his access is by physical sequence of the records: 11, 14, 15, 17, 19, 21, 28, 30, 31, 22, 24.

- Sectioning control areas

  This is exactly analogous to sectioning control intervals, except that empty space is left at the end of an entire control area, defined in terms of partially/totally empty control sections.

Direct access by key

VSAM searches for the key supplied by each user request, beginning with the highest-order index level and continuing down to an index record in the sequence set. The search time depends on whether the required index levels are already in virtual storage buffers.

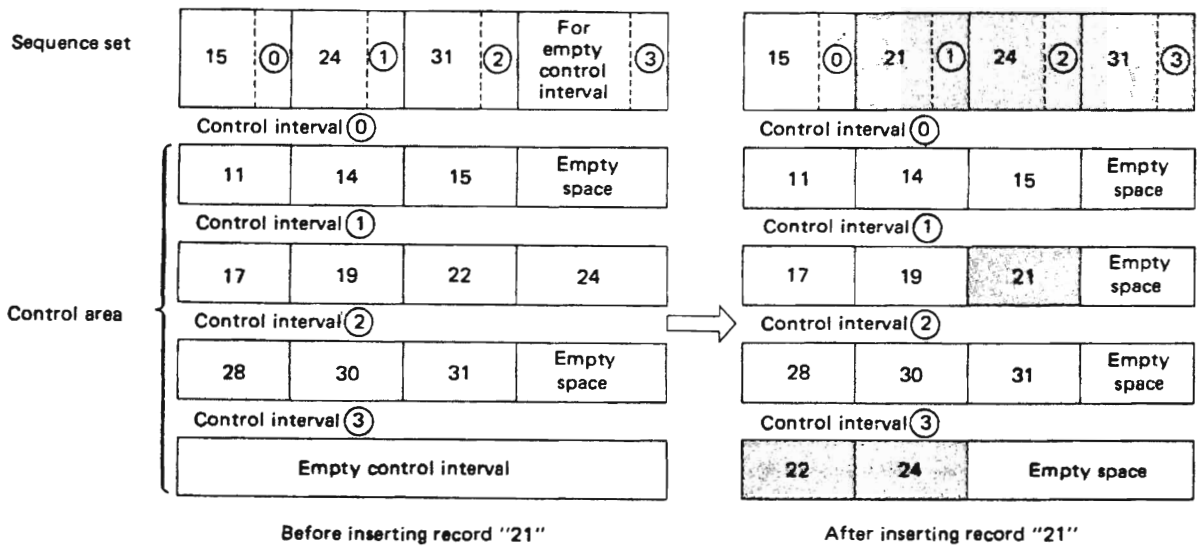


Fig. 5.14 Sectioning a control interval

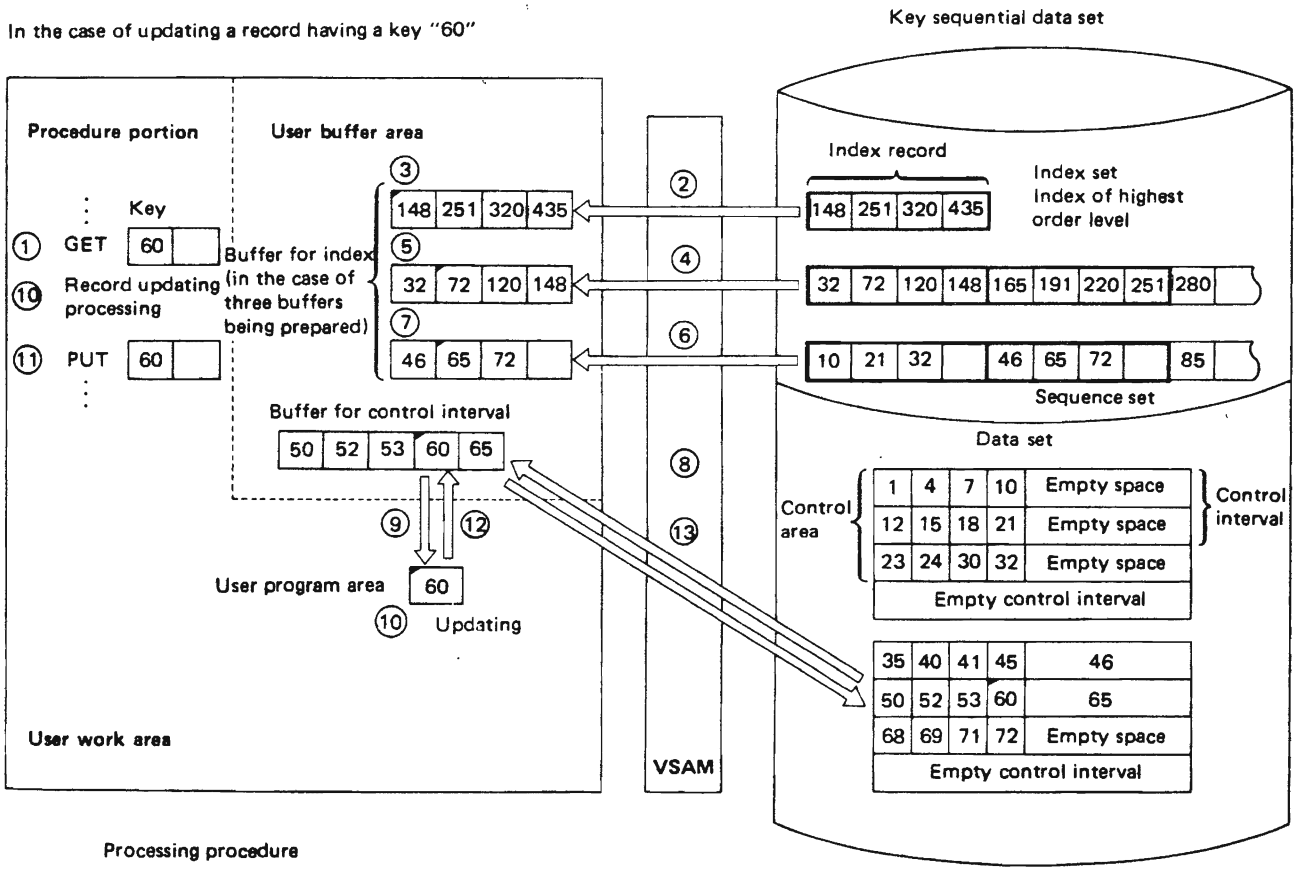
Using direct access by key, the user can read, delete, update, insert, and add records to a KSDS. It is impossible to perform several simultaneous changes to one control section without writing it back to DASD several times, since continual updating of the DASD image is needed to insure data integrity.

- Reading records
  The user furnishes a complete or generic key with each GET macro instruction. Using a **complete key**, he can request either the record having this exact key or the record whose key is equal or next-higher than this key.
  Using a **generic key** of N bytes, the user requests either the first record from the data set whose key matches the furnished key for at least N bytes or the first record whose key is equal to or greater than the furnished key, considering only the first N bytes of each key.
- Deleting records
  After reading a record by key, the user can delete it by issuing an ERASE macro instruction.
- Updating records
  After reading a record by key, the user can update it by issuing a PUT macro instruction, as shown in Fig. 5.15.
- Adding records
  The user inserts or adds records by issuing PUT macro instructions.

Skip-sequential access by key

The skip-sequential access technique searches horizontal pointers of the sequence set without vertically searching the index each time. The skip-sequential technique has somewhat higher efficiency than direct access. Fig. 5.16 shows the difference between direct access by key and skip-sequential access. The first record to be retrieved has key "A". VSAM locates it in the sequence set by using vertical pointers from the index in the sequence (1) → (2) → (4).

If record "B" is now requested, its key is only slightly higher than the present key. Using direct access, VSAM must repeat the entire vertical search. This requires substantial unnecessary search time for index records, resulting in lower efficiency.



Processing procedure

① Assign the key "60" and generate GET macro.
② VSAM transfers index record of highest order level to the buffer.
③ ④ Find an entry "148" and transfer its index record.
⑤ ⑥ Find an entry "72" and transfer its index record.
⑦ ⑧ Find an entry "65" and transfer its control interval.
⑨ ⑩ Transfer to user work area (during move mode) and update.
⑪ ⑫ Return again to the buffer from the PUT macro.
⑬ Store data set in VSAM.

Fig. 5.15 Example of direct access by key

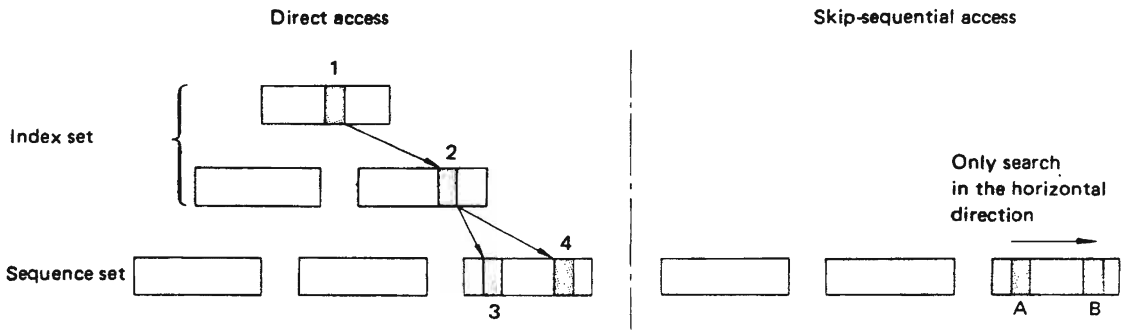


Fig. 5.16 Direct access by key and skip-sequential access

The skip sequential technique processes these two requests as follows. Since VSAM sees that record "B" has a higher key than record "A", it omits searching the index set. The index entry for "B" is immediately found.

As the example indicates, if a user is following each direct access by several sequential accesses, he should use the skip-sequential technique. However, he should use direct access if he is reading individual records truly at random.

Sequential access by RBA

For sequential access by RBA to a KSDS, only reading, deleting, and updating of records are allowed.

- Reading records
  After opening the data set, VSAM starts processing with the first record (RBA=0). To start processing from an intermediate record, the user issues a POINT macro instruction furnishing its RBA.
  Just as for sequential access by key, sequential access by RBA has two modes: one for reading a record, one to prepare for updating or deleting it after reading.
- Deleting Records
  After his GET macro instruction, the user can issue an ERASE macro instruction. When reading and deleting records by RBA, VSAM doess not update the index; hence corresponding records do not exist if he later attempts to access them by key.
- Updating Records
  After a GET macro instruction, the user can update a record by issuing a PUT macro instruction. He may not change either the length of the record or the value of its key.

Direct Access by RBA

A user can directly access a KSDS by RBA. In this mode, he can read, delete, or update records.

- Reading records
  The user must furnish an RBA with each GET macro instruction.
- Deleting records
  The user issues an ERASE macro instruction after formed just as for direct access to a KSDS by RBA. a GET macro instruction. VSAM will reutilize any freed space, but it does not update the index.
- Updating records
  The user issues a PUT macro instruction after a GET macro instruction to update a record.

Example of access by RBA

Fig. 5.17 shows the sequence set and data portion of a KSDS. During direct access by key, the user may request the record whose key is "20":

- VSAM searches the index; in the sequence set, three entries for its control interval are found.
- VSAM knows that "0" is the first RBA of the control area.
- VSAM adds "2048" to "0", the former number obtained by multiplying "1024" (the control interval length) by "2" (the sequence number of the control interval) to determine the first RBA of control interval 2.
- Based on this, control interval 2 is read into a buffer and searched sequentially to locate the record whose key is "20."

To read record "20" by direct access using RBA, the user must previously have learned that the RBA of record "20" is "2500". If he issues a GET macro instruction specifying "RBA=2500", the second control interval is read into his buffer; the record is located immediately.

5.3.3 Processing Entry-Sequenced Data Sets

Since an ESDS has no index, only RBA access is possible, which is similar to accessing a KSDS by RBA. Reading, updating, and adding records are permitted for ESDS. To delete ESDS records, users must assign and manage their own deletion codes.

- Sequential access by RBA
  Reading and updating ESDS records sequentially are performed the same way as for KSDS records. Records cannot be inserted into the middle of an ESDS, but they can be added at its end.
- Direct access by RBA
  Reading, updating, and adding records are performed just as for direct access to a KSDS by RBA.

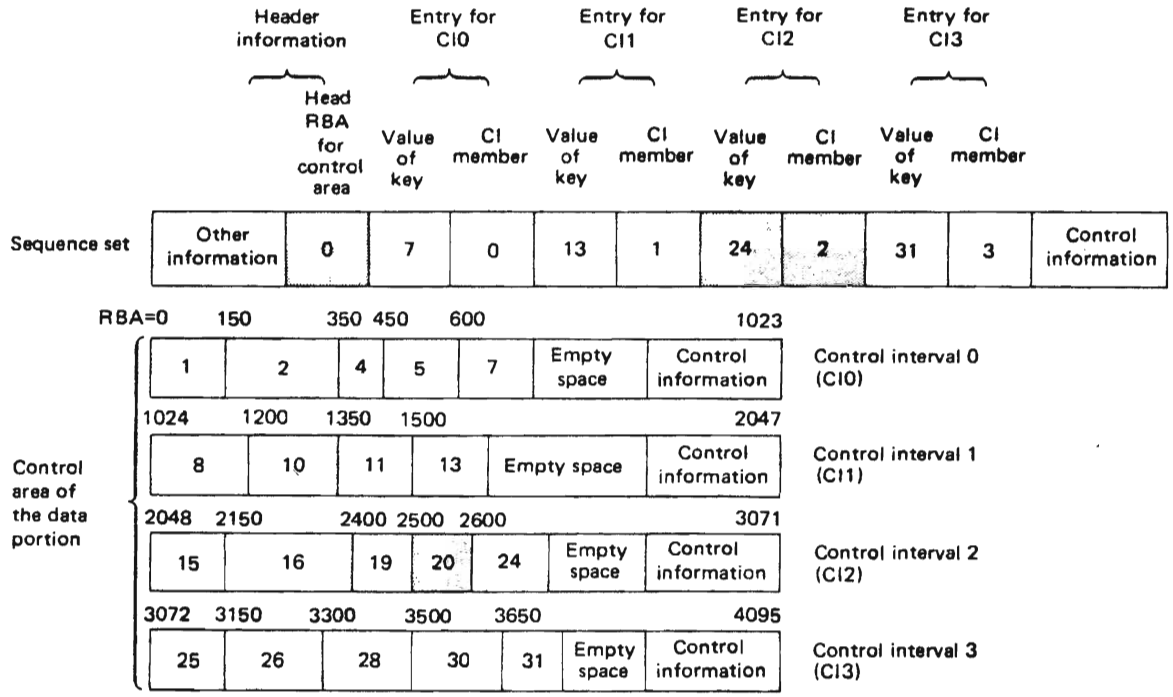


Fig. 5.17 Example of access by RBA

Just as for sequential access to ESDS, deleting records is not handled by VTAM; a user must choose his own deletion code and store it into records when they are deleted, recognizing this code when he reads the data set.

5.3.4 Types of Processing Supported by VSAM

Table 5.3 summarizes the modes of access to VSAM data sets in OS IV/F4.

5.3.5 VSAM Macro Instructions

VSAM furnishes various control blocks and macro instructions for Assembler language programmers. Programmers writing in COBOL, PL/I, etc. use VSAM via the ISAM interface described in Section 5.5.

This section describes VSAM control blocks, followed by macro instructions for the following purpose:

- Creating control blocks.

Table 5.3 Types of processing supported by VSAM

Data set	Access mode	I/O operation				
		Reading	Updating	Deleting	Inserting	Adding
KSDS	Sequential access by key	X	X	X	X	X
	Direct access by key	X	X	X	X	X
	Skip sequential access by RBA	X	X	X	X	X
	Sequential access by RBA	X	X *	X		
	Direct access by RBA	X	X *	X		
ESDS	Sequential access by RBA	X	X *	X **		X
	Direct access by RBA	X	X *	X **		X

* Lengths cannot be altered.
** Users must write their own deletion codes into deleted records.

- Dynamically managing control blocks.
- Linking and separating VSAM data sets from processing programs.
- Processing VSAM requests.

Fig. 5.18 shows the relationship between these control blocks and macro instructions.

VSAM control blocks

VSAM creates certain control blocks when opening a data set. Users create other control blocks to provide VSAM with various kinds of information: ACB, RPL, and EXLST.

- Access method control block (ACB)
  The ACB contains information about allowable processing modes for the data set, passwords, number and size of I/O buffers, addresses of various exit routines, etc. The ACB of VSAM is functionally analogous to the DCB of SAM, PAM, DAM, etc.
- Request parameter list (RPL)
  Each RPL defines one specific I/O request; it is analogous to the DECB (data event control block) of SAM, PAM, DAM, etc.
- Exit list (EXLST)
  An EXLST parameter furnishes addresses for various types of exit routines. EXLST has the same format and general functions for VSAM as for other access methods.

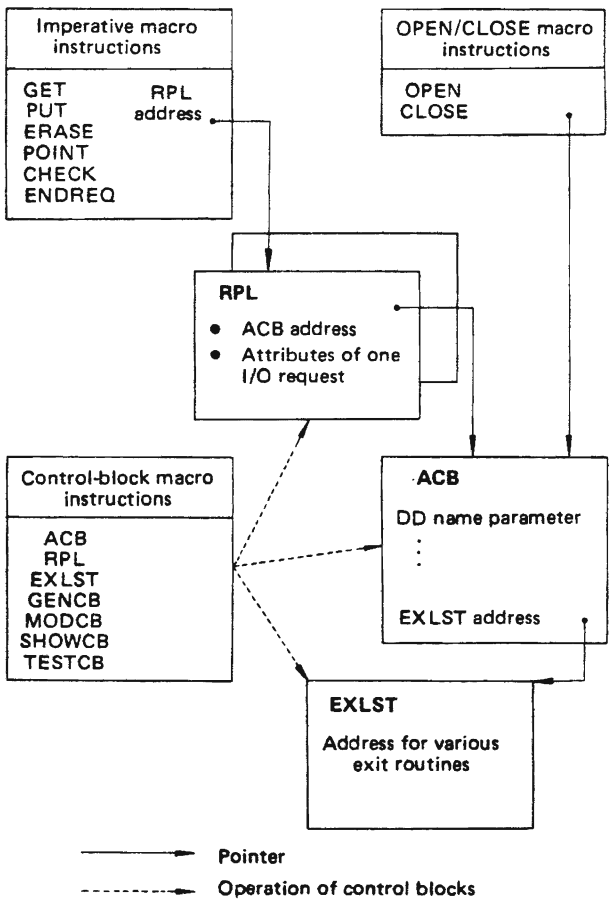


Fig. 5.18 Relationship between VSAM control blocks

Macro instructions for creating control blocks

The ACB, RPL, and EXLST macro instructions are used in programs as follows:

ACB macro instruction

VSAM OPEN routines process the ACB generated by an ACB macro instruction (or dynamically created by the user program with a GENCB macro instruction). The principal attributes which the user codes in his ACB macro instruction are as follows:

- Number of buffers for data records, number of buffers for index records, and aggregate memory to be allocated by VSAM.
- Addresses of various user-furnished exit routines.
- Processing options.
- Passwords (if omitted from the ACB, passwords are requested from the operator during program execution.)
- Maximum number of concurrent I/O requests.

RPL macro instruction

An RPL defines a specific I/O request; one RPL is created by each RPL macro instruction. A user can dynamically alter the content of an RPL by issuing a MODCB macro instruction during execution. He can furnish any/all of the following parameters in an RPL macro instruction:

- ACB address.
- Work-area address and length.
- Key address (or RBA address).
- Address for a message about any unrecoverable I/O error; and the length of this message.
- Next RPL address (if any)
  VSAM will process multiple records within a data set in one I/O request, if the user builds a chain of RPLs as shown in Fig. 5.19.
- Various processing options.
- Updating/no updating of records.

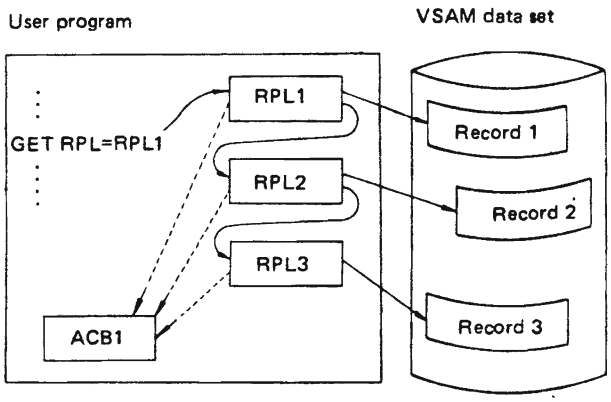


Fig. 5.19 Chained requests

EXLST macro instruction

Users can furnish addresses for various exit routines in EXLST macro instructions:

- I/O error analysis routine
  When an unrecoverable error occurs during an I/O operation, VSAM enters the routine so that the user can analyze detailed error information.
- Logical error analysis routine
  VSAM enters this routine when an I/O problem other than an unrecoverable hardware failure occurs.
- End of data address (EODAD)
  During sequential access by key or RBA, if the user attempts to read past the last record of the data set, VSAM passes control to the EODAD routine.
- Journaling routine
  When a record is added to a KSDS, control passes to this exit so the user can record how RBAs of this data set change thereafter.

Macro instructions for dynamically creating/modifying VTAM control Blocks

ACB, RPL, and EXLST macro instructions create control blocks when a user program is assembled. Alternatively or in addition, the user can issue macro instructions during execution of his program which dynamically create or modify control blocks.

A GENCB macro instruction creates a VSAM control block. A MODCB macro instruction can change one or more fields of an existing control block. A SHOWCB macro instruction displays part or all of a control block. A TESTCB macro instruction tests the value of a specified field in a VSAM control block.

These macro instructions help users create reentrant programs. Also, each user can create control blocks to reflect data sets he is processing against various transactions.

- GENCB (Generate Control Block) macro instruction
  A user executes a GENCB macro instruction to create an ACB, RPL, or EXLST.
- MODCB (Modify Control Block) macro instruction
  A MODCB macro instruction dynamically changes the contents of an ACB, RPL, or EXLST. By using this macro instruction, a user can adapt preassembled control blocks to meet specific needs identified during execution of his program.
- SHOWCB (Show Control Block) Macro instruction
  A SHOWCB macro instruction displays the contents of an ACB, RPL, or EXLST in a user-furnished work area during execution of his program. With this macro instruction, the user can scrutinize information newly added by VTAM to his control blocks.
- TESTCB (Test Control Block) macro instruction
  A TESTCB macro instruction tests specific fields of an ACB, RPL, or EXLST, usually to select subsequent program options.

OPEN and CLOSE macro instructions

OPEN associates one or more data sets with a processing program. It also prepares various routines necessary for processing the data sets. CLOSE terminates processing of these data sets and releases resources (virtual storage buffers, I/O routines, etc.) used for this processing.

OPEN macro instruction

VSAM associates an ACB (access method control block) with a data set. VSAM creates various control blocks based on each DD statement and associated catalog information. VSAM loads routines necessary for GET/PUT/ERASE/etc. processing into the user's region, or it connects already-loaded routines to the corresponding ACB.

If the data set is password-protected, OPEN checks for the correct password, furnished either in an ACB parameter or on the corresponding DD statement. VSAM checks whether the index and data portions are separately processed, in the case of a key sequenced data set.

When processing a multivolume VSAM data set, the console operator need not mount all volumes before/at OPEN time. When initially creating a VSAM data set, a user furnishes a DEFINE command with AMS to define the range of keys on each volume. When the user is ready to access this data set at a later time, he can limit the number of volumes to be mounted by limiting the range of keys at OPEN time.

CLOSE macro instruction

When the user issues a CLOSE macro instruction, VSAM writes all data and index records remaining in his I/O buffers to DASD, then separates the ACB from the data set, and erases various VSAM control blocks and processing routines created by OPEN. The CLOSE routine restores each control block to its pre-OPEN format.

If the end of the data set has moved due to additions or insertions of records, the corresponding VSAM catalog entry is updated. If the installation is capturing SMF data, VSAM writes various SMF records to the current SMF data set.

The TYPE=T option can be requested with CLOSE, which is often called the **TCLOSE function.** TCLOSE writes any portions of the current control interval remaining in I/O buffers onto DASD, to insure logical integrity of the data set. TCLOSE also updates the catalog entry for this data set. After issuing a TCLOSE macro instruction, the user can optionally resume processing this data set without having to re-open it.

Imperative macro instructions

For VSAM, the user can issue GET, PUT, POINT, ERASE, CHECK, and ENDREQ macro instructions, each requiring only an RPL parameter.

- GET macro instruction
  GET reads one record from a data set. With GET, the user can access VSAM records in sequential, direct, or skip-sequential sequence. He can perform synchronous or non-synchronous processing. He can optionally prepare to update each record.
- PUT macro instruction
  Following his GET macro instruction the user can issue a PUT macro instruction to update, insert, or add a record to the data set.
- POINT macro instruction
  The user issues a POINT macro instruction to position VSAM to a particular record within the data set.
- ERASE macro instruction
  An ERASE macro instruction is used together with a GET macro instruction to delete a specific KSDS record.
- CHECK macro instruction
  A CHECK macro instruction is used to synchronize a task with an I/O operation; for VSAM, CHECK plays the same role as for BSAM, BPAM, and BDAM.
- ENDREQ macro instruction
  An ENDREQ macro instruction cancels an asynchronous I/O request issued by another macro instruction.

5.4 VSAM CATALOG

5.4.1 Overview

The VSAM catalog is distinct from the OS IV/F4 system catalog, which was described in Section 4.14. Non-VSAM data sets need not be entered into the OS IV/F4 system catalog. However, all VSAM data sets must be entered into a VSAM catalog, since most of their attributes are retained only in this catalog.

Types of catalogs
Catalogs for VSAM data sets are called **VSAM catalogs,** comprising one **master catalog** pointing to an optional number of **user catalogs.**
Master and user catalogs are identical in format, but a user catalog has the following special purposes:

- Reduce the size of the master catalog.
- Minimize search times.
- Improve the integrity (resistance to failure) of the overall catalog structure.

Fig. 5.20 shows an overall diagram of catalogs and data sets.

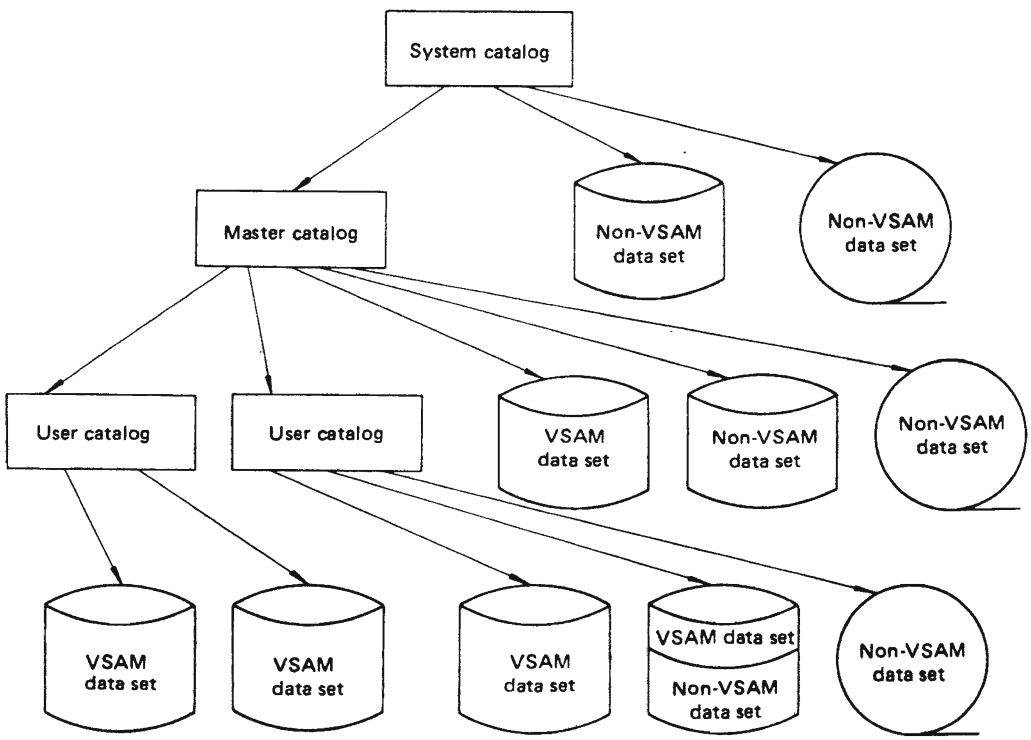


Fig. 5.20 OS IV/F4 catalogs and data sets

VSAM catalogs and JCL parameters
A VSAM catalog not only points to the volumes containing a VSAM data set – like the system catalog – but also contains many attributes of the data set. VSAM always consults a VSAM catalog when processing a VSAM data set, even when merely requesting additional space for the data set.

Due to the functional breadth of the VSAM catalogs, data definition (DD) statements become quite simple. In other words, attributes of a data set are retrieved from the VSAM catalog when it is to be processed, rather than from DD statements. Space management information can also be obtained from the VSAM catalog, except when initially securing a VSAM data space.

Since entries and deletions from the catalog are performed by the AMS utility routine, JCL parameters for cataloging/uncataloging are unnecessary for VSAM data sets.

The following JCL rules and procedures are added to basic OS IV/F4 data management to support VSAM:

JOBCAT and STEPCAT DD statements
These define VSAM user catalogs used in particular jobs and job steps, respectively.

The AMP parameter
The AMP parameter can furnish any/all parameters specified ordinarily by an ACB or EXLST macro instruction. In addition, AMP can furnish sub-parameters for the following attributes:
- Checkpoint/restart option.
- ISAM interface option.
- Control-block dump option.
- Location for a VSAM data set having a specified volume serial number.

5.4.2 Contents of the VSAM Catalog

An installation can store volume pointers for non-VSAM data sets in the VSAM catalog as well as attributes of VSAM data sets. This approach speeds up catalog searches, compared with using the OS IV/F4 system catalog. It is impossible to define generation data sets in a VSAM catalog.

Information about data sets
- Correspondences of relative byte addresses (RBAs) with physical positions on each volume.
- Pointers to each extent of the data set.
- Various activity and format statistics, such as number of inserted records, number of deleted records, empty space information, etc.
- Size of control intervals, number of control intervals per control area, size of physical blocks, position of the key, etc.
- Passwords.
- Timestamp information.
- For a KSDS, a flag indicating whether data and index portions are to be processed separately or not.
- End of data set processing option, etc.

Information about volumes
- Volume serial number and device type of each volume.
- Number of data spaces and data sets.
- Positions of data spaces, etc. on each volume.
- Unallocated areas in each data space, etc.

5.4.3 Using the VSAM Catalog

Opening a VSAM data set
When a user issues an OPEN macro instruction, the VSAM catalog is accessed as shown in Fig. 5.21.

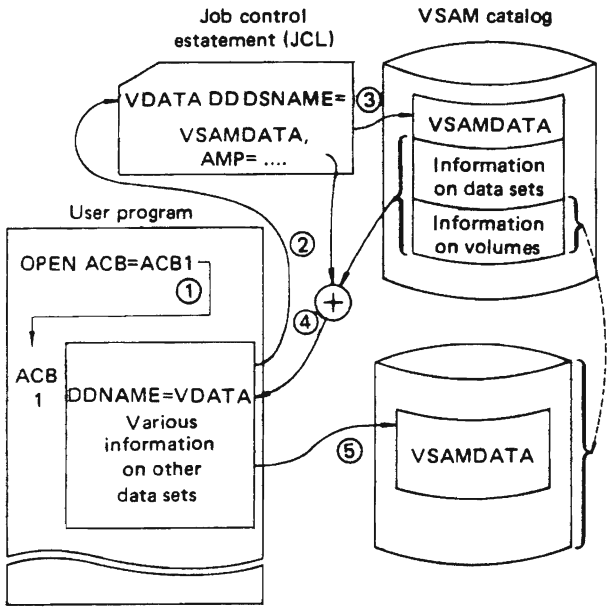


Fig. 5.21 Example of OPEN macro instruction and the VSAM catalog

Accessing VSAM catalogs
The user can furnish a JOBCATT DD statement (whose name field contains "JOBCAT") to allocate a user catalog for the duration of an entire job; this DD statement immediately follows his JOB statement. The user can furnish a STEPCAT DD statement (whose name field contains "STEPCAT") to allocate a user catalog for one step. If he omits both JOBCAT and STEPCAT statements, VSAM utilizes the master catalog for all searches.

Sequence of catalog searching
Catalogs are searched in the following sequence by OS IV/F4.

- User catalog designated in a STEPCAT statement.
- User catalog designated in a JOBCAT statement.
- VSAM master catalog.
- OS IV/F4 system catalog.

5.5 ISAM INTERFACE ROUTINES

ISAM interface routines enables ISAM programs to use VSAM for processing VSAM data sets. The following subsections describe how the interface operates and constraints on its use.

5.5.1 Overview

Indexed sequential and key sequenced data sets are similar to each other in structure and processing functions. Hence, most ISAM programs can process VSAM data sets with little or no alterations. The ISAM interface routines of VSAM facilitate this transition, as shown in Fig. 5.22.

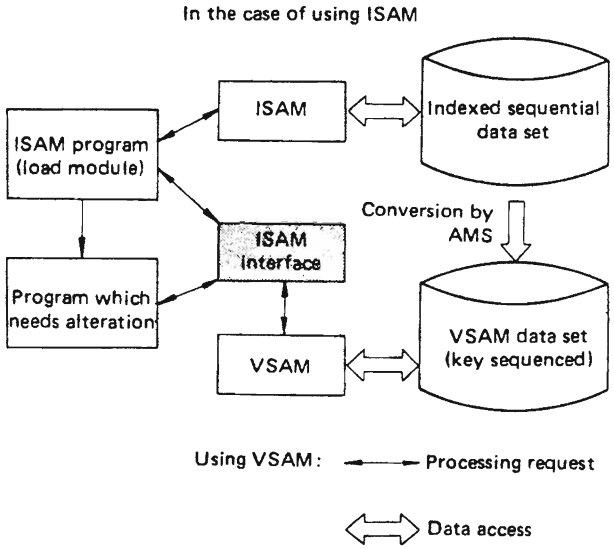


Fig. 5.22 Location of the ISAM interface routines

5.5.2 ISAM Interface Processing

A VSAM data set can be processed with the ISAM interface as follows:

1) The user allocates DASD space for the new VSAM KSDS with a DEFINE command of AMS (access method services). He then copies his ISAM data set to the KSDS with a REPRO command.
2) If necessary, he changes any aspects of his ISAM source program (typically few or none) which cannot be handled by the ISAM interface routines.
3) He changes his JCL to its VSAM equivalent.
4) He excecutes his program; when he issues ISAM macro instructions, the ISAM interface routines convert them into corresponding VSAM macro instructions.
5) When his ISAM-oriented program opens the KSDS, an ISAM interface routine receives control and performs the following services:
- Loads additional ISAM interface routines.
- Creates control blocks needed by VSAM.
- Initializes the ISAM DCB in the user's program, so that it appears to be opened normally.
- Sets any necessary DCB exit routines

6) When thhe user issues an imperative ISAM macro instruction (READ, GET, etc.), an ISAM interface routine receives centrol, selects the corresponding VSAM macro instruction, creates an appropriate RPL, and yields control to the appropriate VSAM routine.
7) When the previous step completes, the ISAM interface routine translates the VSAM return code into the corresponding ISAM return code, appropriately updates the ISAM DCB, and returns control to the application program.
8) When the user issues a CLOSE macro instruction, an ISAM interface routine terminates processing of the data set; thereafter, the ISAM interface routine loaded when this data set was opened is deleted (unless it is still in use for another VSAM data set).

5.5.3 Restrictions of the ISAM Interface

Since VSAM cannot perform all ISAM functions, users must make minor changes to their ISAM-using programs. Typical constraints are shown in Table 5.4.

Table 5.4 Typical constraints on using the ISAM interface

Constraint	Recommended alternative
OPEN (TYPE=J) macro instructions cannot be used with VSAM.	Redesign program so it does not need OPEN (TYPE=J).
Although ISAM data sets can be temporary. (DSNAME omitted), VSAM data sets cannot be temporary	Either (a) allocate a permanent VSAM data set for the same function or (b) use a non-ISAM temporary data set for this function.
ISAM counts accesses to overflow areas.	This function is unnecessary and obsolete with VSAM; associated routines may be deleted.
Some ISAM SETL macro instructions have no VSAM counterparts.	Change to other forms of SETL which are supported by the ISAM interface.

5.6 SHARED AND EXCLUSIVE CONTROL OF VSAM DATA SETS

Several users can concurrently share a VSAM data set:

- Subtasks within one region.
- Jobs between regions.
- Users of different OS IV/F4 configurations.

Since VSAM manages control blocks by region, complete data set integrity can be assured for sharing by subtasks, without any special action by users.

When sharing data sets between jobs or configurations, users must define and perform their own data set integrity control for simultaneous reading, updating, etc.

5.6.1. Sharing by Subtasks

When sharing a data set among sub-tasks, the unit of exclusive control for a VSAM data set is the control interval; the reader should review Section 4.11 for a general discussion of how data sets are shared or exclusively controlled. Exclusive control is unnecessary for several subtasks concurrently reading a VSAM data set.

5.6.2 Sharing by Jobs

In this case, each job must assign DISP=SHR. With an option of the AMS DEFINE command, the creator of a VSAM data set can authorize/prohibit its being shared simultaneously by different jobs in one OS IV/F4 system.

Exclusive control among jobs is exercised by cluster name when the user issues an OPEN macro instruction. By specifying DISP=OLD on his DD statement, a user can insure that his job gains exclusive control of the data set before executing.

5.6.3 Sharing Between OS IV/F4 Configurations

As described in Section 4.11.3, a user must issue RESERVE/DEQ macro instructions if he needs to gain exclusive control of a data set shared by several OS IV/F4 configurations. To keep DASD blocks always current, the VSAM user can elect a sharing option so as to write back physical blocks to DASD every time he reads, updates, deletes, or adds a logical record to a VSAM data set.

5.7 DATA PROTECTION FACILITIES

VSAM offers various functions to prevent data from being (a) accessed by unauthorized users, (b) damaged inadvertently or intentionally, or (c) misread by user programs.

5.7.1 Data Protection

Password Protection

VSAM permits the creator of a data set to define four types of passwords:

- **Master password**
  This allows any type of access to the data set and its catalog record: reading, updating, or deletion.
- **Control password**
  This allows access to the control intervals of a VSAM data set.
- **Update password**
  This allows any type of access to data records of a VSAM data set.
- **Read password**
  This allows only reading of a VSAM data set and its catalog record.

Passwords can be assigned to a cluster, the entire data portion, or to any unit of the data portion by DEFINE commands of AMS.

When opening a protected data set, if the user program does not furnish the appropriate password, VSAM requests it from the operator. If neither the user program nor the console operator can furnish the necessary password, this data set cannot be opened.

Authorization routines

When an OS IV/F4 installation is planning its VSAM password protection, it can furnish its own routine to confirm authorizations of requestors. To confirm that a requestor has furnished the correct password, VSAM yields control to this installation-supplied routine when the requestor issues an OPEN macro instruction to this data set.

Control over deletions and updating

Updating or deleting records from a VSAM data set is limited by specifying read-only access to the data set, using the INHIBIT parameter of the AMS ALTER command. Also, **data protection** is supported by the TO parameter of the ALTER command. Date protection is discussed in detail in the **FACOM OS IV/F4 Job Management Functions and Facilities.**

Timestamping

The data and index portions of a VSAM data set can be processed independently. Whenever the content of either portion is updated, this event is posted in the VSAM catalog as a timestamp. Other users of

this data set can thus confirm which data set has been updated and when (date and time).

5.7.2 Data Integrity Facilities

VSAM furnishes the following facilities to prevent damage or loss of data due to carelessness or system errors:

Reduction of overall I/O operations
When processing records, VSAM uses each I/O buffer quite efficiently. After a block of records has been fully processed, writing operations to DASD are minimized.

When inserting, updating, or deleting records, corresponding index records are not altered so long as their control intervals are not sectioned.

By reducing I/O operations in this way, VSAM reduces the incidence of hardware I/O errors while improving the efficiency of DASD accesses.

Preformatted control areas
When a new control area is required for sectioning, VSAM pre-formats this control area before writing records into its control intervals. This helps prevent added records from being lost.

Write validity checking
After writing or updating a record, the VSAM user can request hardware checking (**write validity checking**) to determine whether the record has been correctly stored.

Final-status checking
If a user does not close a VSAM data set in the normal manner after adding/changing records, final-status information about the data set may not be correctly posted in the catalog. In this case, added records may be lost. By issuing an **AMS VERIFY command**, the user can insure that final-status attributes are entered into the catalog whenever a record is added, to prevent its being lost.

Temporarily closing data sets
By issuing a temporary-CLOSE macro instruction (CLOSE TYPE=T) during processing of a data set, the user can insure that all control intervals remaining in its I/O buffers are written onto DASD. Likewise, the catalog entry for this data set is updated whenever a CLOSE (TYPE=T) macro instruction is issued.

Shared and exclusive control of data sets
A data set can be shared or exclusively controlled at three levels, as described in Section 5.6.

5.8 ACCESS METHOD SERVICES (AMS)

Access method services (AMS) is a service program which performs processing on the following VSAM catalog and VSAM data sets:

- Definition, alteration, and deletion of VSAM catalog entries and VSAM data sets.
- Copying existing sequential, indexed sequential, and VSAM data sets into new VSAM data sets.
- Printing VSAM catalogs and VSAM data sets.
- Moving VSAM catalogs and VSAM data sets to other systems.
- Recovery of damaged data sets.

AMS can be invoked either as a separate job step, by an executing batch program, or by a TSS terminal using one of these methods. There are two kinds of AMS commands: functional commands such as defining the VSAM catalog, and modal commands, which condition execution of functional commands.

5.8.1 Functional Commands

There are nine functional commands:

- DEFINE
- ALTER
- DELETE
- LISTCAT
- PRINT
- REPRO
- EXPORT
- IMPORT
- VERIFY

Fig.5.23 shows their broad relationships to catalogs, data sets, and each other.

DEFINE command
Before a VSAM data set can be created, the user must define its catalog, data space, and other attributes with a DEFINE command.

Definition of a catalog
Before creating any VSAM data sets, an installation must define a master catalog. As necessary, users can define additional catalogs of their own.

Definition of a data space
With a DEFINE command, the user secures a VSAM data space, whose attributes are registered in the catalog.

Definition of a VSAM data set
Attributes of a VSAM data set are defined by AMS as follows:

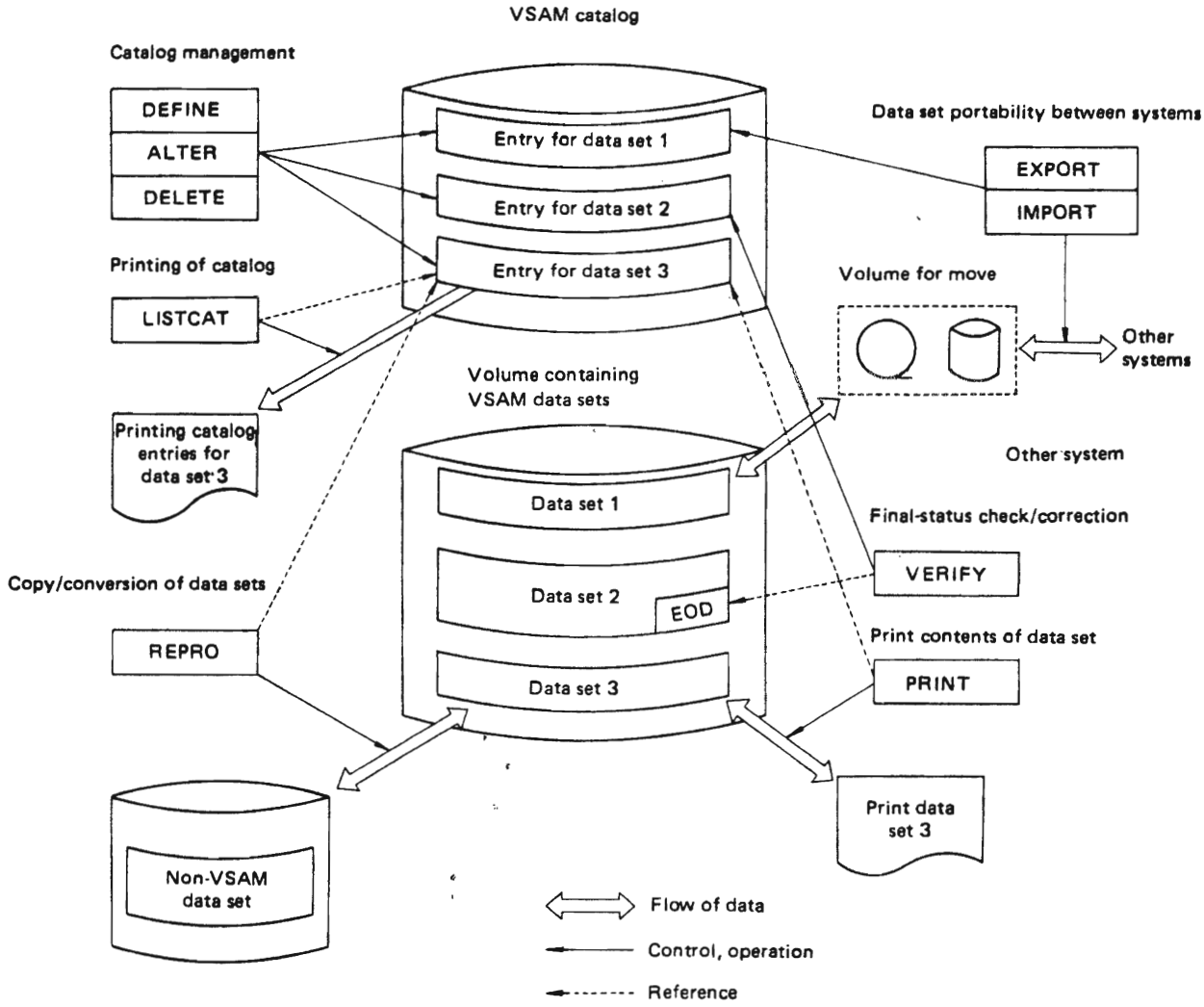


Fig. 5.23 Schematic diagram of AMS commands

- Average and maximum record lengths.
- Information about keys.
- Catalog name.
- Volume serial number(s)
- Space requirements.
- Minimum I/O buffers for accessing the data set.
- Replicated/unique index records.
- Sequence set in index or data portion.
- Whether index and data portions are to be created on separate volumes.
- Length of control intervals.
- Free space reservations.
- Integrity and protection options.
- Write validity checking.
- Preformatted control areas.

The creator of a new VSAM data set can reference a previously-defined VSAM data set as a **model**; in this case, furnishing its attributes becomes very simple. An installation can catalog non-VSAM data sets into its VSAM catalog; in this case only volume pointers are retained in the catalog.

ALTER command

An ALTER command changes selected attributes of a VSAM data set (which were originally defined by DEFINE command).

ALTER has the same parameters and format as DEFINE; most attributes can be altered. However, control-interval length, location of the sequence set, and other fundamental format attributes cannot be altered. To change these, the user must copy the data set to a new DASD location.

DELETE command

This command deletes a specified catalog entry; released space can be immediately used for other data sets.

LISTCAT command

This command displays all entries of the catalog — or a subset specified by the user. If he furnishes an entry name, only the corresponding entry is printed; if he furnishes an entry type, all corresponding entries are printed.

LISTCAT will also display selected fields of each entry. Since the password field is itself password-protected, LISTCAT will not print passwords unless the master password is furnished with this AMS job.

REPRO command
With a REPRO command, a user can copy a sequential or indexed sequential data set to a VSAM data set. Likewise, he can copy a VSAM data set to a new sequential data set. He can copy a VSAM data set, reorganizing its free space automatically in the new data set.

PRINT command
With this command, a user can print part/all of a sequential, indexed sequential, or VSAM data set. For a KSDS, he can print the index separately from the data portion. Sequential data sets and ESDS are printed in physical sequence; indexed sequential data sets and KSDS can be printed in either physical sequence or key sequence.

EXPORT command
When a VSAM data set must be carried (**exported**) to another system, the user issues an EXPORT command to AMS. Both the data set and its catalog information are moved to the mountable volume which contains the exported data set.

- Moving by catalog entry
  When a VSAM data set is moved, the corresponding user catalog may optionally be removed from the master catalog of the exporting system. This catalog is **imported** by the receiving system, together with one or more data sets. In the importing system, this catalog is linked to its master catalog by an IMPORT command.
- Moving by volume
  A user can copy a VSAM data set onto magnetic tape or DASD as a sequential data set. At the same time, he can create a copy of the catalog entry for this data set, which is moved to the new VSAM data set on the importing system.

IMPORT command
IMPORT commands complement EXPORT command as follows:

- Moving by Catalog Entry
  Since both the user catalog and its VSAM data sets can be used without change by the importing system, the IMPORT command merely adds a pointer to the VSAM master catalog of the importing system.
- Moving by volume
  In this case, IMPORT creates new catalog entries in the master catalog of the importing system. IMPORT then creates VSAM data sets from the sequential data sets being imported.

By utilizing IMPORT and EXPORT commands, an installation can conveniently backup its VSAM data sets and VSAM catalog onto tape or DASD.

VERIFY command
The last record of a VSAM data set is recognized by its EOD parameters (highest key and highest RBA), recorded in its catalog entry. Also, VSAM writes a hardware EOF record at the end of the data set.

If an unlikely error occurs during CLOSE processing of a VSAM data set, the position of the hardware EOF record may disagree with the EOD parameters. The VERIFY command checks such EOD contradictions; it corrects EOD information in the VSAM catalog if it finds an inconsistency.

5.8.2 Modal Commands

AMS offers IF, DO-END, and SET commands for controlling the sequence of functional commands. The PARM parameter controls the length of input commands and various AMS outputs for each run.

- IF command
  This controls the execution sequence of functional commands, based on condition codes from each AMS operation.
- DO and END commands
  These delimit the start and end of a **command group,** as defined in the **FACOM OS IV/F4 VSAM Functions and Facilities.**
- SET command
  This command alters an AMS condition code.
- PARM parameter
  The PARM parameter defines the length (other than the default value) for input commands. It also can set special Universal Character Set (UCS) values for printing outputs.

CHAPTER 6
DATA COMMUNICATIONS

6.1 OVERVIEW

The growing complexity of modern data communications networks has made control functions such as scheduling data flows and managing system resources — terminals, telephone lines and modems, computers, and communications control units — critical to successful operation of such networks.

OS IV/F4 furnishes the virtual telecommunications access method (VTAM) as its principal access method for data communications. VTAM operates in conjunction with the network control program (NCP), which operates within a communication control processor (CCP), a new and independently-programmable control unit for M series computers. The CCP architecture permits NCP to assume many functions formerly performed by host computers. This host-CCP architecture enhances overall system reliability and efficiency.

VTAM is an access method applicable to various system configurations: terminals and computers linked via telephone circuits, character display units connected directly to host-computer channels, and linkages between computers utilizing **channel-to-channel adapters** (CTCA).

Major functions of VTAM are as follows:

- Connecting and disconnecting application programs from terminals.
- Data transmission between application programs and terminals.
- Sharing of network resources such as the CCP, communication lines, and terminals.
- Multiple application functions from each terminal; for example, used for inquiries in the morning — order entry in the afternoon.
- Monitoring and reallocating resources during routine operations.

This chapter outlines VTAM and NCP from the viewpoint of installation management. Additional details can be found in the following publications:

- **FACOM OS IV/F4 VTAM Functions and Facilities.**
- **FACOM OS IV/F4 VTAM Macro Instructions Reference Manual.**
- **FACOM OS IV/F4 VTAM Generation User's Guide.**
- **FACOM OS IV/F4 VTAM Operator's Guide.**
- **FACOM OS IV/F4 NCP Functions and Facilities.**
- **FACOM OS IV/F4 NCP Generation User's Guide.**

6.1.1 Purpose of VTAM

A communication control processor is necessary to connect communication lines to a computer. There are many kinds of computer network connections with respect to speed and transmission control procedures.

It is desirable to use the host computer for data processing and noncommunications input/output functions as much as possible, leaving most communications control tasks to the CCP.

Communications control is logically classified into four levels: transmission control, network control, path control, and format control.

The **transmission control** level manages transmission control procedures, polls and selects contiguous stations, controls communications equipment such as telephone circuits and modems, and performs error detection and correction and various types of error recovery.

The **network control** level performs

- selection of path from sender to receiver,
- establishment of network communication paths,
- retry using alternate paths,
- translation and editing of data transmitted via other paths, and
- dialogs with other data networks.

The **path control** level controls logical communication between computers and terminals. It establishes and releases logical communication paths.

The **format control** level provides blocking/deblocking, linefeed, control of displays, and cursor control.

OS IV/F4 VTAM interfaces with the NCP stored in

the CCP. this VTAM-CCP combination furnishes the following advantages:

- shares processing loads to increase host-computer efficiency.
- simplifies interfaces between terminals and the host computer.

6.1.2 Usage of VTAM

Several major OS IV/F4 subsystems utilize VTAM for all of their communications to remote terminals:

- Time sharing system (TSS)—described in Part IV of this manual.
- Remote entry services (RES)—described in Chapter 3 of this part of the manual.
- Advanced information manager (AIM) — described in Section 4.11 of Part 1 of this manual.

In addition, users can write their own terminal subsystems using VTAM macro instructions; comparable communications verbs are available in high-level languages such as COBOL and PL/I. OS IV/F4 subsystems and user programs containing VTAM macro instructions (or equivalent) will be collectively called **VTAM applications programs** in this chapter, as depicted in Fig. 6.1.

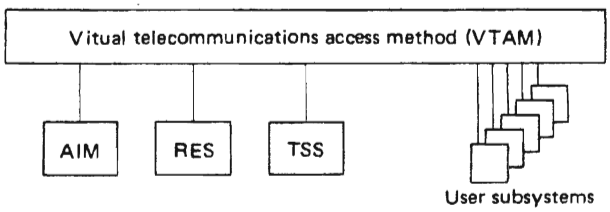


Fig. 6.1 VTAM applications programs

6.1.3 Network Structures

A network consists of one or more central computers and zero, one, or more terminals. Each central computer or terminal comprises one or more nodes from a communications-architecture viewpoint. Nodes comprise the following elements and combinations thereof:

- application program.
- communications control processor.
- communications link.
- terminal (station).
- local I/O device for keyboard users.

Fig. 6.1 illustrates a representative network of nodes. In this chapter, terminals and local I/O devices for keyboard users will be called "terminals" collectively, since they are controlled by VTAM in essentially the same way.

Each **station** itself comprises a node. The configuration of a station node differs according to its hardware characteristics and its control by the host computer. A station node can be one of the following three types.

- Cluster station.
- Terminal station.
- Component.

A **cluster station** consists of several devices sharing a control unit. The host computer can communicate with some or all of them simultaneously, recognizing each device separately. In general, each device is separately operated, calling its own application programs independent of the other devices. In the **polling method** of communication, the host computer identifies each input device by the contents of its message texts. In **contention method,** the host identifies each device by the content of a data field sent by the station with each transmission request. Every cluster station sets its own address into a specified position in each message.

The F1530 banking terminal, F9525 character display terminal, and F9520 character display terminal are representative cluster stations.

Each **terminal station** can receive input from only one source at a time. Likewise, each can write only one stream at a time to a host computer. Terminal stations cannot recognize which input device receives each message. The operator typically uses one input or output device at a time, and only one application program can communicate with a terminal station at a time.

Each device in a cluster station is itself the body of a terminal; F1520 teletypewriter/paper tape unit, U-200 intelligent terminal, and F1510 teletypewriter/paper tape unit are representative devices and terminals.

A **component** is a terminal that can select an output device according to an address transmitted by a host. Paper-tape punches and printers in the F1520 and U-200 terminals are components. The F1510 terminal has no component capability; its output devices are selected solely by the station operator, not by control characters from a host. Fig. 6.3 illustrates the above types of stations and devices.

6.1.4 VTAM Terminals

Terminals supported by VTAM and NCP can be any of the following types:

Private line (point to point)

- asynchronous.
- synchronous.
- polling/addressing.
- contention.
- half duplex.
- full duplex.

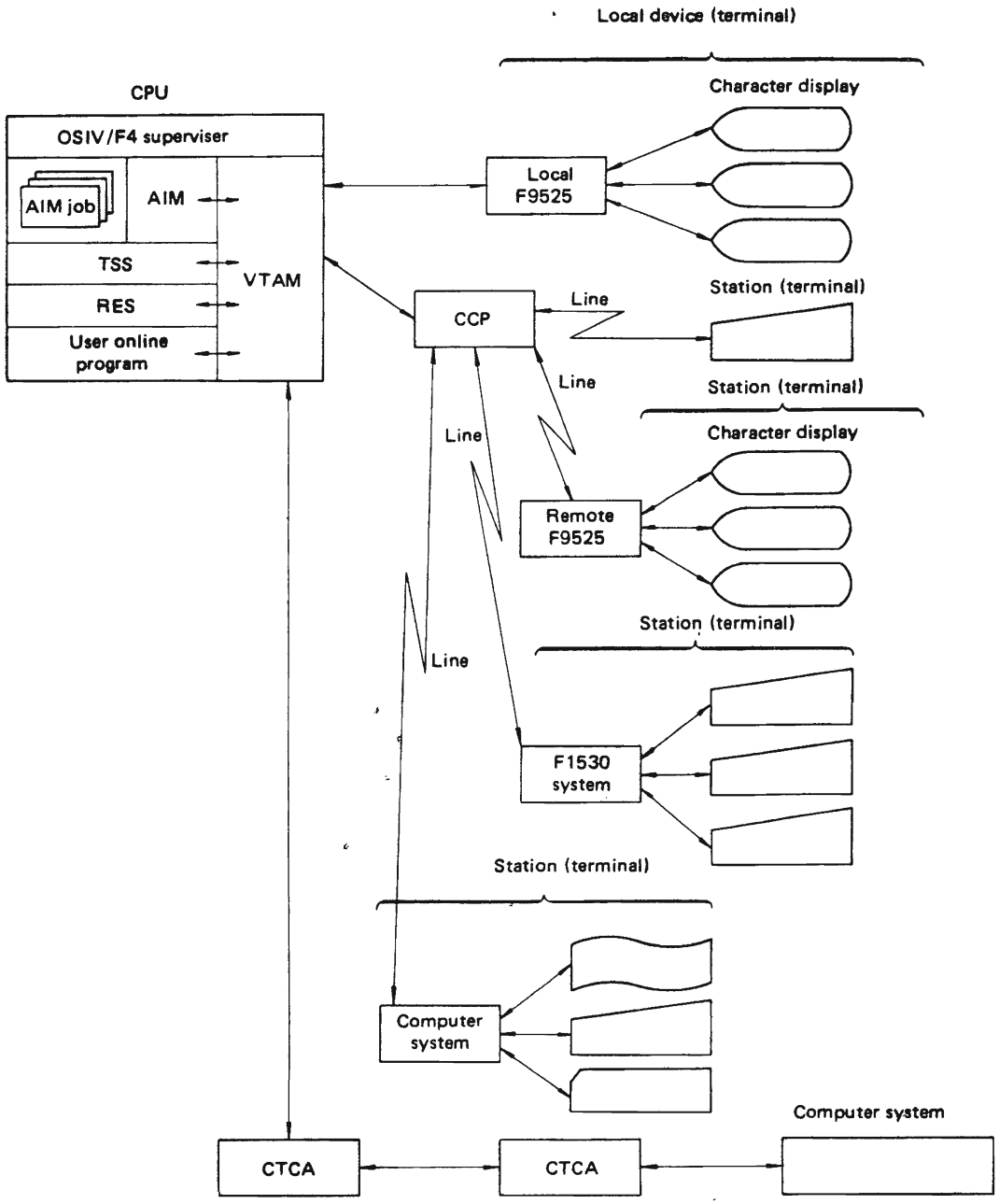


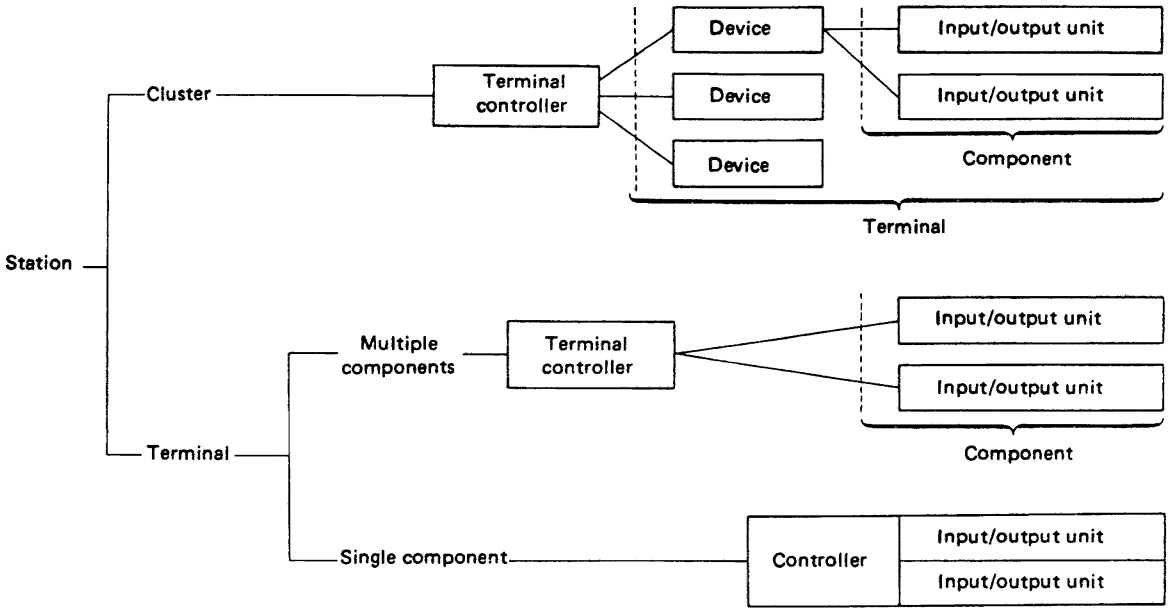
Fig. 6.2 VTAM network configuration

Dialup (switched)
- synchronous.
- synchronous.
- contention.
- half duplex.

Noncommunications
- F9525 character display unit.
- channel to channel adapter (CTCA) link.

6.2 VTAM FACILITIES

VTAM furnishes many macro instructions and operator commands. Application programs use the macro instructions (or higher-language equivalent verbs) to accomplish various functions described later in Section 6.4. The current section describes overall functional capabilities and operations of VTAM.



FUJITSU FACOM STATIONS

FACOM models	Station type	Component	Input/Output device
F9525 Character display	Cluster	Multiple	CRT, KB
F1530 Banking terminal	Cluster	Multiple	KB, CP
F9520 Character display	Cluster	Multiple	CRT, KB, LP
F1520 Teletype-writer	Cluster	Multiple	KB, CP, PT, LP
U-200 Intelligent terminal	Cluster	Multiple	KB, CR, LP
F1510 Teletype-writer	Terminal	Single	KB, CP, PT

Key: CR – Card reader; LP – Line printer; CRT – Display; KB – Keyboard; CP – Character printer; PT – Paper tape read/punch

Fig. 6.3 Types and components of VTAM network

6.2.1 Sharing Network Resources

VTAM manages network resources (control units, lines, and terminals) so that several application programs and dozens/hundreds of terminals can simultaneously share these resources, each unaware of activity by the others.

Sharing control units

Representative shared control units are the CCP itself and the control unit for F9525 character displays. Several application programs can simultaneously share each of these control units, as shown in Fig. 6.4.

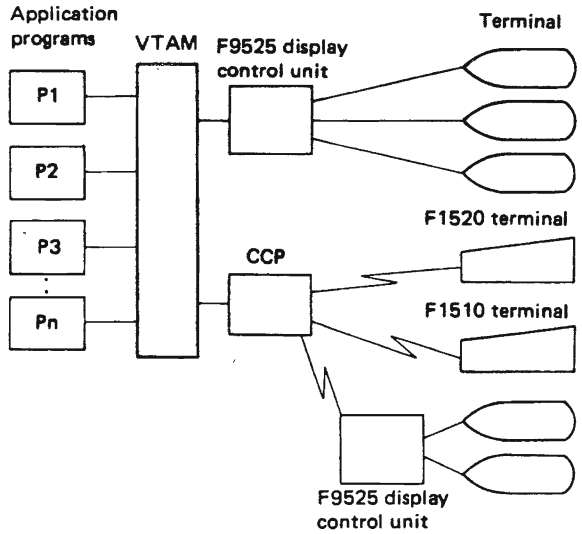


Fig. 6.4 Shared control units

Sharing lines

Several application programs can share one multidrop line. Each terminal can communicate with a different application program.

In Fig. 6.5, terminals T2 and T3 use application program P1, terminals T1 and T4 use program P2.

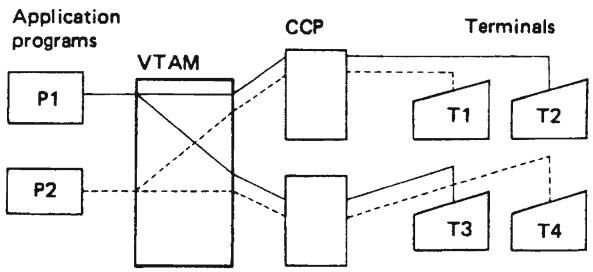


Fig. 6.5 Example of sharing lines

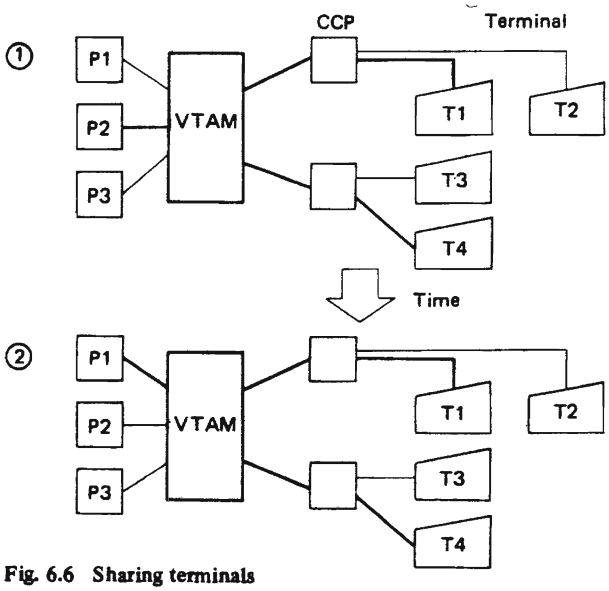


Fig. 6.6 Sharing terminals

Terminals on the same multidrop line can thus use different application programs.

Sharing terminals

In OS IV/F4, all terminals access application programs via VTAM. Hence, several application programs can simultaneously be accessed by certain types of terminals. However, once a terminal commences communication with one application program, it cannot communicate with another program until it finishes (normally or abnormally) with the first.

In Fig. 6.6, terminals T1 and T4 use application programs P2 at timepoint 1 , program P1 at time point 2 .

6.2.2 Establishing Communications Links

Communications links are necessary to establish network paths. A **connection** is a link between a terminal and an application program. Communications connections are established and used in the following stages:

- Connection is requested. A communications path is furnished by OS IV/F4 and NCP if the required node (application program or terminal) is a available.
- Data is transfered if a connection is successfully established.
- The connection is released when communication has been completed.

Initially, all terminals in a network are available to VTAM; an application program can request connection to any VTAM terminal. A connection can be requested by an application program, a terminal, or the center site operator.

A connection is established when VTAM makes a path between the application program and the terminal available for messages. The terminal can then communicate with one application program. If the application program requests VTAM to disconnect this terminal, VTAM makes its path temporarity unavailable. The disconnected terminal can then request linkages to other application programs as appropriate.

VTAM can connect dynamically to terminals. There are two ways to establish connections: to acquire or to receive a connection.

When VTAM connects an application program to a particular terminal which the former has requested, the application program **acquires** the terminal. Conversely, when a terminal successfully requests connection with an application program via VTAM, the terminal **receives** the connection.

Other application programs cannot connect to a terminal already connected with one application program; however, they can enqueue their requests for linkage. During execution, an application program

can not only release a terminal to another application program but also connect to another terminal dynamically.

6.2.3 Data Transmission

Once an application program connects to a terminal, it can communicate with the terminal using one of the VTAM options. If the program transfers data using standard VTAM, it need provide only data records and no transmission control characters. The program need not translate data into a transmission code; it presents EBCDIC data to VTAM. Likewise, VTAM transfers data from a terminal to an application program in EBCDIC; translation and deletion of transmission control characters by the program are unnecessary, since these are all handled by VTAM.

Another option is to incorporate a user program into the NCP for independent user processing, performed through VTAM jointly with this NCP.

6.2.4 SOLICIT Macro Instruction

In a communications network the host computer must be aware of the configuration of lines and types of terminals. The host computer must always be prepared to receive data from terminals. To prepare for such data, the host computer uses VTAM and NCP via a SOLICIT procedure. User programs need not be aware of SOLICIT procedures.

SOLICIT reads data into a VTAM buffer from the network prior to its being required by a program. An application program issues a SOLICIT macro instruction, then transfers data from a VTAM buffer to its work area by issuing a READ macro instruction.

Via SOLICIT, an application program can receive data from any terminal ready to transmit data without specifying which one. After the application program issues a SOLICIT macro instruction, its read macro instruction acquires data from any/all terminals currently connected to this program.

6.2.5 Network Solicitor

Terminals can be shared by requesting dynamic connections, increasing the efficiency of the network. The following functions are necessary:

- Manage terminals so that their linkage requests can be serviced.
- Discriminate data inputs by terminal, i.e., by names of corresponding application programs.
- Transmit connection requests to application programs.

VTAM provides a **network solicitor** routine to monitor connection requests from terminals. That is the network solicitor controls all local and remote terminals not dedicated to specific application programs.

The network solicitor is not necessary when connecting via LOGON, as described in Section 6.4.3. This option is selected at system generation.

When VTAM starts, the network solicitor connects any terminal for which automatic LOGON was specified. The solicitor controls these terminals until connection is actually requested by LOGON or an application program. This function can be performed by a user-written solicitor program or directly by an application program. Unless otherwise specified, the following discussion assumes the standard OS IV/F4 network solicitor is used.

6.2.6 Exit Routines

An installation can add exit routines to VTAM for functions such as the following:

- simplify applications design.
- increase system efficiency.
- dynamic linkages.
- check validity of application-program VTAM interfaces.
- accounting.

There are two types of exit-routine interfaces:

- for a particular program.
- for an installation-standard routine, specified when VTAM is defined at system generation.

Application-program exit routines

VTAM provides an innovative and asynchronous service to exit routines.

Processing several unrelated tasks concurrently with a program is troublesome to program and often inefficient. OS IV/F4 and VTAM avoid these problems and furnish concurrent processing of tasks as follows:

VTAM relieves user programs from synchronizing unrelated subtasks by providing asynchronous exits for such subtasks. That is, an application program can designate one or more routines to process specific asynchronous tasks. There are two interface formats for exit routines: explicit READ, WRITE, SOLICIT macro instructions; and specification by an exit list (LOGON exit, error exit, etc.).

In both cases, exit routines are scheduled asynchronously. For example, exit routines to schedule subtasks can be issued whenever an I/O operation is completed; hence, a program can optionally issue concurrent I/O requests. When data satisfying a VTAM request is received, OS IV/F4 schedules the exit routine to process the data, then returns control to the interrupted program, and the latter resumes execution. By this method, a program can accept data from a terminal and process it simply and efficiently.

When creating line control procedures, an installation should define a hierarchical structure to process several terminals concurrently. In general, each level is dispatched by the line control program whose exit routine can reduce the time to switch tasks, thereby increasing performance.

Fig. 6.7 is an example showing a conversational inquiry task and its associated exit routine. ("RETURN" is a supervisor macro instruction returning to the interrupted program.) If no exit routine is provided, the application program must search a multiple-events ECB list for completion times of each event, then schedule processing routines appropriate to the requests.

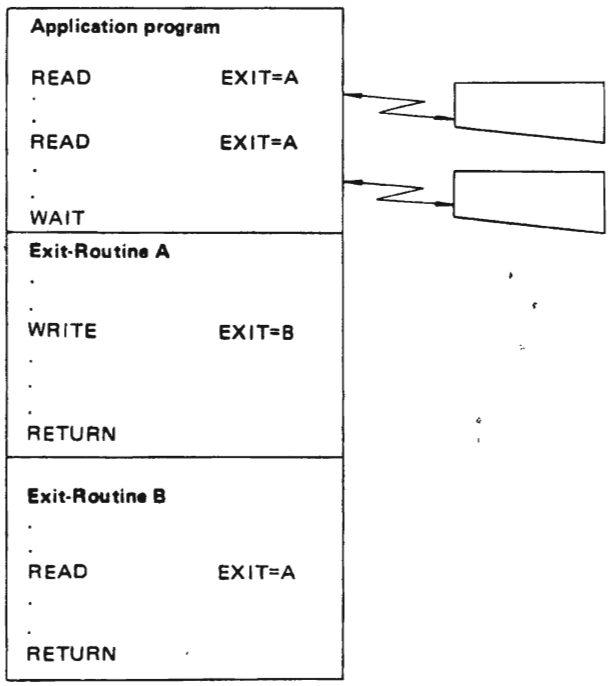


Fig. 6.7 A typical exit routine

In addition to passing control to OS IV/F4 for each access request (such as a READ or WRITE macro instruction), LOGON exit routines also provide dynamic linkage, as shown in Fig. 6.8.

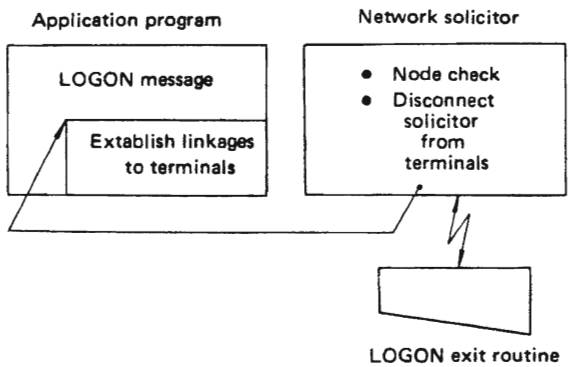


Fig. 6.8 LOGON exit routine

Installation-supplied exit routines

Installation-supplied routines can be included with VTAM during system generation.

A default exit routine will be included by OS IV/F4 if the installation does not furnish its own routine.

Control is passed to an installation-supplied exit routine synchronous with VTAM processing. Two types of these exit routines are:

- Authorization checking exit routine VTAM yields control to this exit routine just before linking a terminal to an application program. If the authorization level of this terminal is insufficient, VTAM transmits it an appropriate return code and cancels the connection request. In some cases, the LOGAN exit routine will check any necessary authorizations.
- Accounting exit routine
  VTAM passes control to this routine just after linking a terminal to an application program and when a linkage is released.

6.3 DEFINITION OF A VTAM NETWORK

The console operator (or an automatic command procedure) issues a START command to initiate a subsystem using VTAM. The OS IV/F4 initiator creates a VTAM address space. Various parameters are then red — and optionally modified by the console operator — to condition the structure and operation of this network.

Preceding start-up of a VTAM subsystem, each installation must perform a VTAM system generation and an NCP generation, define its network, initialize VTAM, and perform any necessary modifications of the hardware/software configurations. These topics are briefly described in the following sections.

6.3.1 System Generation

Each installation defines four elements of VTAM during system generation:

- VTAM modules
  With the standard network solicitor, these programs (defaults supplied by OS IV/F4) are link-edited into appropriate system data sets.
- Interface to the communications control processor (CCP)
  Specifies which host-computer channels and subchannels are connected to the CCP.
- Interfaces to local terminals
  Specifies which channels and subchannels are connected to local terminal-type devices, e.g., F9525 character displays.
- VTAM system data sets
  During system generation, DASD space is reserved for SYS1.VTAMLIB, SYS1.VTAMLIST, and SYS1.VTAMOBJ data sets.

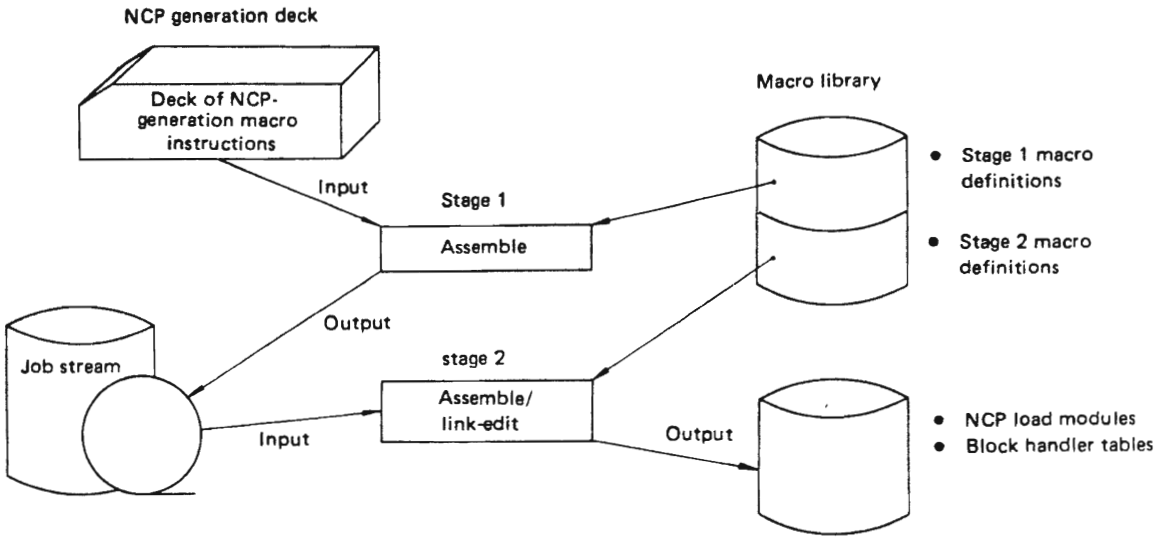


Fig. 6.9 NCP generation

6.3.2 Generating a Network Control Program (NCP)

Stage 1 of NCP generation assembles NCP-generation macro instructions. Stage 2 assembles and link-edits the job stream prepared by Stage 1. Stage 2 creates NCP load modules and a block handler table, as shown in Fig. 6.9.

NCP-generation macro instructions are used both for generating the NCP itself and for defining the VTAM network. Any VTAM macro instructions included in the NCP generation deck are ignored; however, they later become part of the NCP definition deck for defining the VTAM network. Table 6.1 shows which NCP generation macro instructions are used by VTAM, only by NCP, or for generating both subsystems.

6.3.3 Defining a VTAM Network

Each installation furnishes network-definition source modules (**definition decks**), cataloged in SYS1.VTAMLST by member name. SYS1.VTAMLST is accessed immediately after the VTAMSTART command; hence, SYS1.VTAMLST must exit before the operator issues his first START command after initially loading OS IV/F4 (IPL).

There are four members of SYS1.VTAMLST: NCP definition deck, local-terminals definition deck, application-programs definition deck, and a collection of VTAM parameters.

The system programmer can create or modify members of SYS1.VTAMLST with the JSEUPDTE utility program.

Definition of an NCP

The **NCP generation deck** can also be used as the NCP definition deck. Therefore, a distinct **NCP defenition deck** need not be created in SYS1.VTAMLST to define the VTAM/NCP interface. The member name of the NCP definition deck should be the same as for the NCP load module.

At least one NCP must be defined for each CCp. However, it is often useful to define several NCP's, to reflect different hardware configurations or different modes of using the network.

Definition of local terminals

Each local terminal – one directly channel-connected to the host computer rather than via a CCP – is defined by a LOCAL macro instruction.

A **local-terminals definition deck** consists of one or more LOCAL macro instructions, read just after the operator issues a VTAM START command; from it, VTAM creates a control table which defines individual local terminals or groups of such terminals. Selection of a local-terminals definition deck can be modified by a parameter just after the START command.

Definition of application programs

Each application program or program group is defined by an APPL macro instruction or VTAM-created control block. An **application-programs definition deck** consists of one or more APPL macro instructions, entered as a member of SYS1.VTAMLST and read just after the VTAM START command.

Definition of LOGON requests

A **LOGON characteristics table** (LCT) control block specifies types of LOGON requests in a VTAM network. The LCT is defined by a LOGON or LOGCHAR macro instruction and read by the network solicitor with an INTERPRET macro

Table 6.1 Macro instructions to define an NCP

Macro instruction	Function(s) of macro instruction	VTAM	NCP
PCCU	• CCP unit address for host computer • Selection of VTAM functions: a. Whether take check points b. Whether to perform CCP initial test *1	x	
BUILD	• Name of NCP load module • Characteristics of NCP and CCP	x	x
SYSCNTRL	Dynamic control function in NCP *2	x	x
CSB	Type and characteristics of line scanning *3		x
LINELIST	Definition of a logical *4	x	x
HOST	• VTAM buffer size • Number of buffers allocated by VTAM when data is received from CCP • Offset of buffer-header prefix used by VTAM *5	x	x
SERVICE	Definition of service sequence table *1		x
GROUP	Definition of physical group	x	x
LINE	Characteristics of one line	x	x
CLUSTER	Characteristics of a cluster-type control unti (e.g., F9525)	x	x
TERMINAL	Characteristics of a terminal	x	x
COMP	Characteristics of a component	x	x
STARTBH	Start of a block handler *1		x
ENDBH	End of a block hander		x
DATETIME	Whether NCP should timestamp data blocks	x	x
EDIT	Whether NCP should delete data indicated by backspace characters		x
UBHR	User-written block handler is furnished		x
BHSET	Definition of block handler set *6		x
GENEND	End of the NCP generation input deck	x	x

*1 See Section 6.6 for a description of the Network Control Program.
*2 Dynamic control function serves to indicate if modules dealing with sessions and/or the different types of display functions are to be incorporated in the NCP or not. (See 6.6 NCP)
*3 Communication scanner mechanism indicates the CS. (See **FACOM M-190 Hardware Function Specifcs.**)
*4 For exchanges between VTAM and NCP, access can be done in logical communication group units.
*5 Prefix area used for buffer queuing of header areas in the buffer employed for transfer purposes between VTAM and NCP.
*6 User-generated block handling function can be included. This function can carry out specialized processing (handling of codes, insertion of specific messages, etc.) which are dependent on terminals.

instruction which is used to acknowledge LOGON requests and to invoke appropriate application programs.

LOGON and LOGCHAR macro instructions are related to terminals by TERMINAL macro instructions – among the NCP-definition macro instructions – and LOCAL macro instructions. They are associated with application programs by the APPL macro instructions.

LOGON request definitions should be entered as a member of SYS1.VTAMLST but are not required for terminals not logging onto applications programs or if standard LOGON formats are used.

VTAM parameters

Parameters interpreted when VTAM is started determine its initial status; they should be entered into SYS1.VTAMLST as 80-byte card images as follows:

- definition of the initial configuration of NCP, local terminals and application programs.
- VTAM buffer sizes.
- number of VTAM buffers.
- entry-point name of the Network Solicitor.
- number of control blocks for access requests.
- number of control blocks for scheduling exit routines.
- number of control blocks for linkages.

An installation can define several different collections of VTAM parameters, each comprising a member of SYS1.VTAMLST whose name may be specified when VTAM is started.

A typical procedure for defining a network appears in Fig. 6.10.

Table 6.2 Points where a VTAM network can be defined/modified

Type of modification \ Point of generation/modification	System generation	NCP generation	Creation of a VTAMLST member	START VTAM	Operator commands: VARY, MODIFY, etc.
Addition of a CCP	x	x	x		
Addition of a NCP-definition deck *		x	x		
Update of a NCP-definition deck		x	x		
Deletion of a NCP-definition deck			x		
Selection of a NCP-definition deck				x	
Activating/deactivating NCP resources					x
Addition of a local terminal	x		x		
Addition, deletion, or updating of a local-terminals definition deck **			x		
Selection of a local-terminals definition deck				x	
Activating/deactivating a local terminal					x
Addition, deletion, or updating of an application-programs definition deck			x		
Selection of an application-programs definition deck				x	
Reload the NCP					x

* It is impossible to add terminals or other equipment to a CCP which was not defined during system generation.

** It is impossible to modify a VTAM network to include local terminals not defined druing system generation.

6.3.4 Initializing and Modifying a VTAM Network

Section 6.3.3 described how a VTAM network is defined during system generation and NCP generation. The network can also be modified when VTAM is started and thereafter by central-site operator commands, as summarized in Table 6.2.

6.4 OPERATING A VTAM NETWORK

This chapter describes execution of application programs within a VTAM network. The reader will recall that "application program" is used broadly in this chapter to include RES, TSS, and AIM as well as installation-developed subsystems using data communications.

Each application program executes in the following sequence:

- start-up of VTAM.
- start-up of the application program.
- connection of the application program to one or more terminals.
- data transmission between the application program and the terminals.
- release of connections between the application program and the terminals.
- application program termination.
- VTAM shutdown.

6.4.1 Start-up of VTAM

When the operator issues a START command for VTAM, OS IV/F4 allocates an address space and starts the VTAM initialization routine (KCB CPCMD). VTAM determines its initial network configuration, initializes various tables, acquires buffers, and starts a VTAM command procedure and the network solicitor. The NCP is loaded into the CCP if the latter is included in the network.

The cataloged procedure to start VTAM must be in SYS1.PROCLIB; its contents are as follows:

1) EXEC statement
   PGM=KCBCPCMD
   REGION=VTAM-region-size

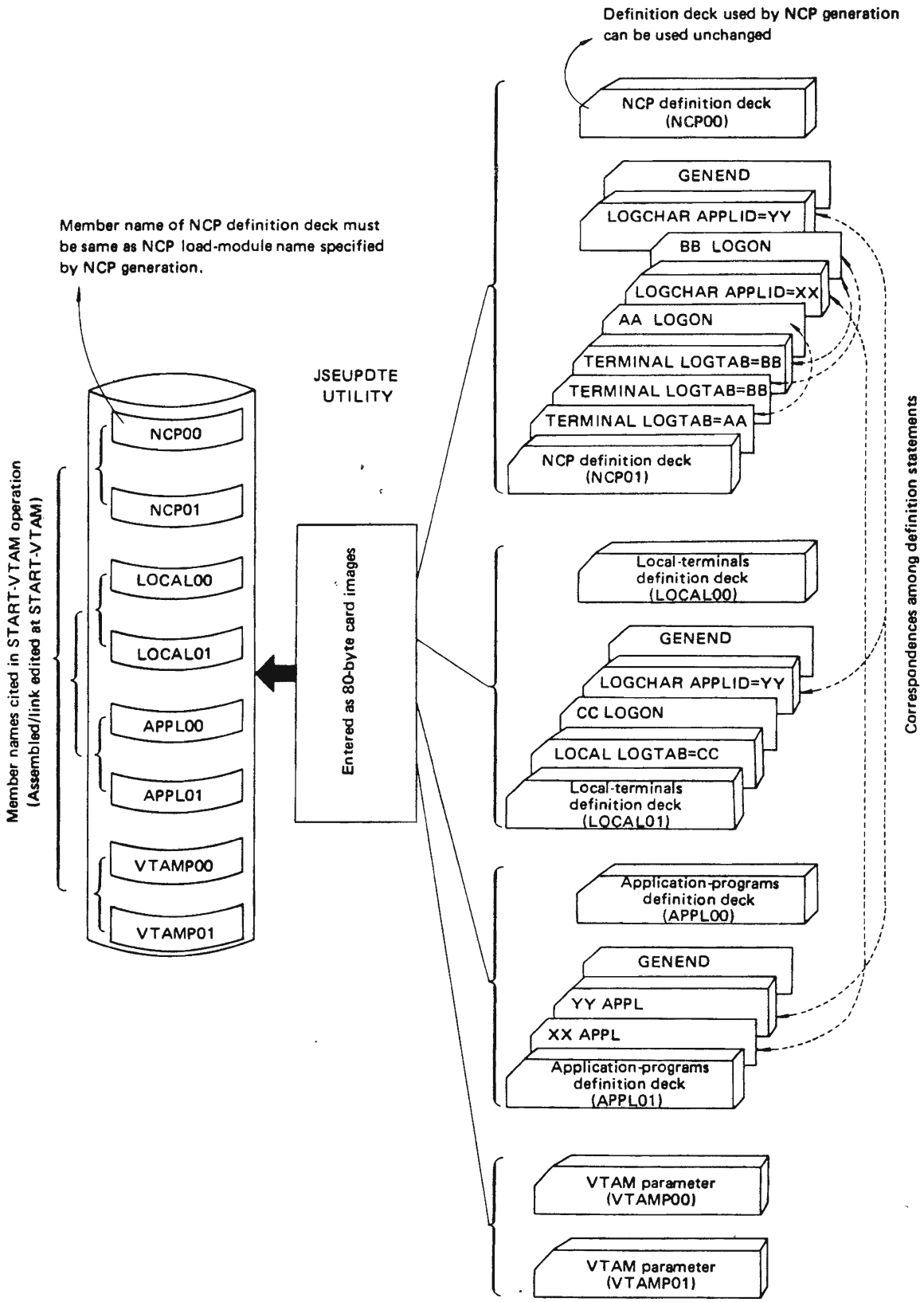
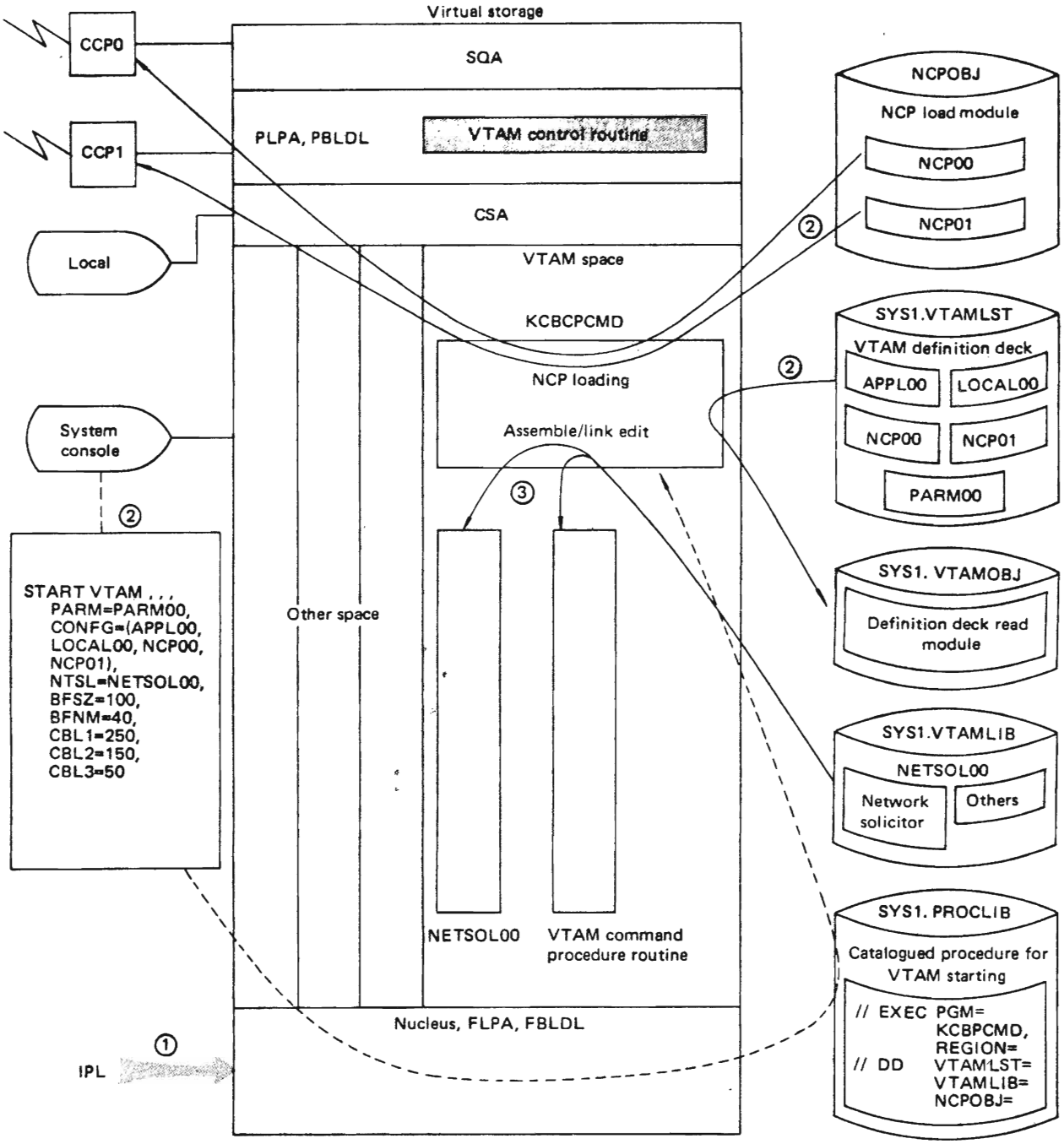


Fig. 6.10 Example of defining a VTAM network



① All basic OS IV/F4 components are determined when it is loaded (IPL). In this figure, the VTAM control routine is added to the LPA (Link Pack Area).

② After the START command for VTAM, the KCBCPCMD program performs a series of initialization processes, such as assembly and linkage editing of the VTAM definition deck, loading of the NCP, and acquisition of buffer pools.

③ The KCBCPCMD program also loads the Network Solicitor and VTAM command procedure routines. Thereafter, VTAM is ready to receive commands, to accept requests for the application programs from terminals, and to manage transmit/receive requests from terminals.
Each application program is started by a terminal, using ordinary JCL statements. Application programs already executing may link to VTAM by OPEN macro instructions issued after VTAM has been successfully started.

Fig. 6.11 Example of starting VTAM

An application program can use two or more ACBs; however, each ACB should correspond to different APPL entry. An ACB can also be dynamically created by a GENCB macro instruction during execution.

VTAM exit list

An EXLST macro instruction creates a list of exit-routine addresses. Operands of this macro instruction indicate characteristics of exit routines to which VTAM passes control:

LERAD	exit routine for logic errors in linkage requests or I/O requests issued by the application program.
SYNAD	exit routine for an unrecoverable I/O error.
ASYIP	exit routine entered if a block of data was read into a VTAM buffer after a SOLICIT macro instruction.
TPEND	exit routine entered when a console operator issues a HALT command, or if VTAM completes normally or abnormally.
RELREQ	exit routine to which control is passed if another application program requests linkage to terminals linked to the current program by OPNDST or SIMLOGON macro instructions.
LOGON	exit routine for queued LOGON requests for application program.
LOSTERM	exit routine to which control is passed when the console operator releases a terminal linked with this application program by issuing a VARY command.

6.4.3. Connecting Application Program to a Terminal

An executing application program can link to VTAM by issuing an OPEN macro instruction. At this point, VTAM still controls the terminal, whether the latter is remote or local.

There are four methods for VTAM to connect a terminal with an application program:

- LOGON from the terminal.
- LOGON from a network console.
- LOGON from the application program.
- Acquisition of the terminal by application program.

Ineach mothod, connection of the application program to the terminal is completed when the application program issues an OPNDST macro instruction.

LOGON from the terminal

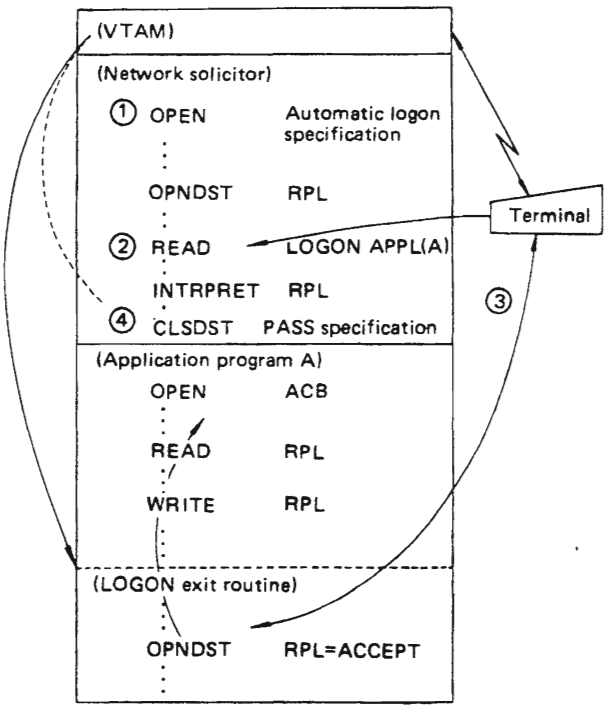
This method links an application program with a terminal by the latter's specifying the program name in a LOGON command. LOGON commands are controlled by the network solicitor specified in a parameter of the START-VTAM command.

Whether LOGON commands are controlled by the network solicitor depends on whether LOGON requests (by LOGON and LOG CHAR macro instructions) are defined for this network. If LOGON is requested by a terminal, VTAM schedules the LOGON exit routine if the ACB points to a LOGON exit routine.

Connection is completed when the programmer issues an OPNDST macro instruction in his LOGON exit routine, using RPL (request parameter List) and NIB (node initialization block) macro instructions specifying specifying "ACCEPT."

Each RPL or NIB can be created by an RPL or NIB macro instruction, respectively. Information necessary for linkage can be obtained from an RPL. Corresponding to each VTAM terminal entry, a NIB contains the name of the terminal to be linked and other information. Characteristics of terminals can be queried by issuing INQUIRE macro instructions during the LOGON exit routine. When connection with terminal is established (RPL=ACCEPT is specified in a corresponding OPNDST macro instruction), data transmission can begin, information in the NIB is moved to an internal table of VTAM.

Thereafter, the application program NIB is no longer used as an information source.



① During definition of a network, an installation may specify that certain LOGON requests are issued by OPEN, even if no command is requested from terminal. Such a LOGON request is called an **automatic LOGON request.**

② INTRPRET macro instruction checks the validity of an application program from the LOGON message transmitted by a terminal.

③ Application program A is linked with the terminal by an OPNDST macro instruction specifying "ACCEPT".

④ A CLSDST macro instruction specifying "PASS" indicates that the terminal requesting linkage is to be accepted by the specified application program.

Fig. 6.12 LOGON from a terminal

LOGON requests ffrom a terminal can be queued within VTAM through an ACB option specified by an application program. This queuing can be halted by a SETLOGON macro instruction.

RPL macro instruction

An application program should furnish an RPL to request connection or an I/O operation to one or more terminals. Each RPL creates a control block used by an application program to describe a processing function to VTAM. For example, specifying an RPL ina READ macro instruction indicates to VTAM which terminal to read, where to store input data, how to notify the application program at the end of each application, and how to process each reuqest.

An RPL can be modified not only by OPNDST, READ, and WRITE macro instructions but also by a MODCB macro instruction. An RPL can also be created by a GENCB macro instruction during program execution.

NIB macro instruction

A NIB indicates which terminal is to be connected or how to communicate between a terminal and the program during execution of an OPNDST macro instruction. A NIB is similar to an RPL; both contain information about I/O requests. However, a NIB contains information necessary to communicate with a terminal.

An RPL also contains information about transactions such as a data address to display at a terminal, or whether to process a request synchronously or asynchronously.

A NIB can be partially modified by a MODCB macro instruction. Subsequently, this modification can be changed by a CHANGE macro instruction. A NIB can also be created dynamically by a GENCB macro instruction during program execution.

LOGON from a network console

A **network console** is a console defined to VTAM during system generation or by a VARY command. It can also be a console permitted to send VTAM commands, defined to VTAM by a command group.

By entering VARY NET, LOGON, terminal name, and the name of an application program from a network console, LOGON for a specified application program is requested. This request is processed if a LOGON exit routine exists. The terminal console becomes connected to the program when an OPNDST macro instruction specifying "ACCEPT" is issued in the LOGON exit routine.

If the specified terminal is already connected with a different application program, connection is impossible; this situation is signalled to the application program attempting the new connection.

If an application program being connected issues a CLSDST macro instruction, a LOGON request is created for this program and the LOGON exit routine is entered. Fig. 6.13 shows a LOGON from a network console.

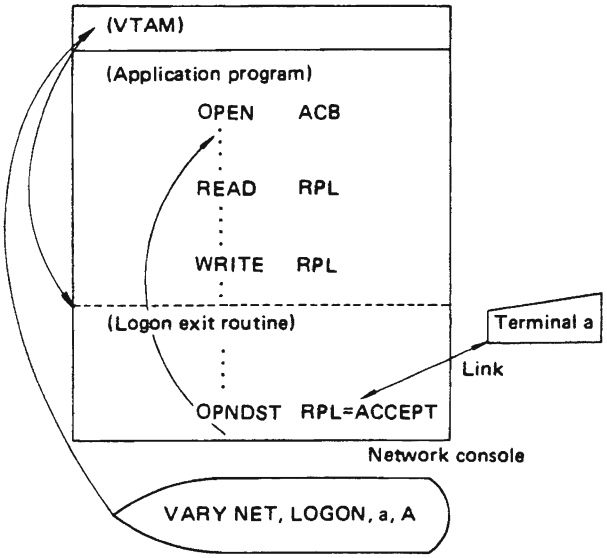


Fig. 6.13 LOGON from a network console

LOGON by an application program

A LOGON can be requested by an application program to connect with a specified terminal. The application program simulates a LOGON request. In this case, the application program processes its own LOGON request with a SIMLOGON macro instruction.

The above sequence is the same as an OPNDST macro instruction linkage request with the "ACQUIRE" option, except that the LOGON exit routine processes the connection request. Fig. 6.14 shows how an application program simulates a LOGON.

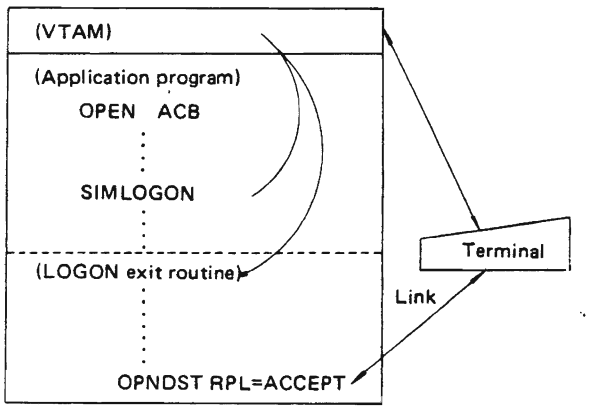


Fig. 6.14 LOGON from application program

Unilateral acquisition by an application program

An application program can request connection unilaterally between a terminal and itself. Such a request is satisfied immediately if the terminal is **available** (operable but no LOGON request has been issued). However, if the terminal has already

been connected with another application program, the terminal cannot be connected with the requesting program until released by its current owner. This type of request is indicated by the ACQUIRE option of the RPL named in the OPNDST macro instruction. ACQUIRE should have been defined in the associated network-definition APPL macro instruction. Fig. 6.15 illustrates unilateral acquisition by an application program.

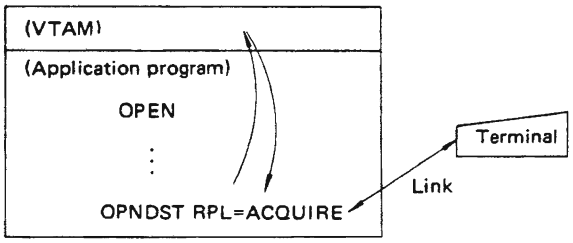


Fig. 6.15 Unilateral acquisition by an application program

Fig. 6.16 illustrates how different control blocks relate to one another when connecting a terminal to an application program via one of the four above methods. When a connection to the terminal has been established, data transmission can commence; NIB data can be moved to the VTAM internal table. The NIB of the application program is thereafter no longer used.

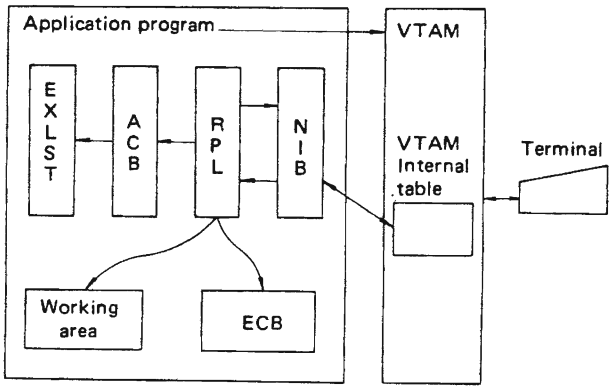


Fig. 6.16 VTAM control table specified by application program

6.4.4 Data Block Transmission Between an Application Program and a Terminal

Data transmission can start when an application progam is successfully connected to a terminal.

Receiving data from a terminal

The application program receives a data block by issuing a READ macro instruction. Data can be preread into a VTAM buffer by a SOLICIT macro instruction preceding the READ macro instruction. Each READ macro instruction specifies an RPL address, as described below.

A SOLICIT macro instruction acquires data from one or more linked terminals and stores these data into VTAM buffers. Data are thereafter transferred to the application program by subsequent READ macro instructions.

If VTAM receives a SOLICIT request, it immediately returns control to the requestor.

There are two types of READ macro instructions: specifying/not specifying physical I/O operations. If ANY is specified in the RPL, no I/O operation is performed; any data already successfully solicited from the terminal are transferred to the application program.

If a READ macro instruction was issued to a special terminal—RPL=SPEC—and data were not yet solicited from this terminal, READ first solicits data.

VTAM transfers data to an application-program buffer and sets a bytes-transferred count into the RPL at the same

A request for a READ operation is also posted in the corresponding RPL. Several options and parameters for the READ macro instruction are determined by this RPL, for example from which terminal (of several solicited terminals) data have been acquired or location in the application program to receive data.

VTAM provides a RESET macro instruction to cancel I/O macro instructions such as READ and WRITE. The RPL specifies whether linkage requests between an application program and a terminal are synchronous (SYN) or asynchronous (ASY).

Synchronous requests

Control is not returned to the application program until the requested operation completes, normally or abnormally. A CHECK macro instruction should not be used for a synchronous request; VTAM automatically performs the CHECK function.

Asynchronous requests

Control is immediately returned to the application program after VTAM schedules the requested operation. When this operation completes, VTAM performs one of the following actions:

- If an ECB operand of the RPL was specified, this ECB is posted. The application program should issue a CHECK macro instruction to test whether the ECB has been posted "complete."
- VTAM schedules the indicated exit routine if the EXIT option of the RPL was specified. A CHECK macro instruction can be issued in this exit routine. In this case, LERAD or SYNAD exit routine is automatically called if the requested operation completes with a logical error or physical error.

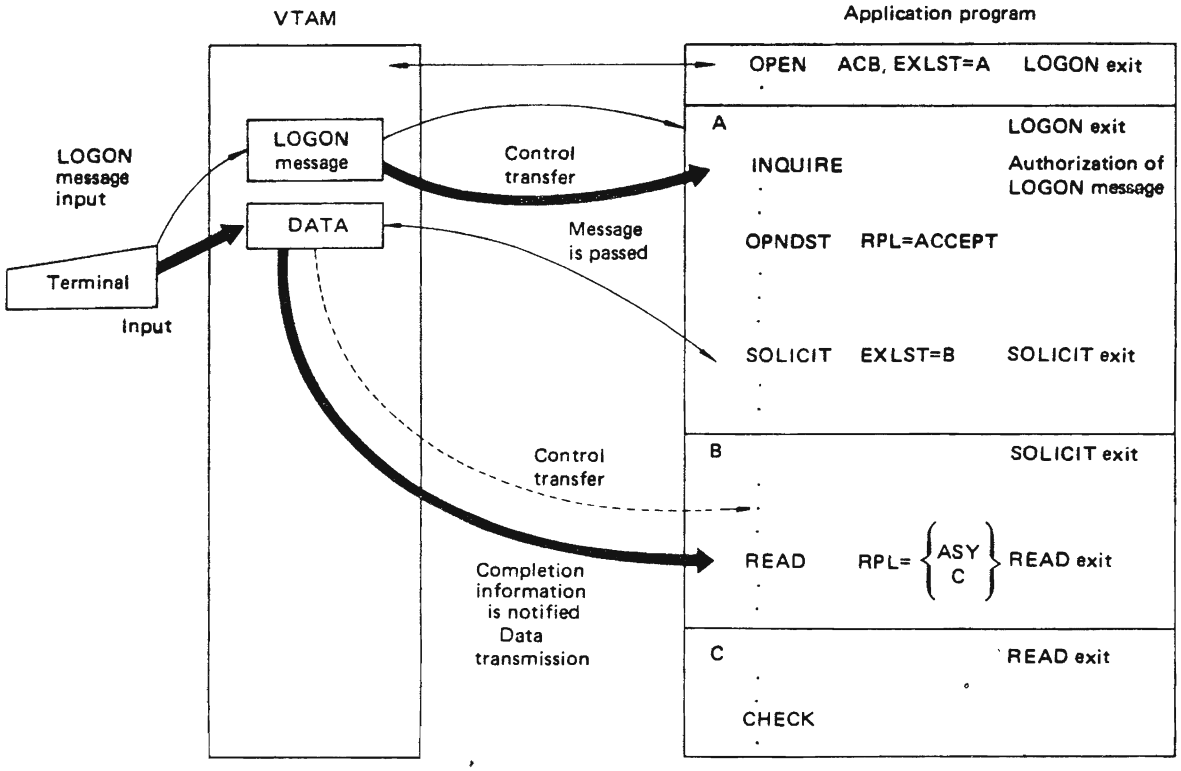


Fig. 6.17 Receiving data

Fig. 6.17 illustrates how data are received by issuing SOLICIT and READ macro instructions.

Transmitting data to a terminal

Data are transmitted by WRITE macro instructions. Each WRITE transfers one block of data from an applications program area to a particular terminal. Highlights of VTAM WRITE include the following features:

- Automatic read after write (conversational WRITE).
- Write after refreshing the screen on a display unit.
- Refresh the nonprotected portion of a screen without transmitting data.

If a WRITE macro instruction is issued to a terminal whose preceding WRITE operation is in process, the second WRITE is executed after the first WRITE operation completes.

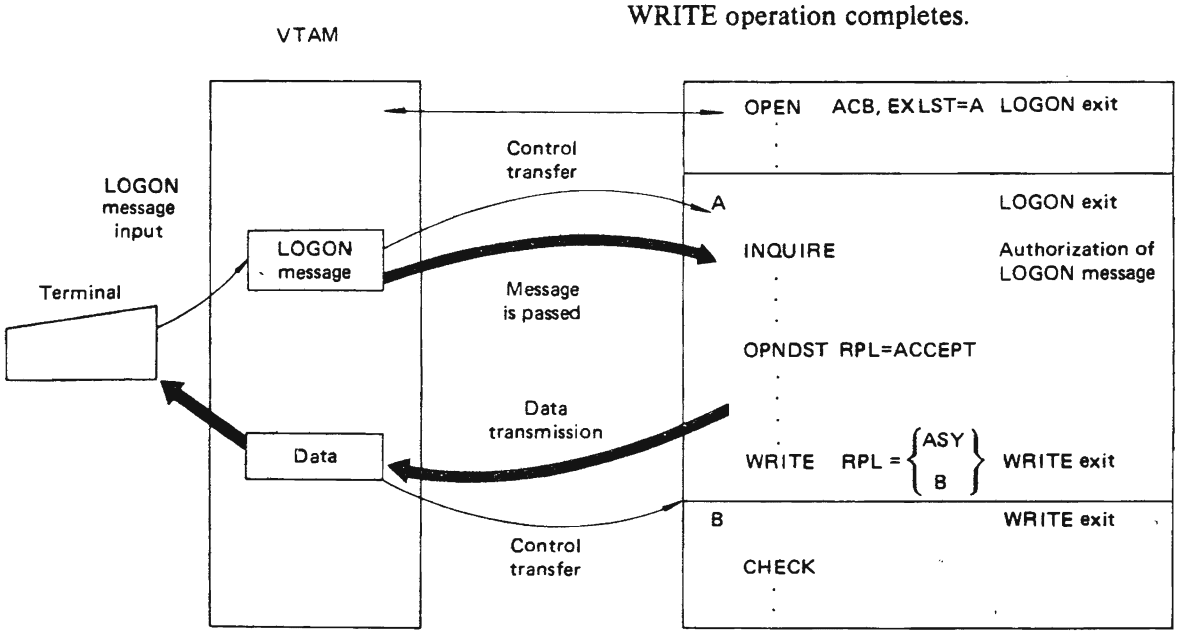


Fig. 6.18 Transmission of data

If a WRITE macro instruction is issued to a terminal to which a SOLICIT is already directed, the WRITE macro instruction is usually executed after the solicitation has been completed.

If a WRITE macro instruction is issued to a terminal subsequent to a SOLICIT/READ sequence of macro instructions specifying CONT (continuous) in its RPL, data are transmitted to the terminal after the latter surrenders control via an EOT signal. In this case—end of transmission (EOT)—the SOLICIT/READ sequence recommences.

When an RPL is specified in a WRITE macro instruction, the RPL furnishes the terminal address, virtual-storage address of output data, operation type, etc. A RESET macro instruction may be used to reset a previous WRITE macro instruction.

Synchronous/asynchronous operation of a WRITE operation (specified by SYN/ASY in the RPL) is interpreted just as for a READ macro instruction.

Fig. 6.18 illustrates how data are transmitted using a WRITE macro instruction.

VTAM furnishes a special macro instruction to transfer blocks with heading characters†: the DO (device order) macro instruction, whose control table is created by an LDO (logical device order) macro instruction. A DO macro instruction names the RPL that specifies the LDO. Operation of a DO macro instruction is the same as that of a WRITE macro instruction.

Dynamic network control by operator commands

VTAM permits a central-site operator to redefine, modify, and monitor a network dynamically during usage. These facilities are as follows:

- VARY command: dynamically redefine the network.
- MODIFY command: modify the network status.
- DISPLAY command: monitor the network.

Dynamically redefine the network

The status of a terminal, line, line list, etc. can be moddified by a VARY command as follows:

- active/deactivate a remote or local terminal
- active/deactivate a communications line.
- active/deactivate a line list (specified by LINELIST macro instruction in the network definition).
- Activate/deactivate an NCP line.

Modify the network status

An operator can modify one or more operational characteristics of a CCP by MODIFY command.

Monitoring the network

The status of an application program, terminal, line, or CCP can be displayed by a DISPLAY command: what resources are used, by whom, error information, etc.

6.4.5 Releasing a Linkage Between an Application Program and a Terminal

The connection between an application program and a terminal should be released when date transmission completes or — infrequently — when an emergency request arrives. The released terminal can be connected with another application program if desirable.

There are two ways to release a linkage:

- LOGOFF from a network console.
- Release by the application program.

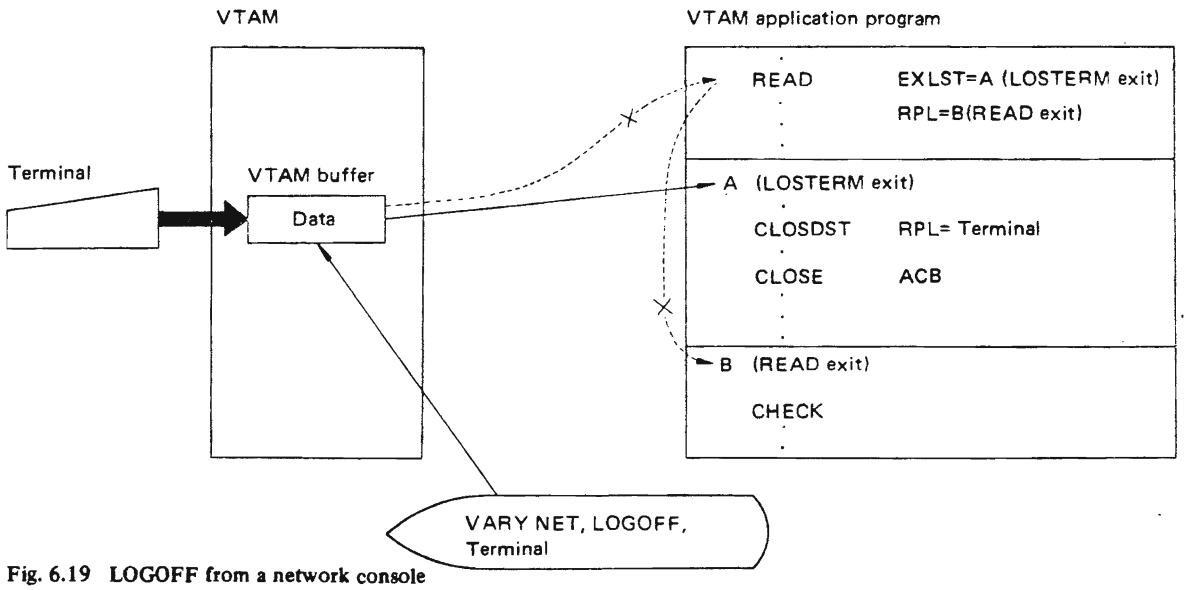
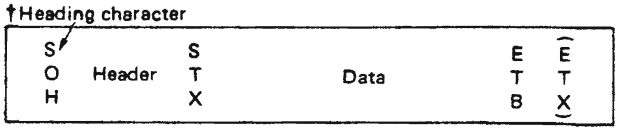


Fig. 6.19 LOGOFF from a network console

†Heading character



LOGOFF from a network console

The connection between the specified terminal and the application program is released by entering from network console:

VARY NET, LOGOFF, TERMINAL-name

When this command is issued, I/O operations stop between the specified terminal and application program. If furnished, a LOSTERM exit routine commences, as specified by an EXLST macro instruction.

Any in-process I/O requests end normally, but new requests are cancelled. Cancellations are posted in the corresponding RPLs.

The applications program should release the terminal entirely by subsequently issuing CLSDST macro instruction, as shown in Fig. 6.19.

In Fig. 6.19, data already in a VTAM buffer are passed to the application program when READ macro instructions are issued. Also, any appropriate READ exit routines are entered. However, if an operator issues a VARY or LOGOFF command during input from a terminal, the LOGTERM exit routine is scheduled.

Release by the application program

An application program can release its connection to a terminal by issuing a CLSDST (close destination) macro instruction. Each such terminal can be specified in the RPL named in the CLSDST macro instruction; any data in a VTAM buffer are discarded and are not retained for any application program subsequently connecting to the terminal.

A terminal can be connected to another application program by a CLSDST macro instruction specifying PASS in the corresponding RPL. VTAM first releases the terminal, then creates a LOGON request from the terminal. In this case, the application program issuing the CLSDST macro instruction should indicate which application program should receive the LOGON request.

A RELEASE option in an RPL merely releases the terminal. If another application program has requested access to this terminal, VTAM connects it to the terminal if appropriate. A CLOSE macro instruction – rather than a CLSDST macro instruction – should be issued if the application program has completed all processing and all terminals are to be released.

A terminal cannot be connected to another application program during/after a CLOSE macro instruction. In Fig. 6.20, terminals AA and BBB are exchanging data with an application program. The application program releases each terminal when the exchanges are completed. It issues a CLOSE macro instruction to release the VTAM linkage when all exchanges with all terminals are completed.

6.4.6 Termination of a VTAM Application Program

Like other programs, a VTAM application program usually terminates by issuing a RETURN macro instruction. Alternatively, it may be cancelled (CANCEL command) by an operator. In either case, it completes only after releasing its connection to VTAM. A RETURN macro instruction should be issued only after releasing connections to all terminals. The OS IV/F4 terminator releases any remaining connections with terminals in case the operator issues a CANCEL command.

The linkage of an application program to VTAM is released by a CLOSE macro instruction. When VTAM receives a CLOSE macro instruction, it releases the linkage between the corresponding ACB and the terminal. All I/O operations are quiesced, and all unprocessed I/O requests and queued LOGON requests are cancelled prior to releasing the linkage. Fig. 6.21 shows how a VTAM application program terminates normally.

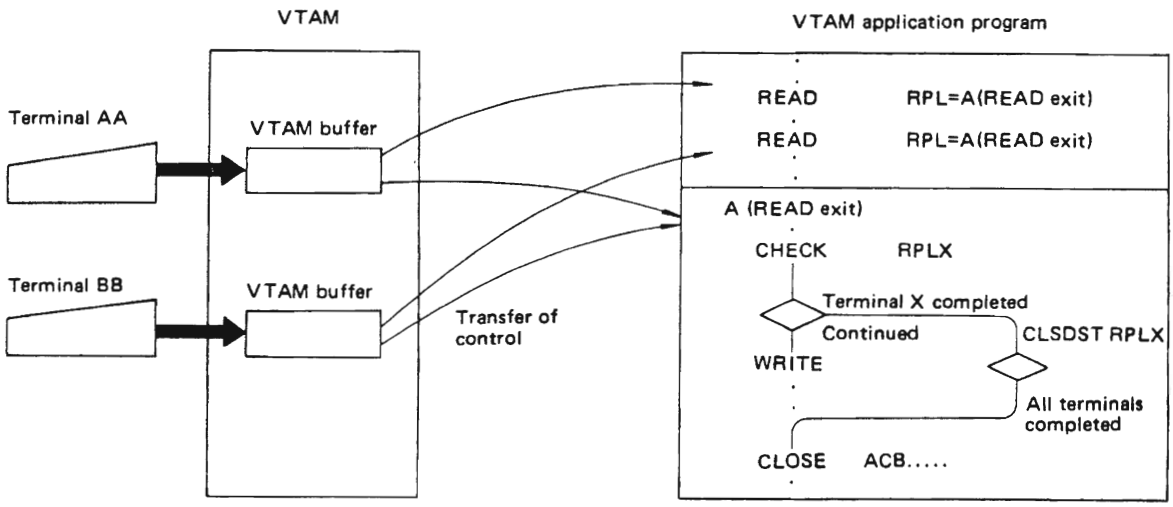


Fig. 6.20 Release by the application program

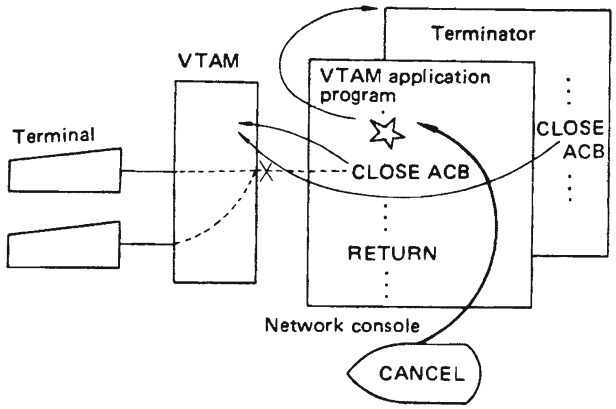


Fig. 6.21 Terminating a VTAM application program

6.4.7 End of VTAM Operations

The HALT command stops VTAM and deallocates the address space acquired by the START VTAM command.

There are two types of HALT command modes to complete VTAM. Either can be selected by the operator, depending on the purpose of the shutdown and current system status:

- FLUSH mode (gradual completion).
- QUICK mode (immediately).

Halting VTAM is trivial if all VTAM users have already been logged off.

The main functions of the HALT command are as follows:

- HALT inhibits any subsequent OPEN macro instructions issued to VTAM.
- HALT initiates any TPEND exit routine which may have been previously specified by a EXLST macro instruction.
- In case of QUICK mode, all terminals are deactivated: subsequent I/O requests are inhibited, any I/O requests already queued by VTAM or NCP are cancelled.
- In case of FLUSH mode, I/O requests and connections are allowed.
- If any VTAM application programs are still active, their names are displayed on console.

In both QUICK and FLUSH modes, all VTAM application programs should have previously issued CLOSE macro instructions whose functions are now completed, so that VTAM can complete in an orderly manner.

6.5 RAS FACILITIES FOR DATA COMMUNICATIONS

Reliability, availability and serviceability (RAS) facilities for OS IV/F4 data communications are offered jointly by VTAM and NCP.

Data-communications RAS is divided into three functional components:

- Diagnostic facilities
  Collects trace information about each event. The status of the entire network can be comprehended by summarized trace information. These summaries also help installation managers increase network performance.
- Recovery facilities
  Recovery is attempted a prespecified number of times when an error occurs. A restart facility is also provided for the CCP.
- Error-recording facilities
  Temporary and permanent errors for each hardware component are recorded in detail and/or summary form on DASDs and/or hardcopy consoles.

6.5.1 Diagnostic Facilities

Diagnostic facilities fall into six categories as follows:

- trace facility.
- NCP dump facility.
- NCP time monitoring.
- dynamic panel display of the CCP.
- error message displays.
- online terminal tests.

Trace facility

The trace facility for OS IV/F4 data communications collects the following data:

- SVC events – collected each time SVC is issued.
- VTAM I/O operations – collected for each I/O operation of VTAM or NCP.
- NCP lines – collected for specific communication lines.
- Addresses – collected register contents and specified areas in the CCP.

VTAM I/O traces and NCP line traces are started and ended by MODIFY commands.

Trace information is written into the SYS1.TRACE data set, as shown in Fig. 6.22.

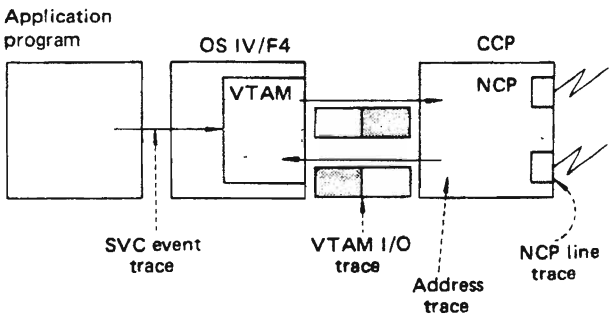


Fig. 6.22 Types of tracing

NCP dump facility
VTAM writes error messages if the NCP malfunctions or if errors are found in I/O requests from VTAM to the CCP.

VTAM does not continue I/O operations for the corresponding CCP after such errors. NCP dumps and NCP diagnostics can be requested by MODIFY commands.

NCP time-monitoring facility
This facility checks whether the CCP is operating normally, using one or more predesignated terminals.

NCP checks whether all of its processes operate normally each time the operator enters this command and then returns its acknowledgement to VTAM.

The CCP time monitor continues until a HALT command is issued from a central-site console. If an error is detected, the NCP time monitoring facility notifies the application program and displays the error on a console.

CCP dynamic panel display facility
The operator can display the following CCP information by pushing buttons on the control panel:
- contents of the CCP external registers.
- contents of main storage.
- information about a communication line.

Error-message display facility
VTAM writes necessary network information on a network console when an error is found, so that it is immediately available to the central-site operator.

Online terminal test facility
The VTAM/NCP combination furnishes an online terminal test facility which may be requested by either the terminal operator or a central-site operator, using the telecommunication online test control program (TOLTEC) in VTAM. The result of each test is passed to VTAM by NCP.

6.5.2 Recovery Facilities

Recovery facilities fall into eight categories as follows:
- line error recovery.
- local unit error recovery.
- automatic network shutdown.
- checkpoint/restart.
- switching to a back-up line.
- switching channel adapters.
- restarting the NCP.
- pause retry.

Line error recovery
NCP attempts to recover from temporary and permanent line errors. When NCP completes a recovery attempt, it transmits an error record to VTAM as a permanent record of the error. VTAM writes this error record into the SYS1.LOGREC data set.

Local-unit error recovery
I/O errors are classified as permanent or temporary. When a permanent error occurs, its status is sent to the network console which records such errors. When a temporary error occurs, the associated block is retransmitted to attempt recovery from the error. If successful, the application program continues to execute as if no error has occurred. If a temporary error can not be recovered, it is tabulated and treated as a permanent error.

Automatic network shutdown
A VTAM/NCP network shuts down automatically if the channel between VTAM and NCP malfunctions or NCP is unable to communicate with VTAM. When a network begins an automatic shutdown, NCP sends an emergency message to all connected terminals; all subsequent I/O operations are inhibited. NCP can be restarted by a restart request from VTAM or reloading the NCP (**NCP IPL**).

Checkpoint/restart
This facility permits resumption of an interrupted NCP from a point somewhat prior to the point of interruption by using the latest checkpoint record.
- Creating a checkpoint record
  A checkpoint record is created when the status of a significant building block of the network changes. NCP status at this time is recorded in this record. If VTAM requests creation of a checkpoint record, NCP creates and transfers it to VTAM.
- Restarting
  If an error requiring reloading of the NCP occurs in the CCP, VTAM reloads NCP and initializes it as requested by the console operator.

  After initialization, VTAM requests a restart effort by sending the most recent checkpoint record to the NCP, in order to restore the NCP to its status prior to the error. NCP uses this record to restore its own status and restart network control.

Switching to a backup line
If permanent error occurs on a **private line** (not on the switched public telephone network), or if a temporary error occurs frequently on a private line, it is sometimes possible to use a switched line as a back-up line, to which it can be manually connected by a remote operator (in the case of a terminal) or a central-site operator (in the case of a central-site line problem).

Exchanging channel adapters
Two channel adapters in a CCP can be exchanged by a request from VTAM. If one adapter malfunctions, it can be switched out of service and a correctly-

functioning adapter switched into service.

Restarting NCP
VTAM can restart an NCP after its CCP has malfunctioned. Restarting an NCP supported by VTAM requires initial program loading (IPL) of its CCP, followed by a VTAM-issued restart. In either case, a MODIFY command is used to restart the NCP.

Pause retry
NCP provides two types of recovery levels for line errors. The first level retries continuously, as often as specified during system generation of the NCP. The second level retries after pausing for a specific interval (**pause retry** if the error was not bypassed by the first-level procedure. Pause retry is attempted only for WRITE operations, not for READ operations. If an error cannot be recovered after several pause retry cycles (number specified at NCP generation), VTAM records this error.

6.5.3 Error recording facilities

VTAM writes various records into the system log data set, where they may be summarized and reported to maintenance personnel periodically. The following types of error records are captured on LOGREC:

- Outboard record (long)
  Failure of a local I/O device.
- Outboard record (short)
  A similar but shorter record.
- Inboard record
  Failure of a channel.
- MDR (Miscellaneous Data Record)
  Created by the NCP, unexecutable command error or a line error in the NCP.

MDR records in turn are of two types, as follows:

- MDR record for adapter check, program check, or invalid interruption.
- MDR record for statistical summaries about a line or a permanent error on the line.

6.6 NCP

The network control program (NCP) is a special program stored in a communication control processor (CCP), which supports communications networks in cooperation with VTAM.

NCP is linked with VTAM via a host-computer channel. Only one subchannel is necessary for this linkage, which is strikingly different from previous architectures which required dozens — even hundreds — of subchannels between the host computer and the communication controller.

Communication between VTAM and NCP is performed in **basic transmission units** (BTUs). Each BTU comprises control information, a selection field, and message data. A BTU transmitted from VTAM to NCP is called a **command BTU**, a BTU transmitted from NCP to VTAM is called a **response BTU**. NCP is almost always ready to receive information from VTAM, without regard to the status of particular line or terminal. During a NCP-to-VTAM transmission, one operation can contain multiple response BTUs. NCP reads them from VTAM into its own buffers in the CCP. An arbitrary number of BTUs can be transmitted to NCP from VTAM at any time.

During NCP-to-VTAM transmission, NCP notifies VTAM of the arrival of new information. However, information transfer cannot begin until VTAM issues an appropriate input command.

VTAM can send multiple command BTUs to the NCP in one input operation; the maximum number of BTUs is determined during NCP generation. During network control, the NCP receives data from VTAM and remote terminals concurrently. VTAM indicates channel speeds for a remote network, while NCP and CCP perform line control for remote terminals with respect to line speed.

Major facilities of NCP are as follows:

- line control.
- dynamic buffer control.
- insertion and deletion of transmission control characters.
- translation of character codes.
- handling of permanent errors, and detection of machine checks.
- collection of line-status data.
- activation/deactivation of lines.
- closing the network.
- processing recoverable errors.
- timestamping messages and control fields.
- notifying VTAM of any errors.

6.6.1 Basic Transmission Units

All communications between host computer and CCP are in basic transmission units (BTUs); data messages, CCP status information, host-computer status information, control messages for the network, responses from the NCP, etc. Also, the NCP sends **unsolicited elements** to VTAM sporadically, describing abnormal status conditions in the CCP as they occur. VTAM does not solicit such elements, since it cannot anticipate when abnormal status conditions will arise.

Fig. 6.23 shows the three categories of BTUs.

BTUs are used for four types of commands:

- Teleprocessing commands
  TP commands control (a) the start and end of each session between VTAM and an I/O device and (b) data transmission. There are five TP commands: INVITE, CONTACT, READ, WRITE and DISCONNECT.

- TEST commands
  These ttelecommunications online test commands (TOLTEC) request start, execution, and end of online terminal tests.
- RESTART commands
  Using checkpoint records, these commands request restart of the NCP.
- CONTROL commands
  These commands modify or monitor network resources dynamically.

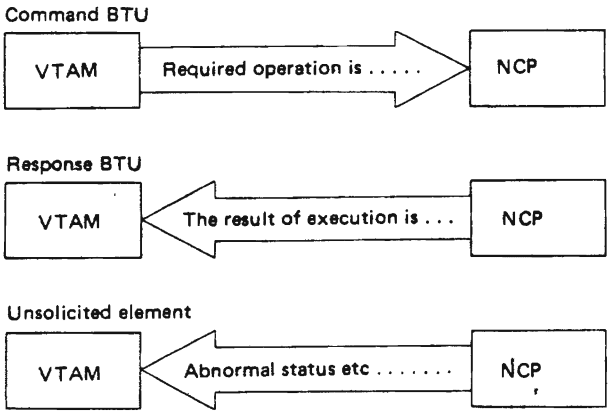


Fig. 6.23 BTU categories

6.6.2 Buffer Management

Both VTAM and NCP use internal I/O buffers to exchange BTUs. The effective number of buffers in a CCP is automatically calculated by NCP after it has been loaded and started execution. NCP divides remaining (unallocated) CCP main storage into buffers whose lengths were determined during system generation.

Each BTU requires one or more buffers. Sizes of buffers within NCP and VTAM may vary this specification is furnished in the VTAM/NCP definition macro instruction. When VTAM and NCP are exchanging BTUs, they chain one or more buffers together. Hence, even if the buffers are too long/short for a particular block, VTAM and NCP can accurately transfer the correct amount of data.

VTAM to NCP data transfer
The NCP recognizes the command-chaining status, the command linking status, and the input format from host; it fills its own buffers as it receives data from VTAM.

NCP to VTAM data transfer
NCP creates response BTUs after receiving data and adding appropriate control information. NCP transmits buffers to VTAM when it receives ending characters such as EOT from terminals or internal NCP buffers become filled.

VTAM chains its buffers together while receiving data from the NCP. Therefore, each received BTU is stored in one or more VTAM buffers.

Buffer insufficiency
When insufficient buffers remain in the CCP, one of two remedial modes is entered:

- Slowdown mode
  NCP enters slowdown mode if it has insufficient buffers for receiving data from VTAM or from a line. NCP notifies VTAM when it must slowdown. Thereafter, VTAM sends NCP single-buffer BTUs at a rate controlled by NCP, to avoid losing data. When NCP has emptied enough buffers, it leaves the slowdown mode.and notifies the host that it is ready to exchange BTUs normally.
- Nobuffer mode
  NCP retains one empty buffer to issue CONTROL commands when it enters slowdown mode. In this case, BTU cannot be received. If this buffer is in use, the NCP can no longer receive BTUs from any source – VTAM or lines.

6.6.3 Starting a Network Control Program

VTAM loads the NCP into the CCP. When loading has been completed and NCP assumes control of the CCP, it initializes the following hardware/software elements of the network:

- channel adapters, communication scanners, and virtual storage.
- NCP control tables.
- buffer pools.

NCP notifies VTAM when initialization has been completed, VTAM then starts NCP network control by a TP or CONTROL command. VTAM restarts network control after an interruption by a RESTART command.

6.6.4 Ending Networkk Control Activities

After VTAM issues a CONTROL command requesting shutdown of the network, NCP performs appropriate termination processing. In the following cases, NCP itself initiates termination:

- If communication between NCP and VTAM becomes impossible due to a channel-adapter failure or channel time-out.
- If the central-site operation initiates network termination from the CCP panel.

6.6.5 Data Units

BTU commands transmit three kinds of data units from VTAM to NCP: **blocks, messages,** and **transmissions,** specified by device type during NCP

generation and modifiable thereafter by control commands. The type of each block is specified by the corresponding WRITE command.

In Fig. 6.24, downward-pointing arrows indicate when NCP transmits BTUs to VTAM. Thereafter, NCP must decide whether additional data are expected from the originating terminal or whether NCP should send it an acknowledgement; this decision depends on which data units are present. The vertical symbols in Fig. 6.24 have the following definitions:

(ETB) End of Text Block

(BET) End of Text

(XOT) End of Transmission

6.6.6 Session Service

Each network controlled by VTAM and NCP has a physical linkage and a logical linkage. The **physical linkage** denotes the hardware network comprising the CCP, terminals, and communications links.

The **logical linkage** is the network of NCP and other resources which are linked at a given instant for transmitting data.

A **session** is defined as a flow of commands and data between VTAM and certain network resources.

For a terminal without multiple concurrent I/O capability, two types of connections are possible: **single-drop** (also called **point-to-point**) and **multi-drop,** where several such terminals connect to a single private (non-switched) line. For a single-drop terminal, only one session can take place. For a cluster-type terminal on a multi-drop line, the single line can be used for multiple network resources, which increases line utilization and avoids its pre-emption for a single function.

Figs. 6.25 and 6.26 illustrate sessions for single-drop and multidrop lines, respectively. A **contact command** transmits the terminal address at the beginning of a session, an **invite command** polls the terminal address. Polling is a communications activity performed by a CCP to determine whether a

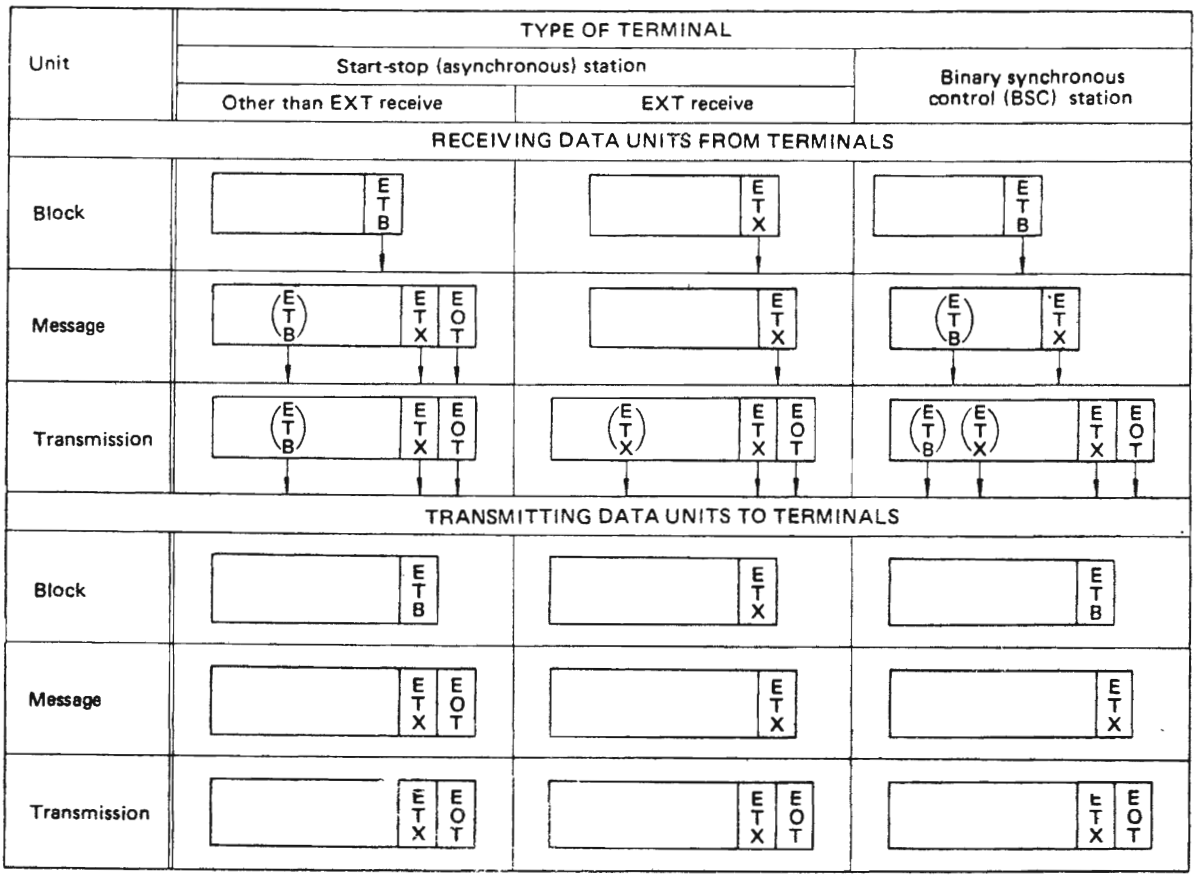


Fig. 6.24 Data units

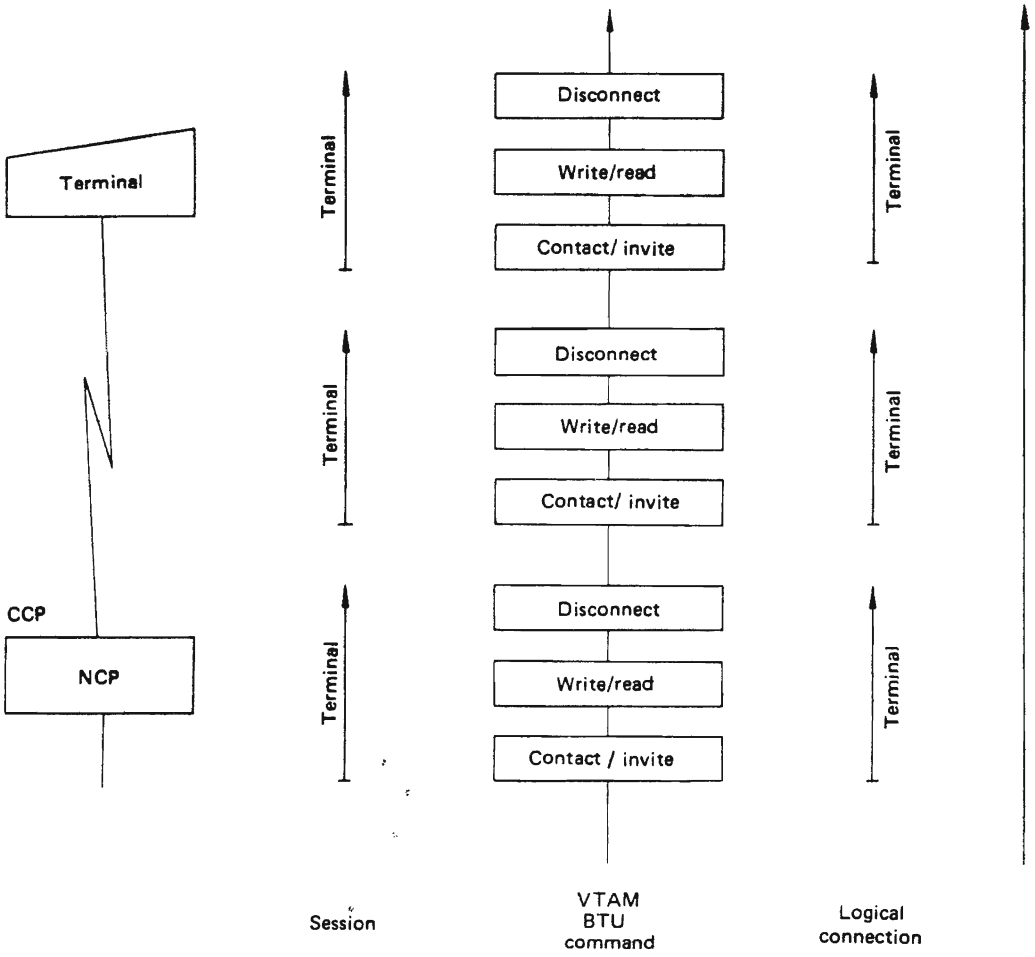


Fig. 6.25 Session with a single-drop private-line terminal

particular terminal is ready to send/receive data.

Macro instructions defining sessions and logical connnections are included in the NCP definition generation deck, as defined in Section 6.3.2:

- GROUP
- LINE
- SERVICE
- CLUSTER
- TERMINAL
- COMP

Sessions and logical connections can be defined by suitably combining these macro instructions. Various session attributes are defined below:

- Session limit
  The **session limit** is the maximum number of concurrent sessions possible on a multidrop line (or for a local cluster-type terminal).
- Service order table
  Service seeking comprises checking resources with which to construct a network when NCP attempts to establish a new session on a multidrop line by polling (INVITE command) or addressing (CONTACT command).
  The **service order table** defines the sequence in which the VTAM application program requests service seeking; it lists network resources on a multidrop line by cluster, terminal, and component.
- Service seeking limit
  The **service seeking limit** is the number of entries in a service order table. NCP generally attempts to establish sessions up to this limit. Service seeking ends as soon as a positive response can be satisfied by a network resource, so that a new session is established.
- Service seeking interval
  The **service seeking interval** is how long NCP will attempt to start a new session. By timing service seeking and stopping after this interval, NCP decreases unnecessary polling and can perform other useful tasks.
- Priority of service seeking
  This parameter determines whether NCP should give higher priority to serving the current session or to establishing a new session.

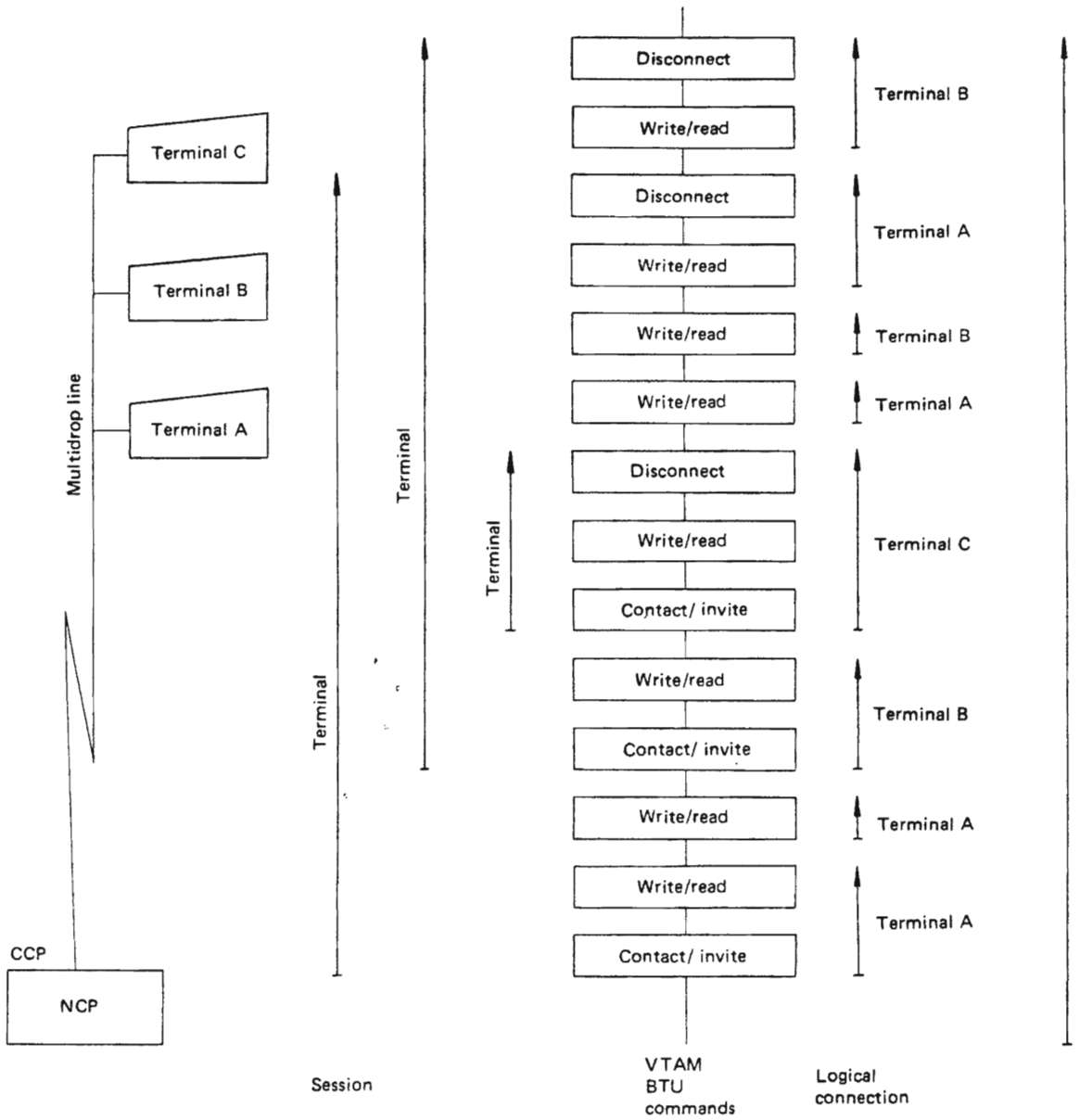


Fig. 6.26 Session with multidrop terminals

- Transmission limit
  A **transmission limit** is a maximum value for the number of transmissions during one logical linkage. If the number of transmissions reaches this limit, the logical linkage is released at the end of the current operation.
- Negative polling limit
  The **negative polling limit** is the number of consecutive negative responses before NCP releases a logical linkage. Therefore, this limit applies to polling which attempts to restart a previously-established session.
- End of polling routine
  This parameter selects what action NCP should take when it reaches the negative polling limit.

6.6.7 Block Handling Facilities

NCP has three optional block **handling facilities:**

- insertion of date and time **(timestamp)** into messages.
- automatic text correction.
- user-written block handling routines.

Timestamp

NCP will optionally insert date and time of its arrival into a message, whether the latter arrived from the host computer via a channel or from a terminal via a line. This timestamp can be placed only in the first block of a message or in each block comprising the message.

Automatic text correction
NCP will automatically delete text-correction characters and error characters from messages entered by keyboard terminals. Any character except a line-control character can be defined as a text correction character.

User-written block handling routines
These can be created and entered as necessary.

CHAPTER 7
RELIABILITY, AVAILABILITY, AND SERVICEABILITY

7.1 OUTLINE OF RAS

As computer systems become larger and more diversified, they require ever-improved reliability, availability and serviceability. Operating systems for these computers provides many aspects of RAS as follows:

- Record problems and probable causes at early stages, whose data may aid prevention of these problems.
- Isolate hardware and software components experiencing problems.
- Recover from these problems automatically as much as possible.

7.2 RECOVERY MANAGEMENT SUPPORT

Recovery management support (RMS) attempts retry of a failing CPU or I/O operation in order to use corresponding hardware as long as feasible. Reliability of the total system is improved by gathering and recording error information; recorded data may help protect the system from trouble and maintenance episodes. RMS includes the following functions:

Machine check handler (MCH)
MCH records failures of the CPU and main memory. It also attempts to recover from these failures, in conjunction with the Alternate CPU Recovery facility.

Alternate CPU recovery (ACR)
When one CPU of a tightly-coupled multiprocessor configuration (two CPUs sharing main storage as well as various peripherals) malfunctions, the surviving CPU uses the ACR facility to (a) diagnose the malfunctioning CPU, (b) attempt to resume its operation, or (c) if the malfunction is serious, continue full processing functions with the surviving CPU.

Channel check handler (CCH)
This facility analyzes hardware failures in a channel, such as channel interface checks and channel control checks.

Alternate path retry (APR)
If a hardware failure occurs on one path of an I/O device having several paths to main memory, the APR facility will attempt the I/O operation on one or more alternate paths.

Missing interruption handler (MIH)
If an I/O interruption is "lost" by a channel, control unit, device, or software routine, the associated task may become delayed indefinitely. The MIH facility timestamps all I/O operations; if certain operations (such as read, write, and control commands to tape and disk drives) do not complete within reasonable intervals, MIH advises the console operator and asks him to issue intervening commands.

Dynamic device reconfiguration (DDR)
When the OS IV/F4 I/O supervisor detects a permanent error on a mountable volume such as a disk pack or tape reel, it will invoke the DDR facility, which asks the console operator to move this volume to another drive where the failing I/O operation can be reattempted.

Error recovery procedures (ERPs)
Whenever a nontrivial hardware failure occurs on an I/O device, the device-dependent ERP analyzes the failure as much as possible and — if feasible — attempts to recover from it.

LOGREC recording
All nontrivial hardware failures are logged into the SYS1.LOGREC data set by the OS IV/F4 supervisor.

Fig. 7.1 illustrates the relationship among these RMS components.

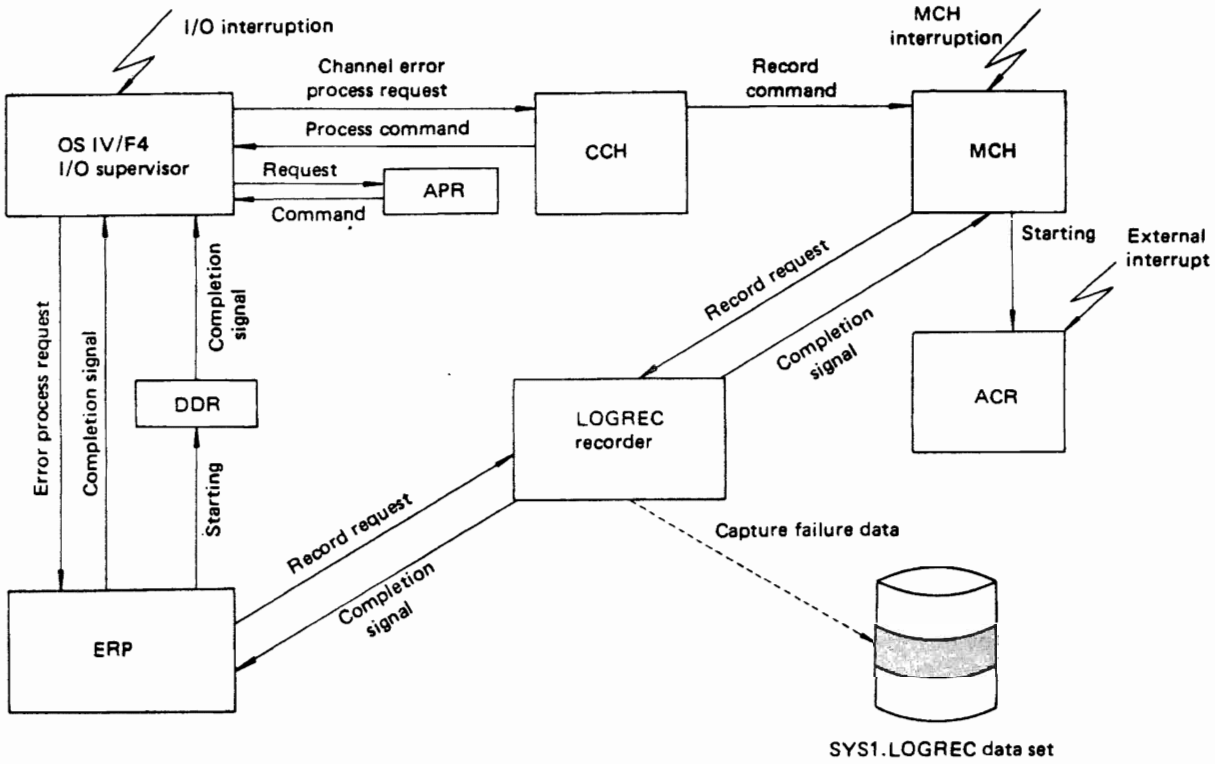


Fig. 7.1 Outline of RMS

7.2.1 Machine Check Handler (MCH)

The machine check handler (MCH) gathers information about all machine checks and records them on the SYS1.LOGREC data set. It determines if recovery from a malfunction can be successfully completed by M series hardware facilities. If recovery attempts are unsuccessful, MCH performs a limited analysis, then invokes appropriate software routines.

In a multiprocessing environment, if MCH is unsuccessful in a failing CPU, it will attempt to initiate further recovery processing by marking the failing CPU offline and invoking alternate CPU recovery (ACR) in the nonfailing CPU.

If the malfunction occurs in a CPU or main memory, recovery is attempted by the **hardware instruction retry** (HIR) and/or **error checking and correction** (ECC) hardware functions. If HIR and ECC fail to recover from the malfunction, a machine check interruption is generated to start MCH.

MCH starts analysis of a main-memory malfunction and provides – according to the result of analysis – the following attempts to continue system operation:

- If a copy of the damaged data or program exists on an external page data set, MCH attempts to page in a fresh copy, so that system operation may be continued without any loss of program/data integrity.
- If the effort to restore destroyed data or program fails, MCH terminates the task which last referenced the malfunctioning page. This does not affect other currently-executing jobs.
- If the error is a permanent one, an invalid flag is turned on in the malfunctioning page frame, so that the OS IV/F4 Supervisor will refrain from subsequent usage of this frame.
- If none of these steps succeed in recovering system operation, MCH puts this CPU into disabled wait state, since it can no longer be used without emergency maintenance.

7.2.2 Alternate CPU Recovery (ACR)

Alternate CPU recovery (ACR) is a new feature in OS IV/F4. When one CPU in a tightly-coupled multiprocessing configuration can no longer function, a signal is emitted before the CPU enters a permanent wait or stopped state. The signal can indicate a hardware malfunction alert or a software emergency signal; in either case, ACR is invoked.

When ACR receives control, it attempts to transfer tasks in progress on the failing CPU to the nonfailing CPU. It cleans up tasks that cannot transferred by processing their failures as abnormal

terminations. ACR resets I/O operations on channels connected to the failing CPU. For symmetrically connected I/O, ACR attempts to restart it through channels to the working CPU. In the case of symmetrically-connected I/O (two-channel switch through channels on two CPUs), ACR invokes recovery termination management routines to recover the program in progress at the time of the failure.

7.2.3 Channel Check Handler (CCH)

If a channel data check (parity error), channel control check (channel device control error) or interface control check (channel-I/O interface control error) occurs, M series hardware automatically stores associated error information into a fixed log-out region in main memory. Then the OS IV/F4 I/O Supervisor transfers control to the CCH, so that this malfunction can be analyzed.

Based on log-out information, the CCH evaluates the error, generates a channel error alarm for the operator, and logs the error information onto the SYS1.LOGREC data set.

In the case of a channel control check or interface control check, the CCH prepares information needed by the corresponding ERP retrying the channel program.

7.2.4 Alternate Path Retry (APR)

The APR option can be generated into OS IV/F4 if the configuration includes one or more I/O devices with multiple channel paths to main memory.

If one channel path develops an error, the APR attempts to process the I/O request on an alternate channel path, which must previously have been assigned to the device performing the I/O operation. An operator can vary a path to a device online or offline, using the VARY PATH command. He can vary offline all paths except to a shared direct access divice with an outstanding RESERVE request or the last path to an allocated device.

7.2.5 Missing Interruption Handler (MIH)

The **missing interruption handler (MIH)** of OS IV/F4 notifies the console operator if a device or channel-end interruption is not received within a prespecified interval. If an interruption has not been received, it is possible that a MOUNT message has not been satisfied or that a device has malfunctioned. Specific actions by the operator depend on conditions he encounters. He may be required to ready a device on which a volume has been mounted, examine indicator lights on the device for abnormal signs or terminate the job. By supervising completion of I/O operations, the operator can protect OS IV/F4 from delays due to indefinite waiting by I/O devices.

7.2.6 Dynamic Device Reconfiguration (DDR)

When an I/O error is sensed, the appropriate ERP retries the channel program so that OS IV/F4 can recover from the failure. If a prespecified number of retry attempts fail, the ERP indicates a permanent malfunction of the I/O device at that point on its recording medium. If this medium is removable, DDR is invoked.

Dynamic device reconfiguration surveys the entire configuration for an allocable I/O device of this type and commands the operator to move the mountable volume to another device. In this procedure, DDR repositions the volume so that the channel program may be successfully executed on the replacement I/O device and hence the interrupted program can resume execution without abnormally terminating.

The operator can initiate DDR activity by issuing a SWAP command, for example, when he wishes to clean a tape drive.

DDR cannot be applied to a system volume, such as one containing a page data set.

7.2.7 Error Recovery Procedure (ERP)

Each error recovery procedure analyzes and recovers from certain types of hardware failures of a specific I/O device (or family of devices).

When a hardware failure is sensed while performing an I/O operation, the OS IV/F4 Supervisor calls the appropriate ERP, which analyzes the error information and — if feasible — retries the channel program. If the ERP fails to recover from the error, DDR may be started for a mountable volume; in all other cases, the task for which the I/O was being performed is terminated abnormally.

7.2.8 LOGREC Recording

Subsequent to each nontrivial hardware malfunction, various data are gathered and written into the SYS1.LOGREC data set through the **LOGREC recorder.** The LOGREC data set is always located on the system resident volume; it is initialized at system generation.

Records written onto SYS1.LOGREC describe the following malfunctions or unusual conditions:

- channel failures.
- I/O device failures.
- error summaries for I/O devices which — like the communications control processor — contain special hardware to record error information.
- main-memory failures.
- information record for each system loading (IPL).
- information record for system shutdowns.

When the SYS1.LOGREC data set is 90% full, OS IV/F4 asks the console operator to dump its contents onto another permanent medium, such as a

tape reel or a printed report. The SYS1.LOGREC data set can be processed by the JQQEREP0 service aid, which edits and prints LOGREC data and optionally captures its contents onto magnetic tape. **The JQQDIP00 service aid can initialize** SYS1.LOGREC and modify its space allocation.

7.3 DYNAMIC SUPPORT SYSTEM (DSS)

DSS provides effective maintenance and debugging procedures for OS IV/F4 control program routines. Maintenance personnel can use the special command language of DSS for detecting and correcting logical errors in their programs.

DSS is invoked when the console operator (or systems maintainer) presses the RESTART key located on the M series service processor. Once started, DSS supervises control-program execution until the operator enters a command to reset this supervisory function.

DSS offers the following services:

1) Debugging and correction of any of the following events:
- successful branch instruction.
- modification of a specified general-purpose register.
- instruction-fetch from a specified main-memory area, and
- modification of a specified main-memory area.

These events can be detected by the hardware program event recording (PER) feature.

2) Updating and/or display of real memory, virtual memory, and/or registers.
3) Accepting DSS commands at preselected points in a program.
4) Setting breakpoints at desired main-memory addresses for debugging purposes.
5) Submitting DSS commands from designated input devices (card reader, magnetic tape, or an operator console).
6) Displaying prespecified data on a DSS output device (line printer, magnetic tape drive, or an operator console).

7.4 SERVICE AIDS

The OS IV/F4 service aids are a group of programs to assist with diagnosis and correction of errors in the control program or user programs.

7.4.1 Service Aids for Gathering Diagnostic Data

JQLSADMP service aid
The JQLSADMP service aid dumps data from real memory or virtual memory. It is used for troubleshooting the control program. Appropriate output devices are line printers and magnetic tape drives.

Dumps may be of two kinds: direct dumps, and high-speed dumps. A **direct dump** formats real and virtual memory areas (either/both the common area and a particular address space) and displays them on a line printer.

A **high-speed dump** is used to capture the image of a real memory area without formatting. Data captured by a high speed dump can be formatted and displayed with the JQLPRDMP service aid at a later time.

This service aid (JQLSADMP) operates standalone; it does not require the OS IV/F4 control program.

Table 7.1 Service aids

Functional classification	Program name	Function
Gathering diagnostic data	JQLSADMP	Formatted dump of real or virtual memory.
	JQMGTF	Trace
Formatting and printing data sets and their elements	JQNLIST	Formatting and printing of load modules, object modules, etc.
	JQOJOBQD	Display the contents of the SYS1.SYSJOBQE data set.
	JQLPRDMP	Formatting and printing gathered data.
Correcting and updating programs	JQPPTFLE	Corrections to the OS IV/F4 control program
	JQPSPZAP	Corrections to load-module instructions or other data on a direct access volume
LOGREC management	JQQEREP0	Edit and display LOGREC data
	JQQDIP00	Initialize the LOGREC data set

JQMGTF service aid
The JQMGTF service aid traces the flow of the control program or a user program. Control-program events which can be traced by JQMGTF are as follows:
- I/O interruptions
- SVC interruptions
- program interruptions
- external interruptions
- Start I/O instructions
- switching of tasks by the OS IV/F4 dispatcher.
- VTAM trace information. Trace data are written into the SYS1.TRACE data set or a data set

specified by the user.

The user can select events to be traced in his program by issuing GTRACE macro instructions. Data captured on the SYS1.TRACE data set can be formatted and displayed by the edit function of the JQLPRDMP service aid.

7.4.2 Service Aids for Formatting and Printing Data Sets and Their Elements

JQNLIST service aid
The JQNLIST service aid formats and prints the contents of an object module, load module, the OS IV/F4 nucleus and/or link pack area, etc.

JQNLIST has the following output options:
- mapping object modules, including ESD cross references.
- mapping load modules, including ESD cross references.
- mapping the OS IV/F4 nucleus.
- software-change histories of selected control sections.
- mapping the LPA.

JQOJOBQD service aid
The JQOJOBQD service aid formats and prints the system job queue data set (SYS1.SYSJOBQE) and the work area data sets for one or more OS IV/F4 initiators (SYS1.SWADS).

Via this service aid, a trained system programmer can analyze software problems causing OS IV/F4 interruptions.

The following items from SYS1.SYSJOBQE can be displayed:
- total contents.
- contents of a specified input job class.
- contents of a specific output class.
- contents of the hold queue.
- contents of a specific job located in the input queue.
- jobs from a specific terminal user.
- job outputs awaiting return to a specific terminal user.
- jobs in the hold queue for a specific terminal user.

Similarly, the following data can be selectively printed out from a SWADS:
- total contents.
- contents by procedure name.

The JQOJOBQD operates stand-alone from the OS IV/F4 control program.

JQLPRDMP service aid
The JQLPRDMP service aid formats and displays the following data sets:
- high-speed dumps created by the JQLSADMP service aid.
- SYS1.DUMP data sets.
- DSS dump data sets, and
- trace data sets written by the JQMGTF service aid.

Optional elements of a JQLSADMP high-speed dump or DSS-written real-memory dump can be formatted and printed:
- all queue control blocks.
- Link pack area map.
- OS IV/F4 nucleus, system queue area, and common service area.
- system control blocks for a particular address space.
- system control blocks for all address spaces.
- real memory areas.
- virtual memory areas.
- address space of a user, plus his local system queue area.

After issuing a DUMP command. a user can request that his SYS1.DUMP data set be formatted and printed, including the following elements:
- virtual memory area for his job.
- system control blocks for his job.

Also, trace data gathered by JQMGTF can be formatted and printed.

7.4.3 Service Aids for Correcting and Updating Programs

JQPPTFLE service aid
The JQPPTFLE service aid corrects failures in user-written programs incorporated into OS IV/F4. It can also be used to correct/change portions of the OS IV/F4 control program. In either case, JQPPTFLE changes individual control sections of selected load modules temporarily, so as to verify their correct operation before permanently installing them.

JQPSPZAP service aid
The JQPSPZAP service aid corrects arbitrary data fields in DASD records. In particular, it corrects instructions and data within load modules as follows:
- Checks and corrects instructions and data in a **load module**, which is an executable member of a program library.
- Checks and corrects any data on DASD, for example, a damaged VTOC.
- Display some/all of the corrected instructions, data, etc. to confirm their validity.

7.4.4 LOGREC Functions

JQQEREP0 service aid
The JQQEREP0 service aid formats data from the SYS1.LOGREC data set and collects the data onto magnetic tape. This service aid can also format and print accumulated LOGREC data.

JQQDIP00 service aid
The JQQDIP00 service aid initializes the SYS1.LOGREC data set during system generation or reconstructs a damaged LOGREC data set. This

service aid can also change the space allocated to SYS1.LOGREC.

7.5 INDEPENDENT UTILITY PROGRAMS

These programs do not operate under control of OS IV/F4; they thereby run **stand-alone.** They are self-loading and perform all their own I/O operations. An independent utility is ordinarily used only when OS IV/F4 is inoperable or before OS IV/F4 has been successfully generated. The reader should consult the **FACOM OS IV/F4 Independent Utilities User's Guide** for additional details on these programs.

JQJDASDI utility
This independent utility initializes a direct access device or allocates an alternate track. It is typically used to correct a defective track on an initialized system volume such as the IPL volume.

JQJDMPRS utility
This independent utility dumps part/all of one direct access volume to another direct access volume or to magnetic tape. This utility is also used to restore the contents of a dumped volume.

7.6 HARDWARE DIAGNOSIS PROGRAM

The hardware diagnosis program is operated under control of OS IV/F4 or the maintenance operating system (MOS). MOS is a specialized operating system used for stand-alone maintenance of a hardware system; diagnostic programs run without using OS IV/F4. The online test control program (OLTEC) furnishes an interface between programs to be tested and OS IV/F4 or MOS.

Under OLTEC, individual diagnostic programs can run as if they were unrelated to OS IV/F4 or MOS.

Diagnostic programs have the following functions:

1) Test and diagnose
- I/O devices.
- the service processor (SVP).
- a communications control processor (CCP).
- a terminal device.

2) Detect errors due to device interactions.
3) Analyze machine-check logout data for the CPU, main memory, or channel to locate a malfunction.
4) Analyze SYS1.LOGREC data to determine which component has malfunctioned and print relevant statistics.

CHAPTER 8
SUPERVISOR

8.1 OVERVIEW

A Supervisor is a basic component of any operating system, whose principal functions are to issue I/O requests, respond to I/O interruptions, control executing tasks and respond to their requests for supervisory services, and maintain continuity of system operation. All OS IV/F4 system programs, as well as all batch and interactive programs, operate under control of the supervisor.

This chapter will outline certain basic hardware features of M series computers which are closely related to the OS IV/F4 Supervisor, then describe the principal elements of the Supervisor together with supervisory macro instructions which can be issued by user programs. Although the I/O Supervisor is an integral part of the overall OS IV/F4 Supervisor, it will be only briefly discussed here, since it relates directly to OS IV/F4 Data Management. For a full discussion of data management, the reader should consult the **FACOM OS IV/F4 Data Management Functions and Facilities.** For a detailed description of supervisory macro instructions, assembler-language programmers should read the **FACOM OS IV/F4 Supervisor Macro Instructions Reference Manual.**

A batch user defines his jobs in terms of **job steps** within which he executes **programs.** However, the Supervisor defines batch and interactive users in terms of **tasks, activities,** and **disabled routines,** which are discussed in Section 8.2.2. The OS IV/F4 Supervisor provides the following services to these program elements:

- Interruption control – Analyze the cause and useful consequence of each interruption.
- Task management – Attach, detach, and change the status of various system and user tasks.
- Virtual storage management – Allocate and deallocate virtual-storage areas as requested by system and user programs.
- Real storage management – Allocate page frames, read and write pages to DASD, and fix and release pages from page frames.
- Program management – Load and link programs within an address space.
- Management of serially reusable resources – Sequence requests for such resources in an orderly and efficient fashion.
- Time management – Manage time intervals (as requested by various tasks) plus the time of day clock.
- Program-interruptions service – Respond to software-caused program interruptions.
- Program dumping and snapshotting.
- I/O supervisor – Support the execute channel program (EXCP) macro instruction and associated appendage routines: also, support the M series channel-DAT feature.

8.2 OPERATION

This section discusses major aspects of how the OS IV/F4 Supervisor operates, beginning with the range of hardware interruptions issued by M series hardware. The supervisor must respond to every interruption carefully, thoroughly, and efficiently. Finally, this section describes how OS IV/F4 defines multiprogramming and achieves a high level of throughput by multiprogramming batch and interactive jobs.

8.2.1 Interruptions

An **interruption** is a solicited/unsolicited change of instruction sequence—a program is executing a sequence of pre-planned instructions when some internal/external event occurs which should be immediately serviced by the operating system supervisor. **External events** include I/O interruptions, external signal interruptions (including lapsation of predetermined time intervals), and machine check and program-restart interruptions. **Internal events** are those initiated by the executing program such as issuing a supervisor call instruction or executing an instruction causing a program check interruption. For a detailed description of causes and hardware actions for interruptions, the reader should consult the **FACOM M series Hardware Reference Manual.** Table 8.1 summarizes the types and causes of interruptions.

Table 8.1 Types of interrruptions

Interruption	Cause(s)
Machine check interruption	Hardware failure in CPU or main storage
Program interruption	Execution of a program resulted in an unusual event such as an operation, memory-protection, or page-fault exception.
SVC interruption	A Supervisor call instruction was executed.
External interruption	1. Interrupt key was pushed on the service processor (SVP) panel. 2. The CPU received a signal processor instruction (SIGP) 3. Preset time interval has elapsed
I/O interruption	Completion of part/all of an I/O operation
Restart interruption	1. Service processor (SVP) generated a restart command. 2. The CPU received SIGP restart subcommand.

As a consequence of each type of interruption, M-series hardware starts an instruction sequence in the OS IV/F4 supervisor called an **interruption handler,** of which there are six:

- MCH (Machine check handler).
- Program interruption handler.
- SVC interruption handler.
- External interruption handler.
- I/O interruption handler.
- Restart interruption handler.

Each executing task can selectively set mask bits for interruptions other than the SVC and restart interruptions. For program interruptions, four mask bits are available to control acceptance/rejection of interruptions when certain arithmetic overflow/underflow conditions occur. All other interruptions are unconditional as far as user programs are concerned.

A typical timeline for interruption processing appears in Fig. 8.1. Before each interruption is analyzed, the corresponding interruption handler saves data stored in the old PSW area in the OS IV/F4 nucleus. The old PSW is moved into another storage area, so that it can be protected from premature overwriting. Then the cause of interruption is analyzed in detail:

- For example, a program interruption can result from an overflow in a decimal-arithmetic operation or an addressing error. During analysis of the interruption, its **code** (two-byte identifier) is checked(1).
- When analysis of the interruption code is completed, the supervisor branches to a routine corresponding to the interruption cause(2).
- When the interruption has been fully serviced, saved data are restored into their prior registers and storage areas, fields of the old PSW are loaded into the current PSW, and the supervisor returns control to the interruped program (3).

When executing in the supervisor, a CPU is in the hardware-defined **supervisor state;** otherwise, it is in the **problem program state.**

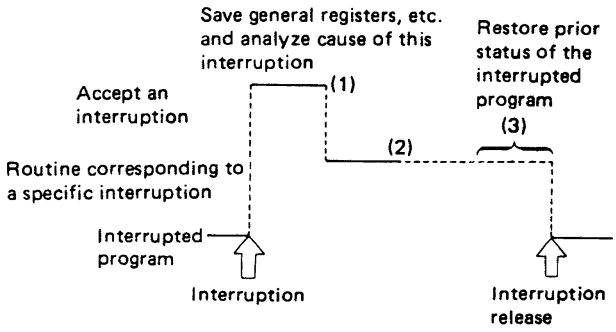


Fig. 8.1 Timeline of an interruption

8.2.2 Tasks, Activities, and Disabled Routines

There are the three types of CPU contenders, as seen by the OS IV/F4 Supervisor: tasks, activities, and disabled routines.

Tasks

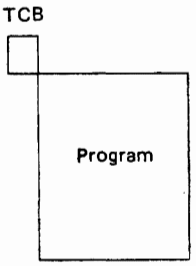
Every active batch or interactive program in an OS IV/F4 system at a given instant comprises one or more **tasks,** each of which is a collection of resources (including one executing program) competing for CPU service. A job step ordinarily initiates one task; in the COBOL or FORTRAN languages, a programmer can specify only one execution sequence at a time. However, in the assembler and PL/I languages, he can specify two or more tasks which are to compete with one another simultaneously for CPU time.

One task is created in each address space by OS IV/F4 as a result of initiating execution of a job step (the **job step task**). The user can optionally create additional tasks in his program. If he does not, the job step task is the only task in his address space. The benefits of a multiprogramming environment are still available even with only one task in a job step; work is performed for other address spaces when this task is waiting for an event such as completion of an input operation.

The advantage of creating additional tasks within a job step is that more tasks can compete for CPU control. When a wait condition occurs for one task in an address space, it is not necessarily a task from some other address space that gets control; it may be another task in the same space.

The supervisor uses a **task control block** (TCB)—created and retained in the **local system queue area** (LSQA) of the corresponding address space —to control each task.

The TCB describes the task status, its resources, dispatching priority, etc. Hereafter, a task will be represented as shown below: a TCB (small box) and a program (large box).



The general rule is that parallel tasks within a job step (subtasks) should be created only when a significant amount of overlap between two or more subtasks can be achieved. The amount of overhead required by the OS IV/F4 Supervisor for establishing and controlling subtasks, and the increased effort to coordinate subtasks and provide communications between them, must be taken into account.

A new task is created when the user (or system routine) issues an ATTACH macro instruction. The task issuing the ATTACH macro instruction is the **originating task**; the newly-created task is a **subtask** of the originating task. The subtask competes for control in the same manner as any other task in the system, on the basis of its priority (both address space priority and task priority within the address space) and its current ability to use the CPU.

Activities

In a typical prior operating system the supervisor classified all CPU-using routines as either **tasks** or **disabled routines**, the latter performing uninterruptible functions within the supervisor. This two-way classification has proved to be cumbersome and inadequate for certain supervisory routines, which are not conventional user-program tasks yet need not execute disabled for interruptions. Hence, OS IV/F4 defines a third category for CPU-using routines, an **activity**, which is described below.

When a supervisory routine executes as a task, it must be managed by the task dispatcher, which imposes substantial overhead on the CPU and may delay rapid responses to interruptions. Yet, if this same supervisory routine runs disabled for interruptions, it causes presentation of these interruptions to be delayed, which slows system throughput and responsiveness. Also, a multiple-address-space operating system like OS IV/F4 should execute with interrupts disabled as infrequently and briefly as possible, especially in multiprocessor configurations where delays of inter-CPU communications can stall both CPUs.

The concept of an activity solves many of these problems. An activity is functionally quite similar to a supervisory routine which executes disabled; the principal difference is that an activity runs enabled for I/O and external interruptions. When interrupted, the supervisor schedules a routine to process the interruption, then returns control immediately to the interrupted activity. Even if the interruption could cause another activity or task to gain a higher dispatching priority, the supervisor returns to the interrupted activity. In this regard, activities are different from tasks; the supervisor may dispatch a different task (or activity) after one task has been interrupted.

As shown in Fig. 8.2, most disabled time required by prior operating systems is assigned to interruptible activities in OS IV/F4, greatly reducing hardware delays to interruptions. The shaded area in this figure indicates what fraction of CPU activity must execute disabled for interruptions. Just as tasks are managed by TCBs, activities are managed by **service request blocks** (SRBs).

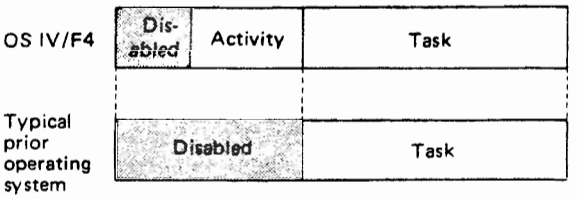


Fig. 8.2 Enabled/disabled fractions of CPU time

Disabled routines

The OS IV/F4 Supervisor uses no special control tables for disabled routines; hence, they cannot be controlled by the dispatcher.

The number of disabled routines has been greatly reduced in OS IV/F4, and most system functions execute as tasks or activities most of the time. Disabled routines include supervisor functions where rapid processing strongly influences total system efficiency, functions managing tasks and activities, and interruption handlers.

8.2.3 Flow of Control

When a hardware interruption occurs, the supervisor analyzes its cause, then commences any required processing. Thereafter, the supervisor usually gives control to one of its own routines, the **dispatcher**, which selects the activity or task with highest execution priority. After this selection, the supervisor gives CPU control to the selected activity or tasks. This function is known as **dispatching.**

Multiprogramming

One of the major purposes of an operating system is to improve the efficiency of using system resources. The CPU is usually the most important system resource — most costly, most heavily used, and necessary for all batch/interactive jobs. Hence, OS IV/F4 has been designed to utilize the CPU as effi-

ciently as possible. If a program is allowed to use a CPU exclusively until it completes executing the CPU often idles while the program is awaiting completion of an I/O operation. In order not to waste this idle time, OS IV/F4 permits another program to use the CPU while the first program is waiting. Thus, the CPU is used by two or more programs alternatively. Viewed externally, several batch and/or interactive programs are running in parallel, which is known as **multiprogramming** or **multitasking.**

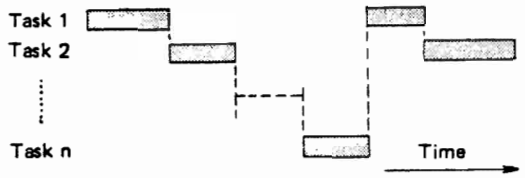


Fig. 8.3 Profile of multi-task CPU operation

"Multiprogramming" is often used to denote "multijob processing"—parallel processing of batch jobs.

Dispatching

As described above, the dispatcher is a routine within the supervisor which selects an activity or task to receive CPU control. The selected activity or task usually has the highest dispatching priority of any which are ready to execute.

There are really three priorities to consider: address space priorities, task priorities, and subtask priorities.

Address space dispatching priority

Each batch job is initiated in an address space. All successive steps in the job execute in the same address space. The address space has a **dispatching priority;** OS IV/F4 normally assigns and alters this priority in order to achieve the best overall balance in the system—that is, in order to make the most efficient use of CPU time and other system resources.

It may be desirable for some jobs to execute at a different address space priority than the default priority assigned by OS IV/F4. To assign a priority, the user codes DPRTY=($value_1$, $value_2$) on his EXEC statement.

The address space priority is then determined as follows: address space dispatching priority = $(value_1 \times 16) + value_2$.

Once an address space dispatching priority has been set, it can be altered only by OS IV/F4. (Thus, there is no limit priority associated with an address space.) However, a user can set a new address space priority for succeeding job steps by specifying different values in DPRTY parameters on corresponding EXEC statements.

Task dispatching priority

OS IV/F4 associates with each task in an address space a **limit priority** and a **task dispatching priority.** OS IV/F4 sets these priorities when initiating each job step. If the user creates other tasks in his address space by issuing ATTACH macro instructions, he can give them different limit and dispatching priorities by LPMOD and DPMOD parameters, respectively.

Dispatching priorities of tasks in an address space do not affect the sequence in which jobs are selected for execution; the latter is based on address space dispatching priorities. Once OS IV/F4 has selected an address space for dispatching, the dispatcher selects the highest priority task ready to execute. Thus, task priorities may affect processing within an address space. In a multiprocessor configuration, task priorities do not guarantee the sequence in which tasks execute, since more than one task may be executing simultaneously in the same address space on different CPUs. In a paging environment, page faults also alter the order in which tasks execute.

Subtask dispatching priority

When a subtask is created, its limit and dispatching priorities are the same as those of the originating task, unless the subtask priorities are modified by LPMOD and DPMOD parameters of the ATTACH macro instruction. The LPMOD parameter specifies a decrement to be subtracted from the current limit priority of the originating task. The result of the subtraction is assigned as the limit priority of the subtask. If the result is zero or negative, zero is assigned as the limit priority. The DPMOD parameter specifies the number to be added to the current dispatching priority of the originating task. The result of the addition is assigned as the dispatching priority of the subtask, unless the number is greater than the limit priority or less than zero. In the latter cases, the limit priority or 0, respectively, is used as the dispatching priority.

Assigning and changing dispatching priorities

Tasks requesting a high rate of I/O operations should be assigned higher priority than tasks with little I/O, since the former will be in wait state for a greater amount of time. Lower-priority tasks execute when higher-priority tasks are in wait state.

A user can explicitly change the priorities of his subtasks by issuing CHAP macro instructions. Each CHAP macro instruction changes the dispatching priority of an active task or one of its subtasks by adding a specific positive or negative value. The dispatching priority of an active task can be reduced below that of another task. When this occurs, and if the other task is dispatchable, the OS IV/F4 Supervisor gives it control after servicing the CHAP macro instruction.

In each address space, an activity has higher dispatching priority than any task or subtask.

A CHAP macro instruction can increase the limit priority of any subtask of an active task. An active task cannot change its own limit priority. The dispatching priority of a subtask can be raised above its own limit priority, but not above the limit of the originating task. When the dispatching priority of a

subtask is raised above its own limit priority, the subtask's limit priority is automatically raised to its new dispatching priority.

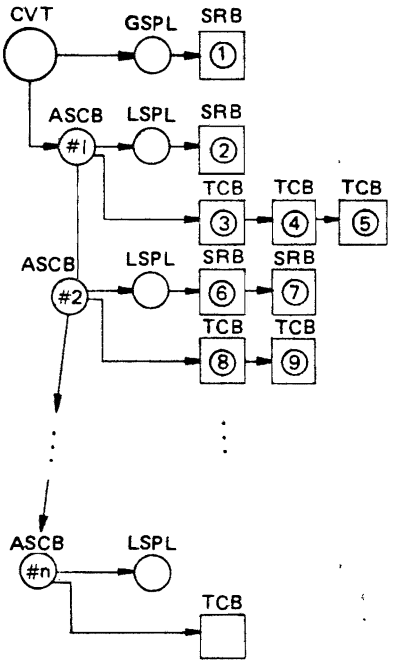


Fig. 8.4 Example of dispatching priorities

Fig. 8.4 shows a collection of activities and tasks at a particular instant in an OS IV/F4 system. In this figure, address space priorities are highest at the top, lowest at the bottom. Likewise, task priorities within each address space are highest at the left, lowest at the right. The following abbreviations have the indicated meanings:

ASCB Address space control block, one for each address space

CVT Communication vector table, which resides in the OS IV/F4 nucleus and points to the GSPL and the highest-priority ASCB

GSPL Global service priority list, which queues **global activities** (those not related to a single address space)

LSPL Local service priority list, which queues local activities for each address space

SRB Service request block, one for each activity

In this figure, the instantaneous priority order among all activities and tasks is shown by circled numbers. For example, if the first SRB (to which the GSPL points) is ready to execute, it will receive CPU control as soon as the dispatcher gains control. If activities (1) and (2) are both awaiting I/O completions, the task designated by (3) will receive CPU control when the dispatcher next gains control, etc.

WAIT macro instruction

A program issues a WAIT macro instruction to move from active to wait state.

Fig. 8.5 shows the sequence of CPU control resulting from a WAIT macro instruction issued by Task A:

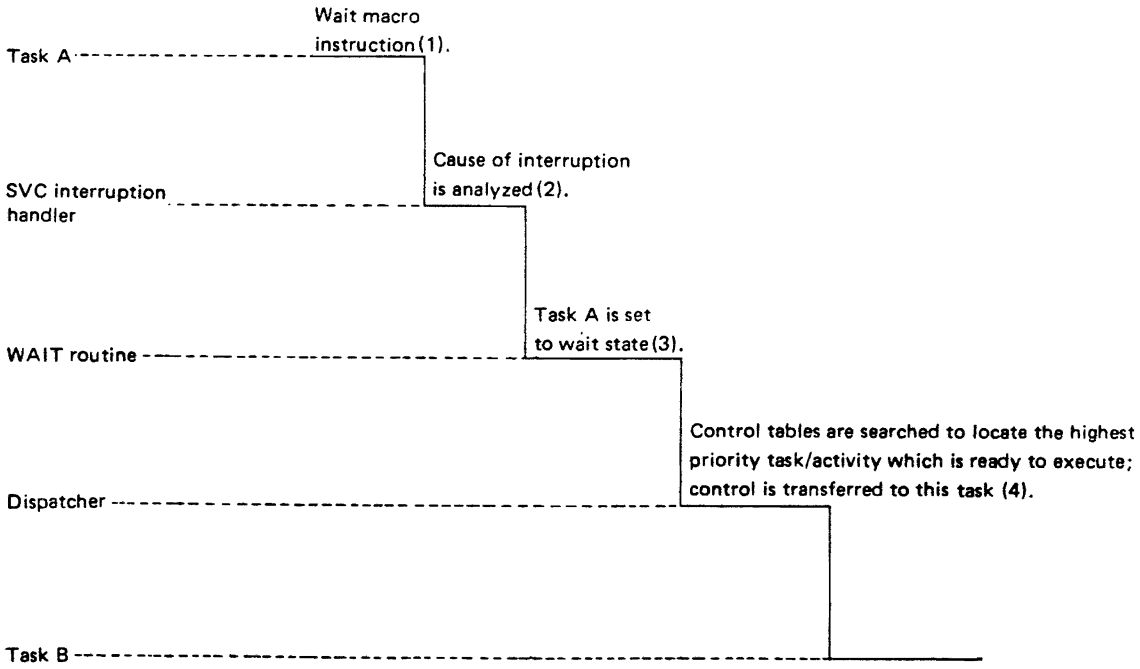


Fig. 8.5 Processing a WAIT macro instruction

(1) The WAIT macro instruction issues a supervisor call instruction, which generates an SVC interruption.
(2) As described in Section 8.2.3, the SVC interruption causes CPU control to be transferred to the SVC interruption handler, which checks the interruption code, then transfers control to the supervisory routine which processes WAITs.
(3) If the specified event has not yet been completed, this wait routine sets the task being executed (i.e., which issued the WAIT macro instruction) into wait state.
(4) When the wait routine completes, it yields control to the dispatcher, which searches the OS IV/F4 tables of activities and tasks (similar to that depicted in Fig. 8.4) for the highest-priority task and transfers control to this task (Task B in Fig. 8.5).

SVC routines

Each SVC routine receives control from the SVC interruption handler. For most installations, all SVC routines are already defined within OS IV/F4. However, an installation can add its own SVC routines during system generation. SVC routines are operated in supervisor state with a protection key of 0.

An SVC routine can be **resident** (Type 1 or Type 2) or **nonresident.** Table 8.2 shows where and how various types of SVC routines are executed.

Table 8.2 Attributes of SVC routines

Other Attributes	RESIDENCY AND TYPE OF ROUTINE		
	Resident		Nonresident
	Type 1	Type 2	Types 3 and 4
Location in each address space	Nucleus	Nucleus	Link pack area
Supervisory control	Disabled routine	Subtask of this address space	Subtask of this address space
Accept interruptions?	NO	YES	YES
Issue other SVC macro instructions?	NO	YES	YES

8.2.4 Automatic Priority Group (APG)

When several unrelated jobs are executing concurrently, it usually happens that some are **CPU-limited** (use the CPU heavily, I/O devices lightly), some are **I/O-limited** (use the CPU lightly, I/O devices heavily), and the remainder are relatively **balanced** (CPU usage overlaps considerably with I/O usage, neither exceeding the other by a large factor). If the user requests a high dispatching priority for a CPU-limited job, his job will occupy the CPU rather heavily and not permit lower-priority jobs – some of which may be I/O limited – to issue their I/O requests. Hence, it is generally desirable for CPU-limited jobs to have lower dispatching priorities than I/O-limited jobs, with balanced jobs falling in between.

APG is a feature of the OS IV/F4 Supervisor which automatically raises/lowers dispatching priorities of tasks according to their recent relative usage of the CPU and I/O devices. The user can choose (with his JCL statements) whether his job should be processed in APG or outside of APG. If outside, his job will run with a dispatching priority which is invariably higher – or invariably lower – than all jobs in APG. Urgent jobs needing fast turnaround should be submitted with DPRTY parameters above the APG range, in certain cases. CPU-limited jobs which can be processed on an overnight basis may be submitted with DPRTY parameters below the APG range. However, most batch jobs at most OS IV/F4 installations should be included in APG, since it achieves high overall utilization of the CPU and I/O devices yet does not degrade turnaround times for most batch jobs.

Each OS IV/F4 installation selects a priority range for its APG, which is set during system loading (IPL) or by the default value specified during system generation. Each job step will be assigned the APG dispatching priority unless it explicitly requests a higher or lower priority with a DPRTY parameter.

For APG, a short time interval (typically 1 – 3 seconds) is defined as the **APG interval.** Whenever this interval has elapsed, the OS IV/F4 Supervisor inspects recent usage of the CPU by all tasks. Any task which has used the CPU throughout the entire interval is considered to be CPU-limited (at least

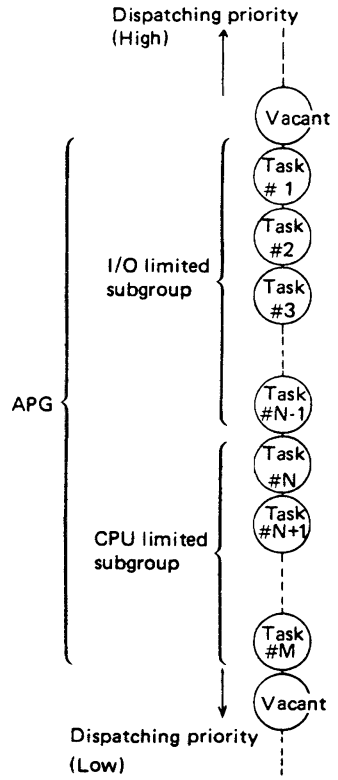


Fig. 8.6 Example of APG dispatching priorities

temporarily). Any task which has used the CPU lightly in an interval is considered to be I/O-limited; often this task issues a WAIT macro instruction soon after receiving CPU control during the interval, so that it can no longer execute. If two or more CPU-limited tasks are ready to execute in an interval, and if the configuration is a **uniprocessor** (one CPU), only one task can control the CPU throughout this interval. Other CPU-limited tasks "appear" to be I/O-limited, and OS IV/F4 raises their dispatching priorities accordingly.

The above ordering of tasks is illustrated in Fig. 8.6.

As each job step is initiated, the dispatcher first determines if its assigned dispatching priority is above, within, or below the APG range. If above, this step executes with higher CPU priority than any task shown in Fig. 8.6; if below, the step executes with lower priority than any in this figure. If the priority of the step is within the APG range, the dispatcher assigns the job a priority just above the highest task in the I/O-limited subgroup shown in Fig. 8.6. Its priority movement thereafter depends on the following algorithm.

Each task in the I/O-limited subgroup will use some or all of the APG interval when it receives control of the CPU:

(1) It may use the entire interval without losing control of the CPU;
(2) It may issue a WAIT macro instruction early/late in the interval, thus yielding control to another task; or
(3) Another higher-priority task may become dispatchable, in which case the dispatcher will assign the CPU to the higher-priority task.

In case (1), the original task will be moved to the top of the CPU-limited subgroup during the following APG interval; in Fig. 8.6 this is just ahead of Task #N. In case (2), the task moves to a position within the I/O-limited subgroup depending on what fraction of the interval it actually used the CPU. In case (3), the rank of the original task is unchanged, but it is allocated only an abbreviated interval of CPU control, which equals the time it lost to the higher-priority task.

Tasks in the CPU-limited subgroup execute as follows, using the same three cases as for the I/O-limited subgroup. In case (1), a task using a full APG interval is moved to the bottom of the subgroup, just below Task #M. Hence, all other CPU-limited tasks move up one position, in a round-robin sequence:

Execution of task #N	**Positions within the CPU-limited subgroup**
Before	N N+1 N+2 . . . M
After	N+1 N+2 . . . M N

If no other task becomes dispatchable during the previous APG interval, the Dispatcher will then give control to Task #N+1 if it is ready to execute — which is quite likely. If Task #N+1 also uses its full interval of CPU time, the round-robin sequence continues:

Execution of task #N+1	**Positions within the CPU-limited subgroup** high low
After task #N+1 executes	N+2 N+3 M N N+1

For a CPU-limited task in case (2), the task is moved to the bottom of the I/O-limited subgroup, just below Task #N-1. In case (3), the CPU-limited tasks are not resequenced but the highest-priority task is given only the remainder of its APG interval before the APG algorithm is again executed.

The APG algorithm gives higher dispatching priority to tasks which are chronically I/O-limited. It gives lower priority to CPU-limited tasks. It continually reevaluates each active task to determine whether it has become more CPU-limited than previously, become more I/O-limited, or remained the same.

The length of the APG interval is itself automatically revised by OS IV/F4. If the relative ranking of tasks changes slowly, the APG interval is lengthened; if the relative ranking changes quickly—which is due to many tasks of differeent attributes executing concurrently—the APG interval is shortened. The frequency with which the APG interval is automatically modified—and the time increment by which it is lengthened or shortened, plus maximum and minimum values for the APG interval — are set during system generation, optionally modified when the system is loaded (IPL).

8.2.5 Multiprocessor Configurations

Multiprocessing is usage of two or more CPUs to process a common workload. A **tightly-coupled multiprocessor** configuration (called a **multiprocessor** in the remainder of this chapter) comprise two CPUs connected to common pools of main-memory modules and peripheral devices. In addition to raising the throughput rate of an installation considerably above that of a **uniprocessor** (one CPU) configuration, a multiprocessor has much higher inherent reliability; if one CPU malfunctions briefly, its workload can be assumed entirely by the other CPU. Even if the malfunction is fairly serious, full OS IV/F4 facilities can be kept operational on the other CPU. Furthermore, the surviving CPU can perform significantly more powerful diagnostic procedures on the malfunctioning CPU than the latter could perform on itself, which facilitates rapid isolation and repair of the malfunction.

When a uniprocessor processes multiple tasks concurrently, its activity profile is like the example in Fig. 8.7:

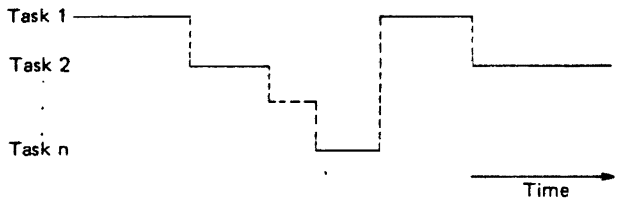


Fig. 8.7 Multiprogramming with a uniprocessor

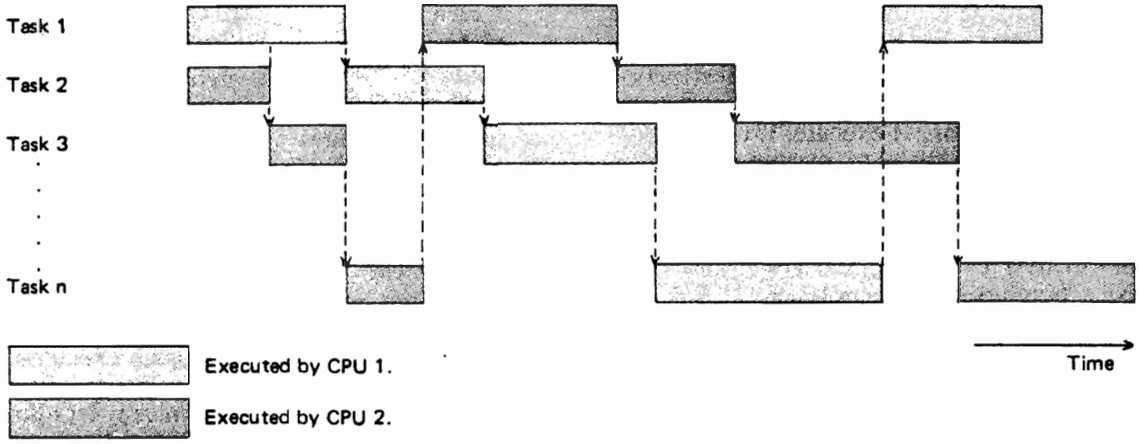


Fig. 8.8 Multiprogramming with a multiprocessor

In each time interval, at most one task can be active, even though other tasks are receiving I/O services while they await control of the CPU.

With a multiprocessor, two tasks can be simultaneously active, as shown in Fig. 8.8.

OS IV/F4 fully supports multiprocessor M series configurations. Each interruption is presented to one of the CPUs. Only one copy of OS IV/F4 is needed—in particular, one supervisor—and most supervisory functions can be executed by either CPU, even concurrently with one another.

Some inherent difficulties and hardware solutions for multiprocessors are summarized in the following sections.

Prefixed storage areas

A prefixed storage area (PSA) is a 4K-byte area of main storage hardware-connected to exactly one CPU. (All other storage is connected to both CPUs.) This area is required for specialized hardware operations, such as PSW-interchanges and diagnostic logouts. In particular, the PSA stores new and old PSW's for each type of interruption, channel status word (CSW), channel address word (CAW), interruption codes, and so on.

A PSA is required for each CPU of a multiprocessor. Since the PSA always has addressed 0 – 4095 for its CPU, the two PSAs must be distinguished from one another. Each PSA is identified by a **prefix register** located at the top of the prefix area.

Each CPU uses **real addresses** which correspond in almost all cases to the **absolute addresses** of the hardware configuration. The relationship between real and absolute addresses is illustrated in Fig. 8.9. In this example, the PSA for CPU A has absolute

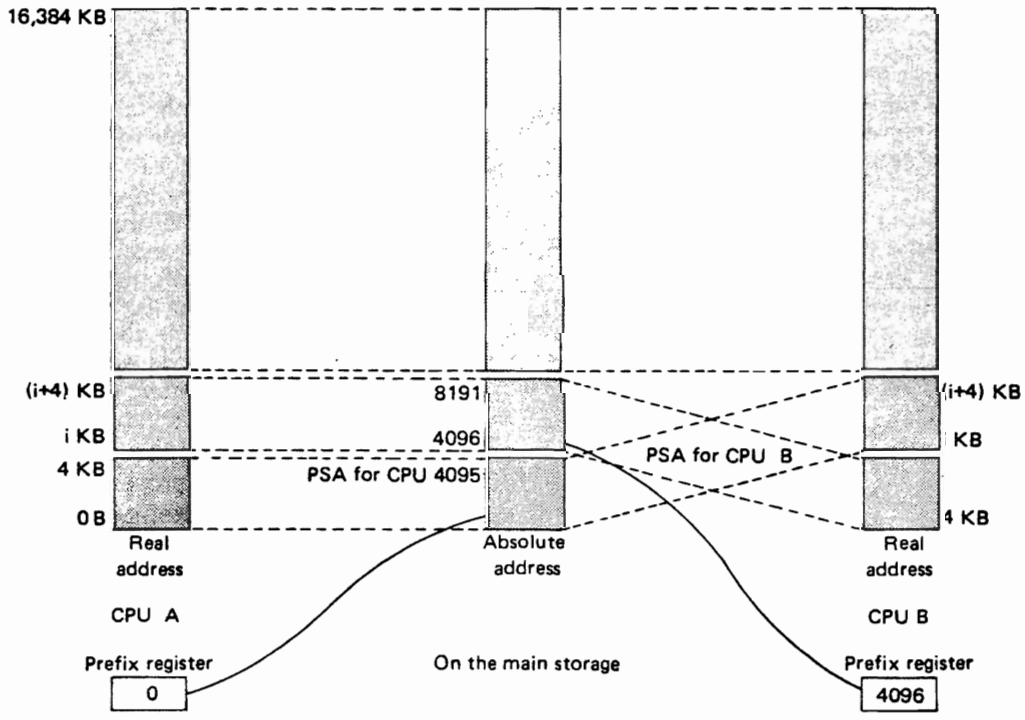


Fig. 8.9 Prefixed storage areas and prefix registers

addresses 0—4095 because its prefix register points to 0. If the prefix register for CPU B points to absolute address 4096; absolute addresses 4096 — 8191 become the PSA for CPU B.

The user cannot directly reference or update prefix registers or absolute addresses.

The reader should note that this discussion is entirely independent of virtual storage concepts and virtual addresses. The DAT feature translates virtual addresses into real addresses. Most real addresses are the same as corresponding absolute addresses, but PSA addresses (0 — 4095) are often translated into absolute addresses other than 0 — 4095, depending on the contents of the corresponding prefix register.

Communication between CPUs

Since two CPUs run concurrently in a multiprocessor, synchronization and communication are required when updating shared system resources. When rewriting a page table, for example, translation lookaside buffers (TLBs) for both CPUs must be cleared. When an I/O device is connected to only one of the CPUs and the other CPU requests an operation on it, the two CPUs must communicate directly.

Communication between CPUs is initiated by an external interruption in one CPU, caused by executing a signal processor (SIGP) instruction in the other CPU. Since an external interruption is maskable, the first CPU may delay receiving this interruption, although typically only for a fraction of a second. Thus the requesting CPU may await completion of communication, or it may start other processing which is independent of this communication. The above TLB example requires synchronized communication but the above I/O request example does not require synchronization.

Communication between CPUs is fully controlled by the supervisor, and the user need not be conscious of it—or of any hardware/software aspect of multiprocessing, for that matter.

Disabled routines in a multiprocessor

In a uniprocessor configuration, any system resource can be exclusively controlled by a routine which executes with interruptions disabled. In a multiprocessor system, exclusive control cannot be guaranteed merely by disabling interruptions; even if one CPU operates disabled, resources can be updated by the other CPU.

Fig. 8.10 shows an example of resource contention where CPU A executes disabled but CPU B nonetheless updates a control block being accessed by CPU A.

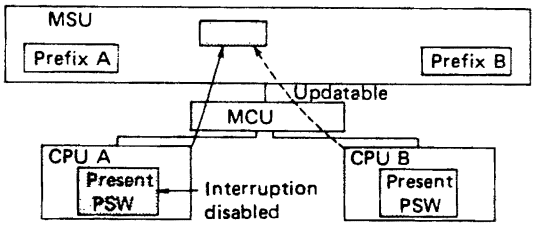


Fig. 8.10 Resource contention in a multiprocessor

To prevent uncontrolled and dangerous contention for resources, OS IV/F4 furnishes several software **locks** on system resources so that these resources can be accessed by two CPUs in an orderly sequence. When accessing any serially-reusable system resource, one CPU locks the resource. If the other CPU attempts to access this resource shortly thereafter, it finds the locked resource unavailable. The second CPU can then take one of two actions: await unlocking of the resource by the first CPU, or perform other activities prior to attempting access again.

In OS IV/F4, sophisticated locking/unlocking of system resources contributes to efficient and safe multiprocessing.

Alternate CPU recovery (ACR)

Described in Section 7.2.2, ACR is a useful hardware/software feature of OS IV/F4. If one CPU of a multiprocessor malfunctions, and if its hardware instruction retry (HIR) and error checking and correction (ECC) hardware-recovery circuits are unable to continue normal operations, this CPU issues a signal processor instruction to advise the other CPU of its (serious) malfunction. The healthy CPU then continues as many tasks, activities, etc. as possible on behalf of the first CPU, terminating abnormally only those tasks whose programs, data areas, register contents, etc. were irreparably damaged by the malfunctioning CPU. ACR is facilitated by the unique set of main-storage tables shared by both CPUs; except for the PSA of the malfunctioning CPU, the healthy CPU utilizes precisely the same real addresses and virtual storage maps as the malfunctioning CPU.

8.3 TASK MANAGEMENT

OS IV/F4 can execute a number of subtasks concurrently within a single job step. In order to perform this function effectively, OS IV/F4 furnishes various supervisor macro instructions and associated exit routines. The supervisory routines supporting these macro instructions and exit routines are called task management collectively, described in the following sections.

8.3.1 Attaching and Detaching Tasks

Attaching a task using ATTACH

By issuing an ATTACH macro instruction, a user requests OS IV/F4 to assign to a new taks resources, authority to compete for the CPU, and the opportunity to be enqueued for CPU control if necessary. Every time a user program (or system program) issues an ATTACH macro instruction successfully, the supervisor creates a new TCB pointing to the associated resources, etc. This TCB is linked to an appropriate queue of TCBs, so that it can compete for CPU control thereafter.

The task issuing the ATTACH macro instruction is called the **attaching tasks,** and the tasks created by the ATTACH macro instruction is called the **attached task** or **subtask.** The attached task can itself issue ATTACH macro instructions to create additional subtasks, as illustrated in Fig. 8.11.

All tasks in a job step compete independently for CPU time; if no constraints are provided, the tasks are performed and terminated asynchronously. However, since each task is performing a portion of the same job step, some communication and constraints between tasks are required, such as notifications when subtasks complete. If a user attempts to a predecessor task before all of its subtasks are complete, those subtasks and the predecessor task are abnormally terminated by the supervisor.

The task management information in this section is required only for establishing communications among tasks in the same job step. The relationship of tasks in a job step is shown in Fig. 8.11. The horizontal lines in this figure separate originating tasks and subtasks; they have no bearing on task priority. Tasks A, A1, A2, A2a, B, B1 and B1a are all subtasks of the jobstep task; tasks A1, A2, and A2a are subtasks of task A. Tasks A2a and B1a are the lowest-level tasks in the job step. Although task B1 is at the same level as tasks A1 and A2, it is not considered a subtask of task A.

Task A is the originating task for both tasks A1 and A2, and task A2 is the originating task for task A2a. A hierarchy of tasks exists within the job step. Therefore, the job step task, task A, and task A2 are predecessors of task A2a, while task B has no direct relationship to task A2a.

The ATTACH macro instruction issued by a task contains a supervisor call instruction. The Supervisor prepares a TCB and places it into the address space for the job step task. If the first program of the attached task is not already in virtual storage, the Supervisor loads it before passing control to the attached task.

When issuing an ATTACH macro instruction, the user can specify a dispatching priority for the attached task as long as it is within the limits already specified for this job class and initiator.

Terminating a task

Each task returns control to the supervisor after it completes all required processing. The attaching task loads the address of a termination-processing routine into general register 14 prior to issuing the ATTACH macro instruction. The end of each executable program is indicated by a RETURN macro instruction (or equivalent higher-level language verb, such as END, RETURN, STOP, etc.). The RETURN macro instruction creates a branch instruction using the address provided in register 14. Ending a program is performed by a RETURN macro instruction not only in case it was called by an ATTACH macro instruction but also when it was called by a LINK or CALL macro instruction, as described in Section 8.6.3.

Fig. 8.12 shows the flow of a task from ATTACH to RETURN.

When a task terminates, all of its system resources (program area and so on) should be released. The ECB and ETXR parameters of the ATTACH macro instruction assist communication between a subtask and its originating task. These parameters indicate

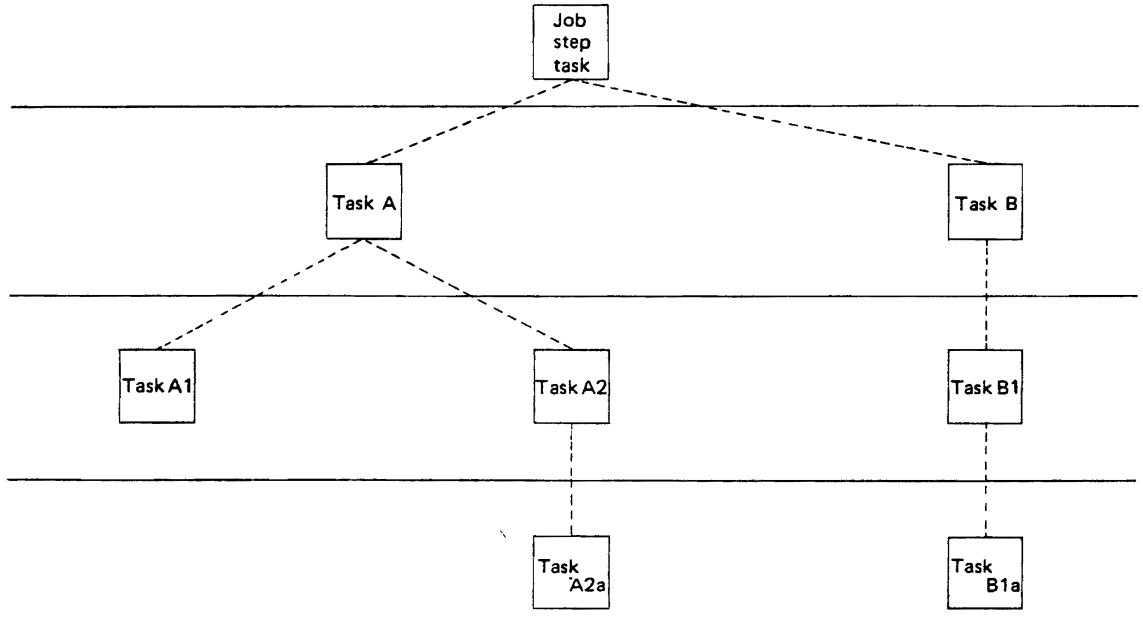


Fig. 8.11 Levels of tasks in a job step

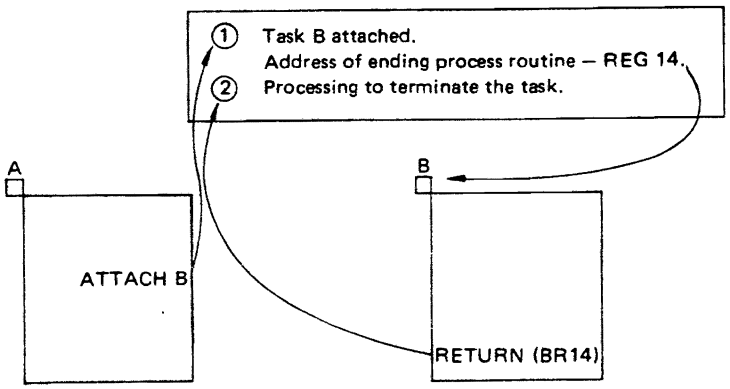


Fig. 8.12 Attaching and return of task

normal or abnormal termination of a subtask to its originator. If ECB and/or EXTR parameters are coded in an ATTACH macro instruction, the subtask TCB is not removed from OS IV/F4 when the subtask is terminated. The originating task must remove this TCB from the system after termination by issuing a DETACH macro instruction. TCBs for all subtasks must be removed before the originating task can terminate normally.

The ETXR parameter specifies the address of an **end-of-task exit routine** in the originating task, which receives control when the subtask being created terminates. The end-of-task routine is given control asynchronously after the subtask has terminated and must therefore be in virtual storage when it is required. After the supervisor terminates the subtask, the specified end-of-task routine is scheduled to be executed. It competes for CPU time using the priority of the originating task and of its address space; it can receive control even though the originating task is in wait state. Although the DETACH macro instruction does not have to be issued in the end-of-task routine, this is a good place for it.

The ECB parameter specifies the address of an **event control block** which is posted by the supervisor when the subtask terminates. After posting, the event control block contains the completion code specified for the subtask.

Detaching a task

Detaching a task is a supervisor action to delete the corresponding TCB.

If neither an ECB nor an ETXR parameter is specified in the ATTACH macro instruction, the supervisor removes the subtask TCB from the system when the subtask terminates. Its originating task does not have to issue a DETACH macro instruction. A reference to the TCB in a CHAP or a DETACH macro instruction in this case is risky, as is task termination; since the originating task is unaware of subtask termination, the user might refer to a TCB which has been removed from the system, which would cause the active task to be abnormally terminated.

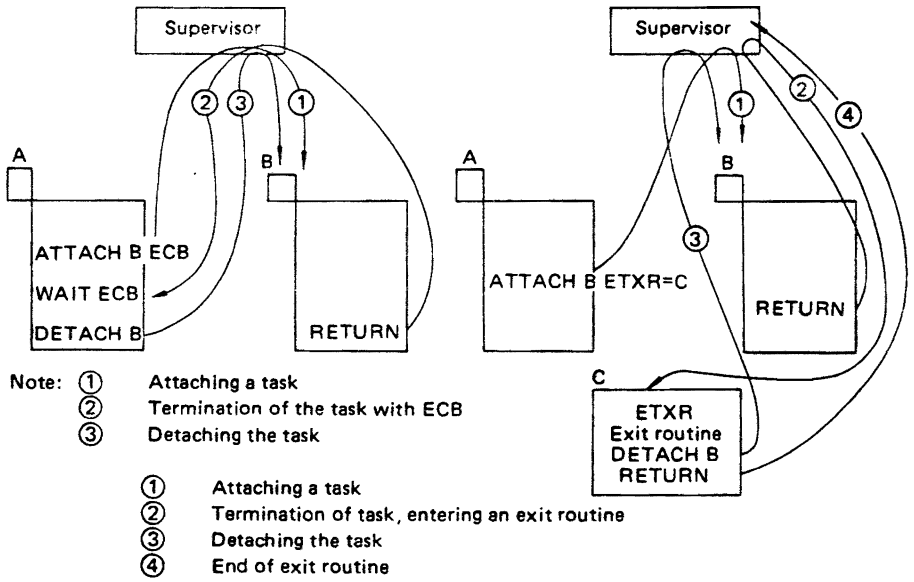


Fig. 8.13 Attahing, terminating and detaching a task

Fig. 8.13 shows the flow of attaching, terminating, and detaching a task. The left side of this figure shows the flow when an ECB has been specified; the right side shows the flow when DETACH is issued by an exit routine.

Tasks for ordinary batch jobs

Job steps run in the sequence specified by JCL statements, after an initiator has selected the job.

The first task in a job step is called the **job step task.** Starting a job step is no more than the supervisor's attaching a job step task by issuing an ATTACH macro instruction. The first program to be executed by a job step task is specified by the EXEC statement. When the job step task terminates, the supervisor notifies the corresponding initiator to start termination processing for the job step.

8.3.2 Processing flow for an abnormal task

A task can abnormally terminate itself or its parent job step by issuing an ABEND (abnormal end) macro instruction.

The supervisor will terminate a user task if it detects an error in conjunction with the task. In this case, the supervisor issues an ABEND macro instruction on behalf of the task. Thus, abnormal termination is requested by an ABEND macro instruction in any case.

The user can intercept an abnormal termination for his task by furnishing an **abnormal end exit routine.** For this purpose, he can furnish a STAE macro instruction (specify task abnormal exit) or a STAI (subtask abnormal intercept) parameter in his ATTACH macro instruction, as shown in Fig. 8.14.

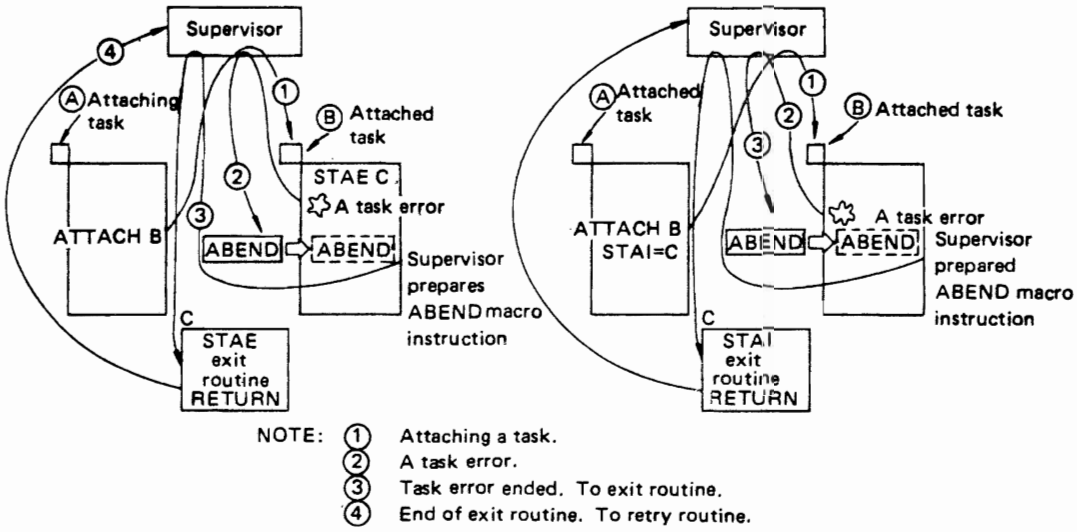


Fig. 8.14 STAE and STAI facilities

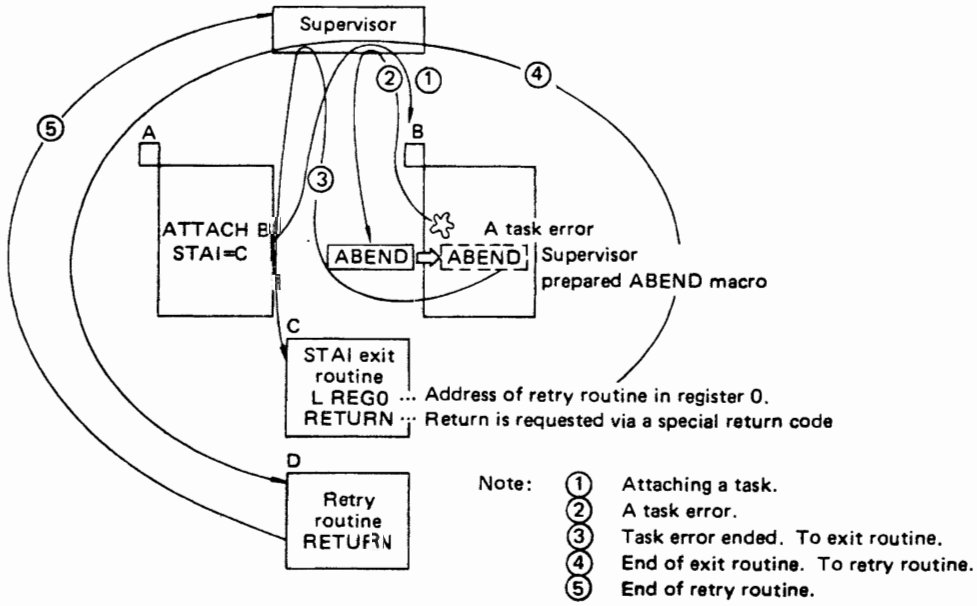


Fig. 8.15 Retry routine

By issuing a STAE macro instruction in his program, the user requests that control be transferred to a specified exit routine if his task misbehaves. By specifying a STAI parameter in his ATTACH macro instruction, the user requests that his exit routine be entered in such a situation.

Abnormal-end exit routines are useful for analyzing causes of error and evaluating extent of program/data damage before tasks terminate. In some cases, the user can recover from the abnormal condition by branching to a retry routine, as shown in Fig. 8.15. (Retry can often be requested by furnishing a certain return code in a RETURN macro instruction in the exit routine.)

Either a STAE or a STAI exit routine is executed on behalf of the task which issued the ABEND macro instruction.

In general, any exit routine must be linked into the load module containing the program which incurred the ABEND condition.

8.3.3 Status of a Task

From the viewpoint of the supervisor, tasks can be classified as follows:

- Active— using a CPU
- Ready— ready to use a CPU, but not currently executing
- Wait state— waiting for completion of one or more I/O operations or another task
- Nonexecutable— execution inhibited by the Supervisor or another task
- Terminated— execution of all programs completed.

Transitions among these conditions are shown in Fig. 8.16, and macro instructions to control these transitions are described thereafter.

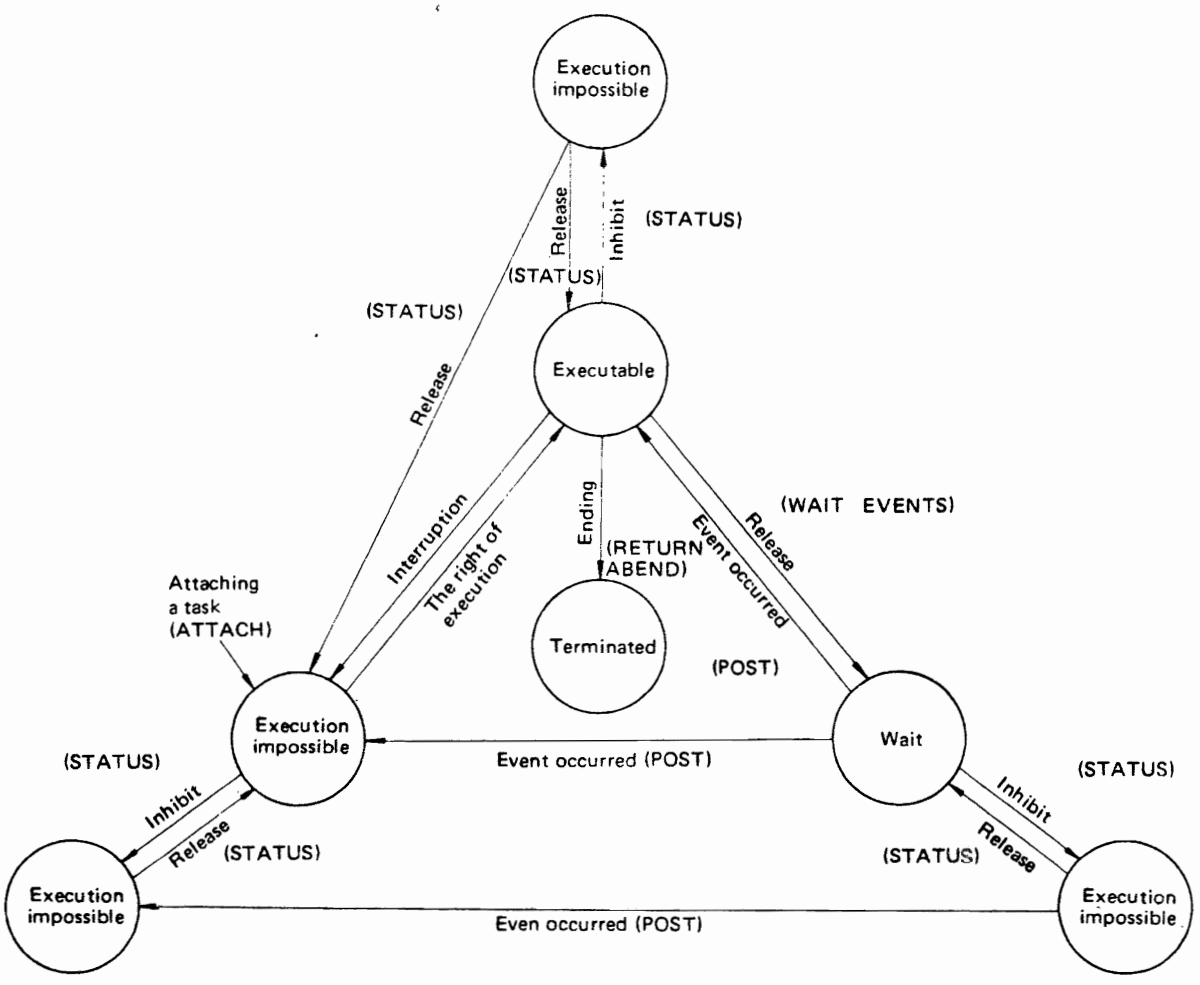


Fig. 8.16 Task status transitions

WAIT and POST macro instructions
A program issues a WAIT macro instruction to move from active to wait state. A program (including a system program such as the supervisor) issues a POST macro instruction to make another task ready (if it was in wait state).

Moving from active to wait state is at the initiative of the active task, typically when it must stop executing until one or more I/O operations have completed, etc. These **events** are formally designated to the supervisor by various tasks; each WAIT macro instruction names one or more events which must occur (or must already have occurred) prior to this task's resuming execution.

A POST macro instruction changes the status of one event to "completed." After an event has been posted, the supervisor regains control—often at once, sometimes considerably later—and enters its dispatcher routine, which will now consider the posted task to be ready rather than waiting.

Most WAIT macro instructions "issued" by user jobs are actually issued by OS IV/F4 data management, especially on behalf of COBOL, FORTRAN, PL/I, and ALGOL users. Even with the Assembler language, a user typically issues GET, PUT, CHECK, and other macro instructions rather than explicit WAIT macro instuctions. GET, PUT, etc. link to data management routines which issue WAIT macro instructions on behalf of the user program. Likewise, the I/O supervisor issues most POST macro instructions, which result from completion of I/O operations requested by system/user programs.

In certain situations, a user may wish to await the completion of any one of several concurrent operations. For example, he may issue several I/O requests in rapid succession whereupon he must wait for at least one I/O operation to complete. He then issues a **multiple-events WAIT** macro instruction pointing to several ECBs. When any one of these ECBs is posted, his task is moved from wait to ready state. More generally, a multiple-events WAIT macro instruction can specify that the task becomes ready when any M of N events complete as shown in Fig. 8.17. In this example, the task becomes ready when two out of three events are completed.

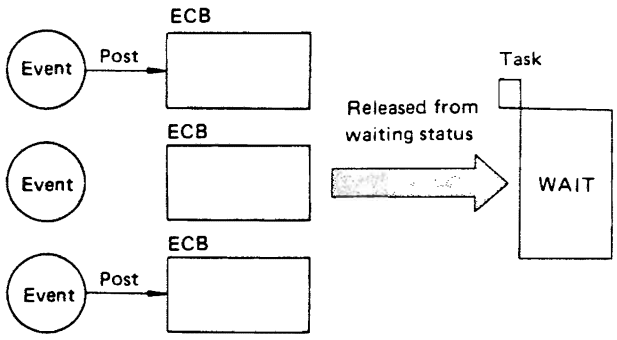


Fig. 8.17 Example of a multiple-events WAIT macro instruction

EVENTS macro instruction
OS IV/F4 furnishes an EVENTS macro instruction, which is functionally similar to a WAIT macro instruction but imposes much less CPU overhead.

STATUS macro instruction
This macro instruction designates an executable task as non-executable, or vice versa; the reader should review the beginning of Section 8.3.3 for explanations of these terms. An **executable task** is one that is active, ready, or in wait state; a **non-executable task** is one that has been explicitly suspended by the OS IV/F4 control program or a controlling task.

8.3.4 Other Task-Management Facilities

These include macro instructions for changing dispatching priorities and extracting task information.

Change dispatching priority
A user can change the dispatching priority of a task by issuing a CHAP (change priority) macro instruction. The CHAP macro instruction is valid for the issuing task or any of its subtasks.

Extraction of task information
By issuing an EXTRACT macro instruction, a user can move copies of selected fields from a control block (such as his TCB) into his own region. Hence, EXTRACT does not change any control blocks or task status; it merely retrieves information into a more convenient region and format.

8.4 VIRTUAL STORAGE MANAGEMENT

The virtual storage management of OS IV/F4 is a component of the supervisor which manages multiple address spaces. It processes requests for small, medium, or large areas issued by the OS IV/F4 control program or user programs.

Virtual storage management includes the following activities:

- Allocating and releasing each address space and its LSQA (local system queue ares).
- Allocating and releasing regions.
- Allocating and releasing the SQA (system queue area) region within LSQA.
- Allocating and releasing areas within regions.
- Allocating and releasing areas within the CSA (common service area).

This section describes how user-region areas are allocated and released. Chapter 1 describes the overall structure of an address space and how it is created. The user of a high-level language such as COBOL or FORTRAN need not be aware of how areas in his region are allocated and released; the

supervisor automatically secures any necessary areas for him, based on his job control statements. When an assembler-language user needs to acquire or release a work area within his region, he issues the supervisor macro instructions GETMAIN or FREEMAIN, respectively.

The GETMAIN macro instruction allocates an area in an address space, the FREEMAIN macro instruction releases an area.

8.5 REAL STORAGE MANAGEMENT

Virtual storage is allocated, released, etc. by the OS IV/F4 virtual storage management routines. **Real storage** (also called **main storage**) is managed by the OS IV/F4 Paging Supervisor

As described in Chapter 1, virtual storage architecture permits each user to execute programs which use address spaces much larger than real storage. Typically, several – even dozens – of these users are executing concurrently on an OS IV/F4 configuration. Hence, real storage management must furnish sufficient **page frames** (4K-byte blocks of real storage, aligned on 4K-byte address boundaries) so that an efficient number of users are executing. In particular, high-priority tasks should be allocated more page frames—in proportion to their need for frames—than low-priority tasks, so that they can execute longer between page faults and hence complete faster.

The paging supervisor is a component of the OS IV/F4 Supervisor and resides in the system nucleus. It manages real storage of up to 16 megabytes and **external page storage** (EPS) on up to 16 DASDs.

OS IV/F4 furnishes several macro instructions for managing program pages and real-storage page frames, of which the PGRLSE (page release) macro instruction is available to any program; it releases the designated page frame for other users after the paging supervisor has handled it appropriately. A user may issue PGRLSE when he knows that a page he has previously defined—but not necessarily used—will not be referenced again for a protracted period—say, at least 20 seconds. Possibly he knows that the page will never be used again during this job step. PGRLSE is an optional facility by which users can assist the paging supervisor to work more efficiently.

Use of other page-frame macro instructions such as PGFIX and PGFREE is restricted to the OS IV/F4 control program and authorized system programmers; these facilities are cited in Chapter 1 of this part plus the **FACOM OS IV/F4 System Programmer's Guide.**

8.6 PROGRAM MANAGEMENT

This supervisor function manages all executable programs (and other load modules) to be loaded into address spaces. Only the OS IV/F4 nucleus is not loaded by program management; the nucleus is loaded by the IPL program when OS IV/F4 is initially loaded. Program management includes the following functions and features.

- Loading and deletion of programs.
- Control of overlay segments.
- Management of potentially sharable programs: reentrant, serially reusable, and other types of load modules.
- Support of dynamic link structures.
- Support of the PSECT facility.
- Support of the authorized program facility (APF).

8.6.1 Program Libraries

There are five types of program libraries in OS IV/F4, as follows:

- System library (SYS1.LINKLIB)
  This library contains all system programs (compilers, assembler, utility programs, and service aids) not contained in the SYS1. LPALIB and SYS1. SVCLIB.
- Job library
  Each batch job can define a job library with a special DD statement whose name field contains "JOBLIB". It contains load modules potentially useful for several steps of this job.
- Step library
  Each job step can define a step library with a special DD statement whose name field contains "STEPLIB". This library contains load modules used only in this step.
- Task library
  A user specifies a task library by furnishing a TASKLIB parameter when attaching a subtask. The **task library** is used when loading the first program of the subtask and if the attached task issues LOAD, LINK or XCTL macro instructions. The attaching task is responsible for opening the task library.
- Private library
  Program libraries other than the above types are called **private libraries.** When a user issues a program loading request (LOAD, ATTACH, and so on) for a private library, the latter must be specified in the DCB parameter of this macro instruction.

  The DCB macro instruction must be assembled into the user's program, which is also responsible for opening the private library.

When a user issues a program-loading request, program management searches these libraries in the following sequence:

No DCB in program loading request
Modules in JPA (job pack area)
Task library for module requesting task.
Task libraries of higher-priority tasks in the same address space.
Step library (or job library)
Modules in the various link pack areas
SYS1.LINKLIB.

DCB parameter furnished in program-loading request
Modules in JPA.
Private library specified by DCB parameter.
Modules in the various link pack areas.
SYS1. LINKLIB

8.6.2 Usage Attributes of a Load Module

Each load module can be assigned one of the following **usage attributes** at the time it is link-edited:

- Reentrant
  Read-only program, which issues GETMAINs for any work areas it needs. Can be used simultaneously by an arbitrary number of tasks.
- Serially reusable
  Self-initializing program. Cannot necessarily be used simultaneously by more than one task, but can be reused (from the start of the program).
- Nonreusable
  A module which cannot be used safely more than once after loading from its library. Hence, a fresh copy must be loaded each time this module is used.

The programmer assigns a usage attribute to his load module at link-edit time. Neither the linkage editor nor program management can verify whether he has assigned the correct attribute to this module.

Program management retrieves load modules with these attributes as follows:

- Re-entrant modules
  An arbitrary number of tasks are permitted to use a reentrant module. Once loaded, a reentrant module is not deleted from an address space even when all requesting tasks relinquish control of it. The user can delete this module by issuing a DELETE macro instruction.
- Serially reusable module
  When several tasks in one address space simultaneously request a serially reusable module, OS IV/F4 services them serially according to their priorities.
  Similar to a reentrant module, a serially reusable module is not deleted from an address space even when all requestors have completed using it. The module can be deleted by a DELETE macro instruction.
- Nonreusable module
  This module is loaded every time it is requested. After use, this module is deleted from the address space.

8.6.3 Program Management Macro Instructions

Program management furnishes the following macro instructions:

- LOAD
  Module with the specified entry point is loaded.
- DELETE
  Module previously requested by LOAD macro instruction is deleted from this address space.
- CALL
  Control is transferred to a specified entry point. When the called routine completes executing, it usually returns control to the program which issued the CALL macro instruction. This macro instruction utilizes the BALR (branch and link) instruction.
- LINK
  Module with the specified entry point is loaded into this address space, and it immediately receives CPU control from the linking module. Control normally returns to the program which issued the LINK macro instruction.
  If a serially reusable module is requested by a LINK macro instruction, this request is processed by the supervisor. If a serially reusable module is requested by a LOAD or CALL macro instruction, the user is responsible for controlling its access.

- XCTL (Transfer Control)
  Module with the specified entry point is loaded into this address space, and control is transferred to the loaded module. This program overwrites the requesting program, so that control cannot be returned to the requestor.
- IDENTIFY
  A new entry point is defined for the specified load module after it has been loaded.
- SEGLD (Segment Load)
  Requests loading of a specified program segment asynchronous with continued execution of the requesting program.
  If the requestor tries to transfer control to the segment during the course of loading, his execution is delayed until loading is completed.
- SEGWT (Segment Wait)
  Requests loading of a specified segment; control is not transferred until the loading is completed.

Differences in program control between LOAD, CALL, LINK, XCTL, and ATTACH macro instructions is shown in Fig. 8.18. The circled numbers indicate the sequence of CPU control. Since ATTACH permits parallel CPU control, its circled numbers are duplicated.

8.6.4 Dynamic Link Structures

The OS IV/F4 Supervisor permits programs to have a dynamic link structure. With this structure, load modules need not be linked fully by the linkage editor; they can be partially/totally linked during execution.

For example, an external reference is written in the assembler language as shown below:

```
L      15,=V(SUB1)
BALR   14, 15
```

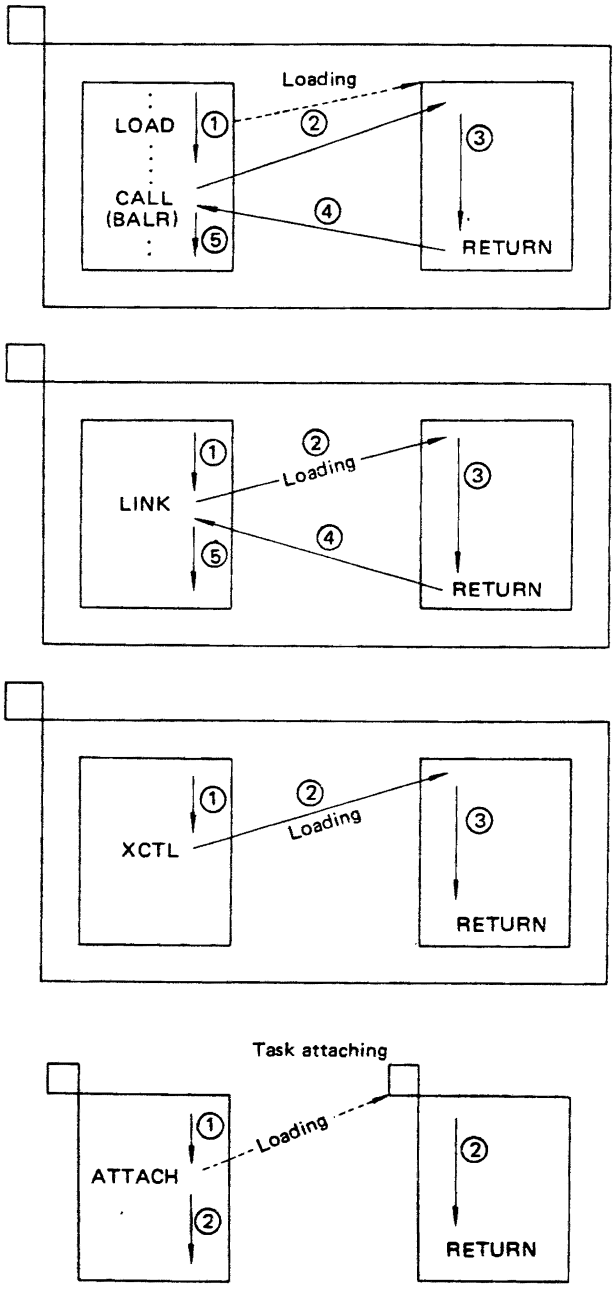


Fig. 8.18 LOAD, CALL, LINK, XCTL, and ATTACH macro instructions

If a user specifies the DYNAMIC attribute during link-editing of this reference, the linkage editor creates a dynamic rather than a static link structure, as illustrated in Fig. 8.19.

For a **static link structure,** the linkage editor locates the SUB1 entry address and uses it to satisfy the V(SUB1) parameter shown above; this statically links the subroutine with the main routine.

For a **dynamic link structure,** the address of the dynamic linking table (DALTAB) is stored into the V(SUB1) field, and DALTAB is linked with the main routine. After the BALR instruction has been executed during the first linkage to this subroutine, the supervisor gains control. If SUB1 has not been loaded, the supervisor loads it into the address space. Then, the supervisor stores its address into DALTAB before branching for the first time to SUB1. These steps correspond to (1)–(3) in Fig. 8.19. When SUB1 is next called, control flows directly to it without any supervisor intervention, since the address of SUB1 is now in DALTAB.

In addition to a static link or dynamic link structure, the user can design a **dynamic program structure** with LINK, ATTACH, LOAD, and XCTL macro instructions. Linkages are established during execution. These different structures can also be created with programs written in higher level languages. Static link or dynamic link structures are selected by linkage editor parameters.

To create a dynamic program structure in COBOL, a verb equivalent to the LOAD macro instruction is available. Verbs equivalent to LOAD and ATTACH are available to PL/I programmers.

8.6.5 Prototype Control Sections (PSECTs)

In addition to designating load modules as reentrant or serially reusable, a user can ask OS IV/F4 program management to provide PSECT support to reentrant programs.

In prior operating systems, the user was obliged to issue a GETMAIN macro instruction to secure a work area. Furthermore, his program had to initialize this work area before the latter could satisfy a reentrant program. OS IV/F4 solves this overhead problem by introducing the concept of a prototype control section (PSECT).

When the supervisor gives control to a load module containing a PSECT, in response to a LINK macro instruction, it automatically allocates a work area to the PSECT which it then copies into this area. Hence, this work area is preinitialized by the OS IV/F4 Supervisor as shown in Fig. 8.20. So long as the work area is difined entirely within this PSECT, the requested program is automatically reentrant, thereby easing its preparation and usage.

In order to compile reentrant programs, OS IV/F4 compilers for FORTRAN, COBOL, and PL/I generate PSECTs for their work areas.

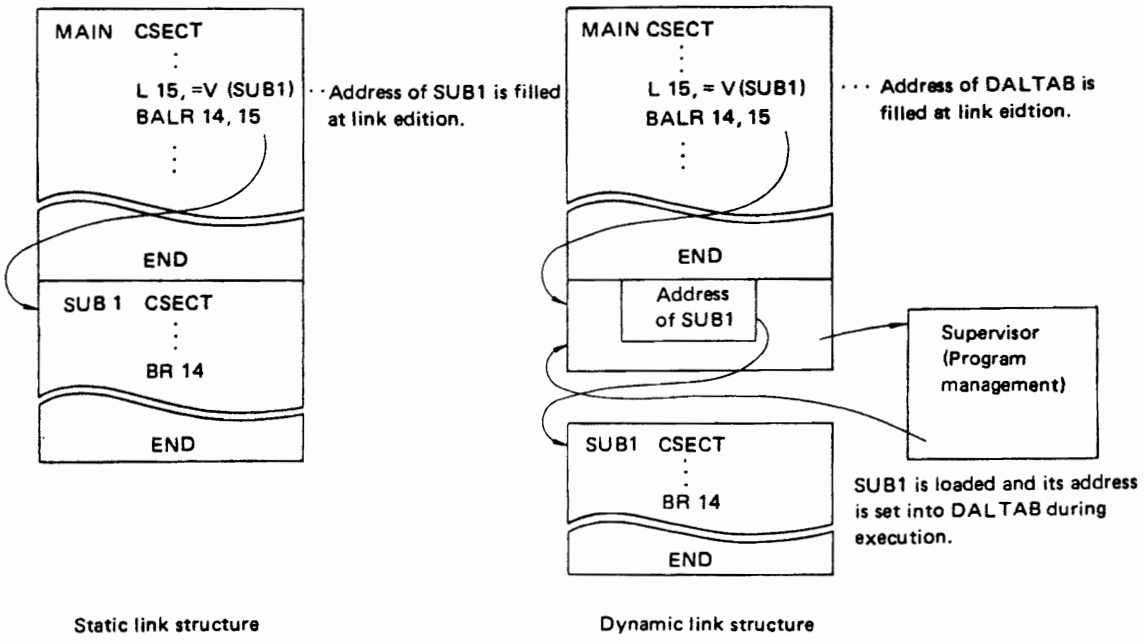


Fig. 8.19 Static and dynamic link structures

8.6.6 Authorized Program Facility

For ordinary user jobs, the OS IV/F4 Supervisor furnishes several macro instructions. Other supervisory macro instructions are defined in OS IV/F4 which strongly influence operation and performance of the system; these facilities and related utility programs are not authorized for most user programs, although they are usually available to system programmers at an installation.

The OS IV/F4 Supervisor offers the authorized program facility (APF), which permits an installation to limit certain programs and macro instructions to users furnishing a special password (**authorization**).

8.7 MANAGEMENT OF SERIALLY REUSABLE RESOURCES

When several tasks attempt to simultaneously access a serially reusable resource, their sequence of accesses must be carefully controlled; otherwise the resource can be damaged and/or the tasks can cause one another to fail.

The OS IV/F4 component to control this function is known as serially reusable resource management.

The supervisor offers the ENQ and DEQ macro instructions to request use and release of serially reusable system resources. Requests issued via ENQ macro instructions are accepted in FIFO sequence (first in, first out). These reequests are further classified into exclusive requests and sharing requests. When requests are mixed, they are

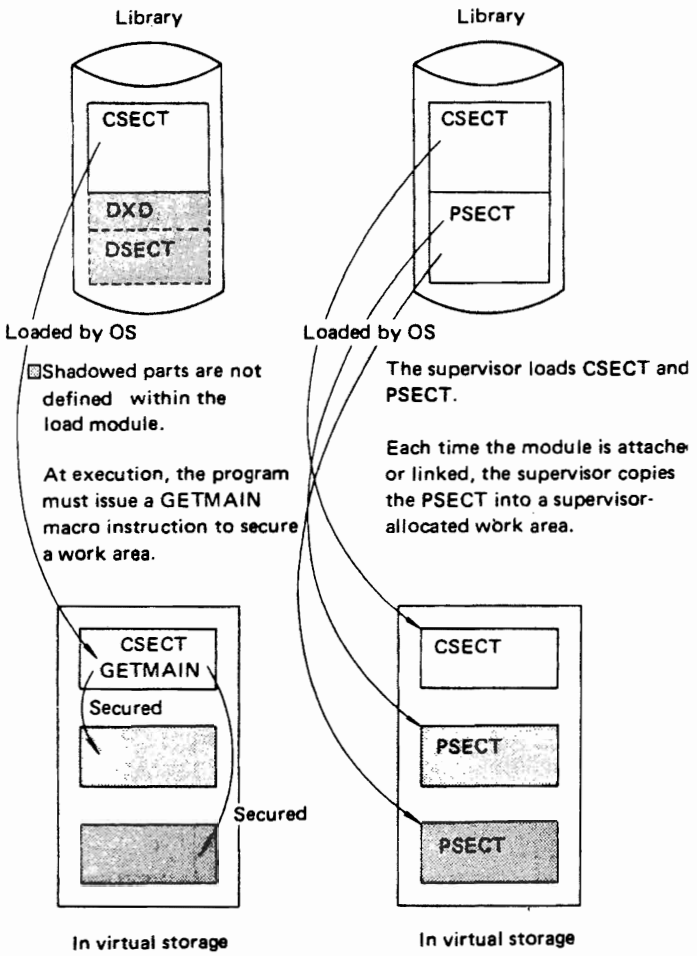


Fig. 8.20 Reentrant program with/without PSECTs

accepted in the sequence shown in Fig. 8.21. A **sharing request** is for read-only access to a resource. An **exclusive request** solicits permission to update the resource.

When all current requests for a particular resource can share it, no special control is required, since the resource can be used simultaneously. When requests include at least one which is exclusive, sharing requests must be included in the same queue as exclusive requests. In this case, priority to access the resource can be established by ENQ macro instructions specifying the same queue name for exclusive and sharing requests.

Specifying shared or exclusive data sets is facilitated by the DISP parameter of the DD statement. The initiator allocates and deallocates a data set based on the DISP parameter of the corresponding DD statement.

If all DD statements in one job which reference a particular data set specify DISP=SHR, the data set can be shared simultaneously with other tasks. Otherwise, it must be exclusively controlled by this job. For two different computer configurations to share DASDs, RESERVE macro instructions are often needed instead of ENQ macro instructions. The RESERVE macro instruction issues an ENQ macro instruction and also issues a RESERVE channel command to attempt to capture this DASD for the duration of a job. The corresponding DEQ macro instruction notifies the I/O supervisor to issue a RELEASE channel command to this DASD prior to performing the normal dequeuing operation.

8.8 TIMER MANAGEMENT

OS IV/F4 timer management provides time of day and time-interval services by using the hardware TOD (time of day) clock, the clock comparator, and the CPU timer. The following macro instructions are offered by timer managment:

- TIME
  Hours, minutes, seconds, and fractional seconds (optionally). In addition to the Julian year and day format, the date can be presented in the form YYMMDD (year, month, day).
- STIMER
  Sets a time interval after which an external interruption will be generated. Time is measured selectively, either when the requesting task is executing (**task time**) or continuously (**real time**). When the elapsed time reaches the preset interval, control is transferred to the timer exit routine of the requesting task. An event completed signal can be requested when a specified time interval is reached, by furnishing an ECB. Tasks can also be set into wait state for specified time intervals.
- TTIMER
  Checks remaining time until the STIMER function will generate a timer interruption or cancels a previously requested timer service.

In addition to the above macro instructions, timer management monitors the following time intervals for the control program:

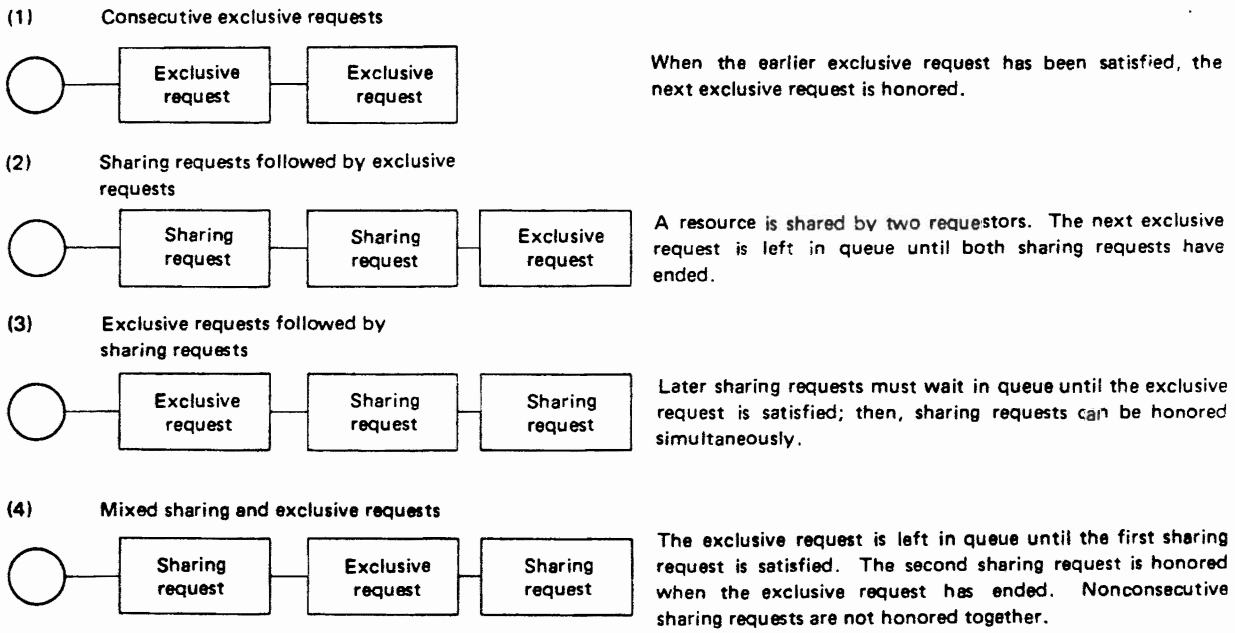


Fig. 8.21 Examples of sharing and exclusive requests

- Monitoring CPU time used by a job or job step
  When a TIME parameter has been specified in an EXEC statement or JOB statement, timer management generates a timer interruption when the job or job step exceeds the specified value.

- Calculation of CPU idle time
  The total idle time of a CPU is accumulated every 10 minutes; the cumulative statistic is used as an installation-management aid via SMF.
- Time intervals for page management
  Control is transferred to page management every time a pre-determined interval elapses, to check the paging rate per time interval.
- Date change interval
  This interval is used to change the date every midnight.
- Automatic Priority Group (APG) interval
  Every time this interval elapses, a slice of CPU time is allocated to an APG job.

8.9 PROGRAM INTERRUPTION PROCESSING

The OS IV/F4 Supervisor handles all program interruptions, subject to certain user options for handling program interruptions. Program interruptions are originated by M series hardware when the CPU or channel detects an illegal instruction, operand, datum, or channel command.

The user can optionally preplan how to handle program interruptions such as an operation exception, memory-protection exception, or data exception by specifyying an exit routine with an SPIE (specify program interruption exit) macro instruction. The flow of control for a SPIE routine is shown in Fig. 8.22. Data are exchanged between the interrupted program and the exit routine through control areas known as the PICA (program interruption control area) or PIE (program interruption element).

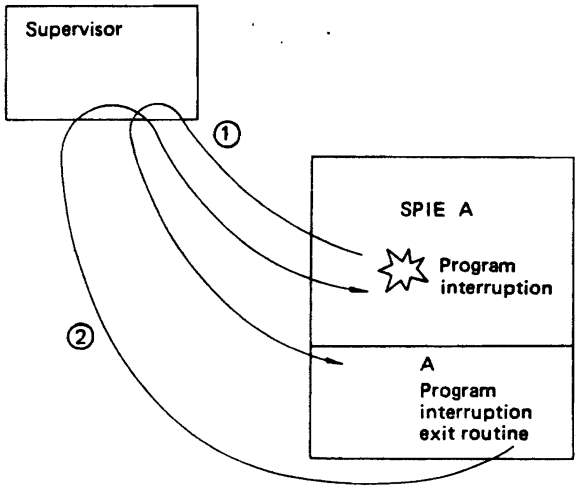


Fig. 8.22 Program interruption exit routine

8.10 PROGRAM DUMPING

The program-dumping facilities of OS IV/F4 includes the following:

- SNAP and DUMP facilities
  A user can issue a SNAP macro instruction to copy part or all of an address space to a specified data set in an edited format. This function is also performed when a DUMP parameter is specified in an ABEND macro instruction.
- SVC dumping function
  With this function, a user can dump the contents of an address space onto magnetic tape or DASD without any formatting, dumps can later be edited and printed by using the JQLPRDMP service aid.

  This function is actuated when a DUMP is keyed in from console.

  In addition, the stand-along JQLSADMP service aid will dump part or all of real storage or virtual storage onto tape. It is capable of dumping storage either formatted or unformatted for printing.

CHAPTER 9
SYSTEM GENERATION

9.1 OVERVIEW

System generation is the procedure for creating an operating system tailored to the needs of a particular installation. System generation accepts the detailed description of a particular hardware configuration plus a large number of OS IV/F4 software options.

Fujitsu furnishes several library tape reels which contains all available software functions, plus a starter system reel used to generate the initial operating system.

Generating a complete operating system
One tpe of system generation creates a full-function operating system. Except for minor I/O equipment changes, this type of system generation is invariably used. For example, circumstances requiring generation of a full-function system include the following:

- When initially creating OS IV/F4 at an installation
- When adding and deleting major subsystems such as large numbers of communication lines and/or major functions such as APR function, AVR function, APG function, etc.

Generating a new I/O configuration
An abbreviated system generation can be performed to add/delete I/O devices and/or channels. Compared with generating a complete version of OS IV/F4, considerably less processing time is required for generating only a new I/O configuration.

For additional details on OS IV/F4 system generation, the reader should consult the **FACOM OS IV/F4 System Generation User's Guide.**

9.2 FLOW OF SYSTEM GENERATION

System generation comprises four stages: Stage 0, Stage 1, Stage 2, and testing the new system.

Stage 0
During Stage 0, the installation prepares to perform system generation.

If it has no established operating system, it uses the Fujitsu-supplied starter system with the library tapes and independent (stand-alone) utility programs:

- The starter system is restored from magnetic tape to several DASD volumes. Stand-alone utility programs are used for dumping/restoring reels to volumes.
- Controlled by the starter system, other libraries are copied from magnetic tape to DASD by the JSECOPY utility program.

If the installation already has an operating version of OS IV/F4, it need only catalog any new or updated library tapes during Stage 0.

Stage 1
This stage of system generation is executed as a single OS IV/F4 job, processing an input stream prepared by the installation's system staff. This stream begins with assembly of a group of system-generation macro instructions defining the new OS IV/F4 system, plus JCL for Stage 1.The OS IV/F4 Assembler checks these macro instructions for syntactical correctness and valid parameter ranges. If the Assembler detects one or more errors, it issues the usual diagnostic messages but no other outputs. After the system staff have corrected these macro instructions, they reattempt Stage 1 until it runs successfully.

Stage 2
The outputs from Stage 1 are (1) the assembly listing and (2) a stream of JCL statements and other control statements which become the input stream to Stage 2. Added at this time are any installation-supplied source libraries, SVC routines, additional LINKLIB members, etc. During Stage 2, several assemblies and link-edits are performed, which create the system nucleus and various other components of the OS IV/F4 control program. Other control statements entering Stage 2 request copying and merging of system libraries such as LINKLIB, SVCLIB, PARMLIB, PROCLIB, etc. Stage 2 directs these program libraries and source-statement libraries to their ultimate DASD locations within the generated

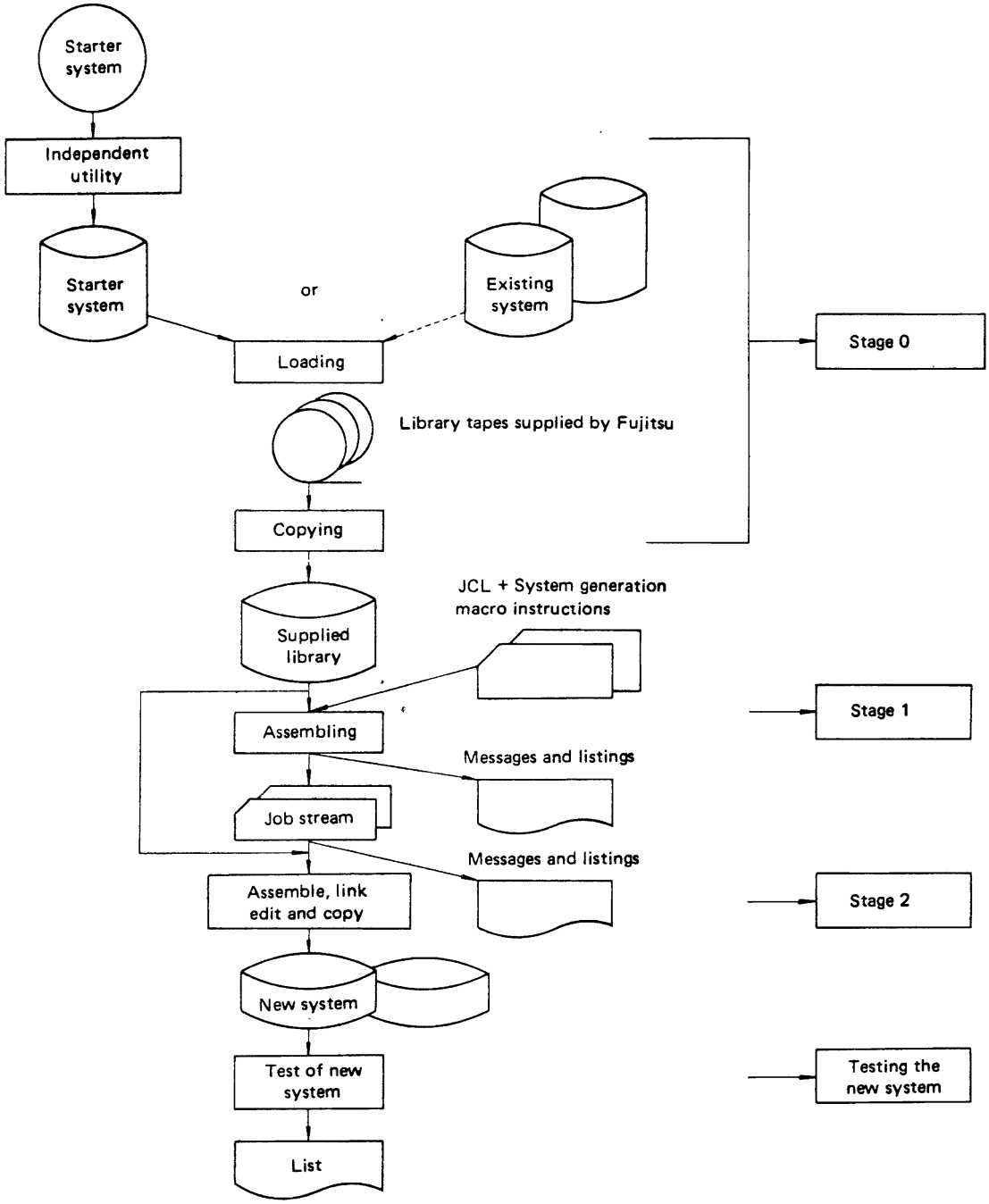


Fig. 9.1 System generation flow

system, as directed by the installation staff with their Stage 1 parameters.

Testing the new system

In order to thoroughly test a new OS IV/F4 system, Fujitsu supplies a group of jobs known as the installation verification procedure (IVP), supplied on the library reels together with the operating system itself. These jobs perform the following diagnostic and validation tests:

- Execute the assembler, linkage editor, and loader.
- Print out the system procedure library (SYS1.PROCLIB).
- Print out the system parameter library (SYS1.PARMLIB).
- Simulate an OS IV/F4 control-program failure, which tests recovery software and associated operating procedures.
- Print out lists of all I/O channels, control units, and devices.

9.3 RESOURCES REQUIRED FOR SYSTEM GENERATION

An installation can use either the starter system or its prior OS IV/F4 system to perform system generation. The minimum configuration for performing a system generation is as follows:

- M series CPU
- At least 304K bytes of main storage
- Standard console
- Card reader
- Line printer
- At least 4 DASDs
- At least 2 nine-track magnetic tape drives (density at least 800 bpi)

The result of system generation is the output from Stage 2, including the following three groups:

- Nucleus - stored in SYS1.NUCLEUS.
- System parameters.
- System data sets.

9.4 SYSTEM PARAMETERS

Parameters determining characteristics of the version of OS IV/F4 at a particular installation are called **system parameters**, stored in the **system parameter library** (SYS1.PARMLIB). Each installation creates/changes members of PARMLIB as follows:

- Parameters of system generation macro instruction generate some PARMLIB members.
- Subsequent additions, deletions, and changes, can be performed with the JSEUPDTE utility.

PARMLIB members are read by OS IV/F4 when the system is being reloaded (IPL). In addition, the console operator can override certain parameters at IPL time.

9.5 SYSTEM DATA SETS

Table 9.1 lists OS IV/F4 system data sets.

Table 9.1 System data sets

Data set name	Contents	System residence	Required	Creation	Cataloged	Secondary allocation allowed	Organization
CMDPROC	Command procedure library	Optional	Required in a TSS system	When it is to be used	Optional	Yes	PDS
PASSWORD	Password and corresponding data set name	Yes	Required for PASSWORD registration	Prior to using data set protection feature	Optional	No	Sequential
SYSCTLG	Catalog of data sets	Yes	Yes	At SYSGEN time	No	Yes	Sequential
SYS1.BRODCAST	Notices (messages to all users) Mail (messages to specific users)	Optional	Required in a TSS system	After SYSGEN, first use of SEND initializes it	Yes	No	Direct
SYS1.CMDLIB	Command processor library	Optional	Required in a TSS system	At SYSGEN time	Yes	Yes	PDS
SYS1.DCMLIB	Program function key (PFK) definitions for display consoles	Optional	Required if transient module display function is specified in SCHEDULE or SECONSLE	At SYSGEN time	Yes	No	PDS

Table 9.1 Continued

Data set name	Contents	System residence	Required	Creation	Cataloged	Secondary allocation allowed	Organization
SYS1.DSSVM	Command language modules used by the dynamic support system (DSS)	Optional	Yes	At SYSGEN time	Yes	No	Sequential
SYS1.DUMP	Core image dumps recorded by ABEND or ABTERM	Optional (either DASD or magnetic tape)	No	At SYSGEN time if DASD, or IPL time if megnetic tape	Yes	No	Sequential
SYS1.HELP	TSS HELP information	Optional	Required if TSS HELP is to be used	At SYSGEN time	Yes	Yes	PDS
SYS1.IMAGELIB	UCS and FCB image modules	Optional	Required for line printers with USC and paper tape units	At SYSGEN time	Yes	No	PDS
SYS1.LINKLIB	Load modules which are not in the link pack area	Optional	Yes	At SYSGEN time, members can be added, additional data sets can be concatenated	Yes	Yes	PDS
SYS1.LOGREC	Statistical data of machine errors	Yes	Yes	At SYSGEN time; size may be changed after SYSGEN	No	No	Sequential
SYS1.LPALIB	Load modules in link pack area	Optional	Yes	At SYSGEN time	Yes	Yes	PDS
SYS MACLIB	Definition of system macro instruction	Optional	Yes	At SYSGEN time	Yes	Yes	PDS
SYS1.MANX SYS1.MANY	Data collected by SMF	Optional	Yes	Allocate before IPL	Optional	No	Sequential
SYS1.NUCLEUS	Resident portion (nucleus) of control program modules	Yes	Yes	At SYSGEN time	Yes	No	PDS
SYS1.PARMLIB	System parameter library	Optional	Yes	At SYSGEN time Additions and changes allowed	Yes	No	PDS
SYS1.PROCLIB	System procedure library	Optional	Yes	At SYSGEN time Members may be added	Yes	Yes	PDS
SYS1.SAMPLIB	Independent utilities, IPL text and SMF exit routines	Optional	Yes	At SYSGEN time	Optional	Yes	PDS
SYS1 SVCLIB	Nonresident SVC routines, SER, MCH modules, etc.	Yes	Yes	Allocate before SYSGEN	Yes	Yes	PDS

Table 9.1 Continued

Data set name	Contents	System residence	Required	Creation	Cataloged	Secondary allocation allowed	Organization
SYS1.SYSJOBQUE	System job queue	Optional	Yes	Allocate just before IPL	Required if not on system residence volume	No	Sequential
SYS1.SYSPOOL	Auxiliary data sets for SYSIN, SYSOUT, SYSLOG	Optional	Yes	Allocate just before IPL	No	No	Sequential
SYS1.UADS	RES and TSS user attributes	Optional	Required for TSS	At SYSGEN time Members may be added	Yes	Yes	PDS
SYS1.VTAMLIB	Load module of network solicitor	Optional	Required for VTAM	At SYSGEN time	Yes	Yes	PDS
SYS1.VTAMLST	Network definition decks and VTAM parameters	Optional	Required for VTAM	At SYSGEN time	Yes	Yes	PDS
SYS1.VTAMOBJ	Load module of network control program (NCP)	Optional	Required for VTAM	Allocate before VTAM generation time	Yes	Yes	PDS

9.6 SYSTEM GENERATION MACRO INSTRUCTIONS

These fall into the following four categories summarized in Tables 9.2, 9.3, 9.4 and 9.5:

- Hardware configuration.
- Control program.
- Installation-supplied routines.
- System generation.

Table 9.2 Macro instructions defining the hardware configuration

Macro instruction	Parameters
CENPROCS	• CPU model
CHANNEL	• Address and type (multiplexor, selector, etc.)
IODEVICE	• Address of first (or only) device of a group • Number of consecutive-address devices in this group • Auxiliary functions • Model number • Any alternate channel • Adapters (only for F2803 or F2805)
UCS	• Identifier for default character set • Default value for UCS parameter on DD statement
UNITNAME	• Name for group of I/Odevices • Their device addresses

Table 9.3 Macro instructions defining the control program

Macro instruction	Parameters
CKPTREST	• Normal step-termination codes for which automatic restart is to be excluded • ABEND termination codes for which step restart is automatically attempted
CTRLPROG	• APG – Range of dispatching priorities – APG interval: maximum, minimum & increment values – Period between automatic changes of APG interval – Ratio of wait state to time slice, per task • Support of ANSI/JIS tape formats – Accept/create ANS tapes – Accept/create JIS tapes – No ANS or JIS tapes • Size of the common system area (in each address space) • Maximum number of simultaneous I/O requests
CTRLPROG	• Maximum number of user address spaces • Fixed BLDL option • At IPL, option to display offline devices on a console • Limited/full number of error retries for certain devices • Reliability Data Extract (RDE) option • Page frames zeroed prior to reallocation • Number of segments for system tables • Maximum number of frames for a real-storage region • Greenwhich Mean Time or local time • Date format for messages: YYMMDD or YYDDD
DATAMGT	• VTAM option • VSAM option
EDITOR	• Load-module block size
JES	• Size of a JES buffer • Number of JES buffers • Names of demand-output classes • Card readers for demand output • Line printers for demand output • Maximum number of SYSOUT data set records (OUTLIM default value) • Priority aging – Upper limit for selection priority – Lower limit for selection priority – Time interval between aging increments • SYSOUT default device(s) • JES readers – Maximum number – Number of records per JAM input request – Maximum storage for all SYSIN buffers • Threshold value for SYS1.SYSPOOL capacity • Spool volumes identifier • Maximum number of SWADS records per JCL statement • Maximum number of WTL macro instructions and LOG commands • JES writers – Maximum number – Maximum storage needed for installation-supplied writer and/or job separator routines – Number of records per JAM output request Maximum storage for all SYSOUT buffers
LOADER	• DD name for load module library • DD name for input source • DD name for diagnostic and map outputs • Loader options

Macro instruction	Parameters
	– Diagnostic messages – External references map – Abort execution if major errors are encountered – Default load module libraries – Automatic usage of LPA
PAGE	• Pageable LPA • Size of page data set • Primary/secondary volumes in page data set • Device address and volume serial number of page data set
SCHEDULER	• Main console and its alternate console • Size of area (for display console) • Size of system broadcast data set (SYS1.BRODCAST) • Disposition of error messages for volumes: SMF data sets or console • Error Statistics by Volume (ESV) and Error Volume Analyses (EVA) limits • Hardcopy log – Device address – Optional SYSLOG capture of hardcopy messages – Destination code – Types of messages • SYSJOBQE blocking factor • Number of SYSJOBQE records initially allocated to each job stream • Default destination for WTO and WTOR macro instructions • Remote entry services • Number of buffers for WTO commands • Number of outstanding WTORs requiring responses • Destination codes for the main console • Number of PFKs for a display console • Output class for the system log • Number of WTL buffers • WTL output class • Automatic volume recognition • Default AVR density for 7-track magnetic tapes
SECONSLE	• Address of one auxiliary console • Address of its alternate console • Number of PFKs, if a display console
SECONSLE	• Destination codes associated with this console • If output display console: – Display status – Operator messages • Authorization level

Table 9.4 Definitions of user-generated routines

System generation macro instruction	Content of specification
LINKLIB	• member names of user-generated routines that should be added to the new system's SYS1.LINKLIB • name of library from which added members are retrieved during system gen. • names of members that are to be stored in the KAAFIX00 list in order to fix those added members into main memory (FLPA)
LPALIB	• member names of user-generated routines to be added to SYS1.LPALIB • name of library from which added members are retrieved during system gen. • names of members, from among those added members, which are to be stored in the KAAFIX00 list
RESMODS	• member names of user-generated routines to be included in SYS1.NUCLEUS • name of library from which added members are retrieved during system gen.
SVCTABLE	• SVC numbers of user-generated SVC routines added to the system • SVC type (1, 2, 3, or 4) • type of lock (see FACOM OS IV/F4 Supervisor Functions and Facilities • SVC authorization: only by specified users, or by any user

Table 9.5 Definition of system generation

Macro instruction	Content of specification
DATASET	• name of system data set • amount of space • volume serial number and device type
GENERATE	• type of system generation • temporary data set index names used during system generation • step names for stage-2 job • output class for system-generation jobs • serial number of the new system resident volume

PART 3
PROCESSING PROGRAMS

CHAPTER 1
COBOL

1.1 OVERVIEW

COBOL is a high-level language especially suitable for complex file-handling and data-processing problems. This chapter provides a brief description of the OS IV/F4 COBOL compiler. For more detailed discussions of COBOL, the user is referred to the following publications:

- **FACOM OS IV JIS COBOL Reference Manual**
- **FACOM OS IV/F4 JIS COBOL User's Guide.**

The OS IV/F4 COBOL compiler conforms with the Japan Industrial Standard (JIS) COBOL language (1972) and also incorporates all major functions of the American National Standard (ANS) COBOL (1974).

JIS COBOL includes the nucleus (basic elements such as data-item operations, comparisons, and transmissions), table handling (OCCURS clause, SET statement, SEARCH statement, subscript comparison), sequential access, random access, sort, report generation, segmentation, and Library. The OS IV/F4 JIS COBOL provides several extensions of these functions as well as all mandatory elements.

1.2 OUTLINE OF FUNCTIONS

The following describes the major and extended functions of OS IV/F4 JIS COBOL.

1.2.1 Reentrant Programs

Object programs generated by the OS IV/F4 COBOL compiler and the system library modules are always reenterable. When link-editing a load module, the user should specify the RENT parameter to the linkage editor.

1.2.2 Program Linkages

To pass control and reference common data fields among several programs, the programmer issues CALL, CANCEL, ENTRY, EXIT PROGRAM, GOBACK, and STOP RUN statements. A CALL statement passes control (and various parameters) to a subprogram. The CANCEL statement resets and releases areas used for a subprogram to which access has been already made. The ENTRY statement indicates an entry point in a subprogram.

Object modules from different languages can be linked directly, such as programs written in COBOL and FORTRAN, if the user converts data formats and/or array elements in each program. Where data formats are incompatible, it is still possible to link COBOL and FORTRAN programs via an intermediate PL/I program: COBOL→PL/I→FORTRAN (or vice versa). Data arrays or data structures can be easily converted by specification through PL/I. The PL/I linkage approach also permits user handling of various types of program interruptions, as discussed in Section 3.2.2.

1.2.3 Program Structures

The COBOL compiler can generate the following program structures: simple, overlay, dynamic link, and dynamic program structures.

Various linkages such as attaching subtasks can be performed with the assembler and PL/I languages for controlling a dynamic program structure. However, COBOL supports only one type of linkage: passing control to specified subprograms. In a simple or overlay structure, all required modules must be link-edited together before execution. In a dynamic link structure, load modules can be brought into memory as required.

If a dynamic link structure is used for a COBOL program, common subroutines need not be link-edited into each main program; they can be maintained as separate load modules in a private library. These subroutines can be easily corrected, facilitating their maintenance.

1.2.4 Optimization

At the user's option, the OS IV/F4 COBOL compiler will optimize execution speeds and/or program lengths of designated object modules. Items (1) and (2) below are unconditional optimizations, items (3) and (4) must be explicitly requested.

(1) Constants for condition names, literals, etc. are uniquely compiled for each object program. Only initial values in the data division need sometimes be repeated.

(2) During execution, COBOL programs do not unnecessarily recompute addresses for subscripted data items, as shown in part (a) of Fig. 1.1.

(3) The compiler allocates base registers semipermanently to frequently-referenced data record areas.

(4) By dividing the procedure division of each program into control sections of at most 4096 bytes (**procedure blocks**), the COBOL compiler translates GO TO statements (and other branching statements) into RX-type branch instructions, which are only four bytes long and hence highly efficient. Each branch instruction uses a base register pointing to the start of its procedure block (PB), plus a displacement pointing within the PB. This optimization considerably decreases the number of address constants and makes object programs both smaller and faster.

```
MOVE ABC (J, K) TO XYZ ..... 1
MOVE EFG TO ABC (J, K) ..... 2

The address calculated for ABC(J,K)
in statement 1 is saved and used for statement 2.
```

(a) Optimization example

```
FD AFILE . . . . . .
01 AREC.
   02 A1    PIC X(5).
   02 A2    PIC X(20).
      .
      .
      .
FD BFILE . . . . . .
01 BREC.
   02 B1    PIC X(10).
   02 B2    PIC X(20).
      .
      .
      .
   MOVE A2 TO B2.

   Without optimization        With optimization

   L    α, = A (AREC)
   L    β, = A (BREC)          MVC 10(20,8),5(7)
   MVC  10(20, β), 5 (α)

The registers α and β are      The registers 8 and 7
indefinite. At each MOVE       are used.
statement, two Load instructions
are generated.
```

(b) Optimization example

Fig. 1.1 Examples of optimization

1.2.5 Conversational Processing

Since the COBOL compiler can operate under TSS (time sharing system) control, a user can generate, debug, and execute a program from a terminal.

The TERM or TEST option specifies conversational processing to the COBOL compiler at the time of translation. The TERM option directs ordinary system output to the SYSTERM data set: listings, error messages, etc. The TEST option permits debugging in a conversational style. Data sets and programs generated by COBOL under batch or TSS control are compatible.

OS IV/F4 provides the following conversational aids for the COBOL compiler (see Part 4 of this manual for TSS details):

- COBOL prompter
  When invoked by a command from the terminal, the prompter assists the user in supplying all necessary program elements, DD statements, etc. to execute successfully.
- COBOL interactive debug
  When the programmer specifies the TEST option for execution, he can debug his program in a conversational style; he can specify certain line numbers and data names in his sources program which OS IV/F4 will display as encountered, thereby tracing his program flow.

1.2.6 Communications Interface

The OS IV/F4 COBOL optionally provides a communications interface between a program and various communications facilities. Programs can transmit information between OS IV/F4 and remote terminals independent of the type(s) of terminals.

1.2.7 Extended Source Program Library

An **extended source program library** (ESPL) is defined by a combination of BASIS, and DELETE statements. This facility makes it simple to compile a modified version of a source program from a user's private library. Without this feature, recompilation would require two steps: (a) correcting the source program and (b) compiling the updated program. With ESPL, only step (b) is necessary; the contents of the source program library are not changed, and debugging is facilitated, as shown in Fig. 1.2.

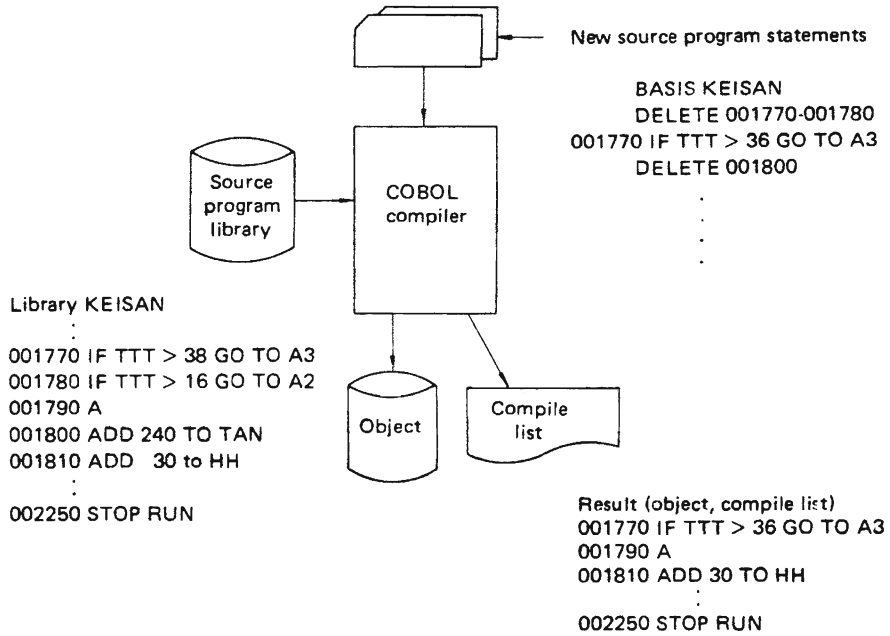


Fig. 1.2 Example of compiling an updated source program using ESPL

1.2.8 Bit Processing

Bit processing permits the user to address the smallest M series unit of information. In a loaded OS IV/F4 environment such as a heavy-traffic online system, bit processing can reduce virtual-storage and DASD needs. Also, bit processing facilitates flexible linkages between different languages (especially the PL/I and assembler languages) and various application programs. Fig. 1.3 shows an example of defining and using bit processing.

```
01  A1          PIC 1(2) BIT.
    02  A2      PIC 1(6) BIT.
    02  B.
        03  B1  PIC 1(2) BIT.
        03  B2  PIC 1    BIT.
            .
            .
            .

    COMPUTE A1 = B1 AND B2
```

Fig. 1.3 Example of bit processing

1.2.9 Character-String Processing

MOVE statements can be used to concatenate/separate character strings; however, the programmer must write several statements if he uses only the standard COBOL facilities. The OS IV/F4 character-string facility furnishes the following statements:

- STRING statement
  This statement concatenates several data items into one item.
- UNSTRING statement
  This statement separates one data item into several subfields.
- INSPECT statement
  This statement replaces or tallies specified characteristics of data items.

1.2.10 File Organizations

An OS IV/F4 COBOL program can process the following file organizations:

- Sequential (SAM).
- Relative (see below).
- Direct (DAM).
- Virtual sequential (VSAM).

A **relative file** is a direct organization specially defined for COBOL; relative files must be assigned to direct access storage devices. With this file organization, the position of each record is determined relative to the first record of the file. A relative file must be created sequentially but may be retrieved or updated sequentially or randomly. To access a relative file, the programmer must specify RELATIVE in the ORGANIZATION clause of his source program.

A member of a partitioned data set (PDS) can be processed as a sequential file by furnishing the member name on the corresponding DD statement. For instance, when processing member B of a PDS named "A", the following DD statement is appropriate:

```
//INDATA DD DSN=A(B), . . . . .
```

1.2.11 Debugging Facility

The OS IV/F4 COBOL debugging facility provides statements which may appear anywhere in the source program and specify compile-time debugging.

Source program debugging facilities

- TRACE
  When a TRACE statement is encountered during execution, each time a section or paragraph is entered the corresponding line number is displayed. The user can specify the start (READY TRACE) and end (RESET TRACE) of tracing.
- EXHIBIT
  Execution of an EXHIBIT statement displays a specified identifier and its value. The user may specify display of an identifier only if its value has changed since the previous time the EXHIBIT statement was executed.
- ON
  An ON statement specifies when corresponding statements are to be executed. Counters are provided within each ON statement during compilation. Every time an ON statement is reached, the counter is incremented by one and the count condition is tested. If the condition is satisfied the corresponding statement is executed. The following is an example of using an ON statement:

```
77 TWO PIC 9(1) VALUE 2
.
.
.
ON TWO MOVE A TO B ELSE
       MOVE A TO C
.
.
.
```

- Debugging packet
  Statements for debugging a given section or paragraph in the program may be placed directly after the body of the procedure division. These **debug packets** will be compiled with the source program and will be executed at object time. These packets can include the statements for debugging stated above (TRACE, EXHIBIT, ON). The packet for debugging is arranged as follows:

```
PROCEDURE DIVISION
.
.
.
P2
.
.
.
P3 SECTION
.
.
.
DEBUG P2
D13   }
.     }
.     } Packet for debugging
.     }
D14   }
.
.
.
```

Compiler option statements

- STATE option
  This option indicates the program status if an abnormal program termination occurs. It also displays the line number and verb being executed at the time of abnormal termination.
- FLOW option
  The FLOW option traces the flow of up to 99 procedures entered just prior to an abnormal termination.
- SYMDMP option
  This option dumps the values of selected data names at any time, not necessarily for an abnormal termination. The user can specify the location and frequency of these dynamic dumps.

1.2.12 Other Features

OS IV F/4 COBOL also provides the sort/merge, the Report writer feature, and the segmentation feature.

Sort/merge

By using the sort verb, sorting and merging of files can be requested within COBOL source programs. Procedures can be optionally included for special handling of files before and after they have been sorted. When the sort program is requested, the program can set special registers to control its options, as follows:

- SORT-FILE-SIZE register
  Specifies the data item containing the number of input records.
- SORT-CORE-SIZE register
  Data item specifying the size of the memory area to be used by sort/merge.
- SORT-MODE-SIZE register
  Data item specifying the average record length for variable-length records.
- SORT-RETURN register
  Data item to receive the sort/merge return code.
- SORT-MESSAGE register
  Specifies the DD name for sort/merge messages.

Report generation

This feature facilitates report generation by specifying the format of each report in the data division

instead of requiring detailed procedure division coding.

Each report is divided into several report groups; each group is further divided into a series of items. Whenever a report group is cited by the program, all of its items are cited by implication.

Segmentation

The procedure division of a COBOL source program is usually written as a series of functionally distinct sections. When the programmer requests the **segmentation** feature, the compiler segments the entire procedure division. The compiler must distinguish whether each section belongs to the fixed portion or to an independent segment.

The **fixed portion** is defined as that part of a COBOL load module which is always in virtual memory; it comprises **permanent segments** and **overlayable fixed segments.** A permanent segment is a segment in the fixed portion that cannot be ovelaid. An overlayable fixed segment is treated logically as if it is always located in virtual memory, but it may be overlaid by another segment. An **independent segment** can overlay—and be overlaid by—another independent segment or an overlayable fixed segment.

In the SECTION clause, the programmer assigns to the fixed portion and independent portion priority numbers from 0 to 99. Priority numbers 0 to 49 are for the fixed portion, 50 to 99 for the independent segments.

To decrease the number of permanent segments, the programmer can specify priority numbers between 1 and 49 in his SEGMENT-LIMIT clause. Only sections whose priority numbers are smaller than the SEGMENT-LIMIT are considered permanent. Sections whose priority-numbers are larger than the SEGMENT-LIMIT but less than 49 are considered to be overlayable fixed segments.

In order to execute a segmented program, the programmer must specify OVLY in the parameter field of the linkage editor EXEC statement.

1.3 REQUIRED CONFIGURATION

The configuration of I/O functions (and associated DD names) required for the COBOL compiler is shown in Fig. 1.4.

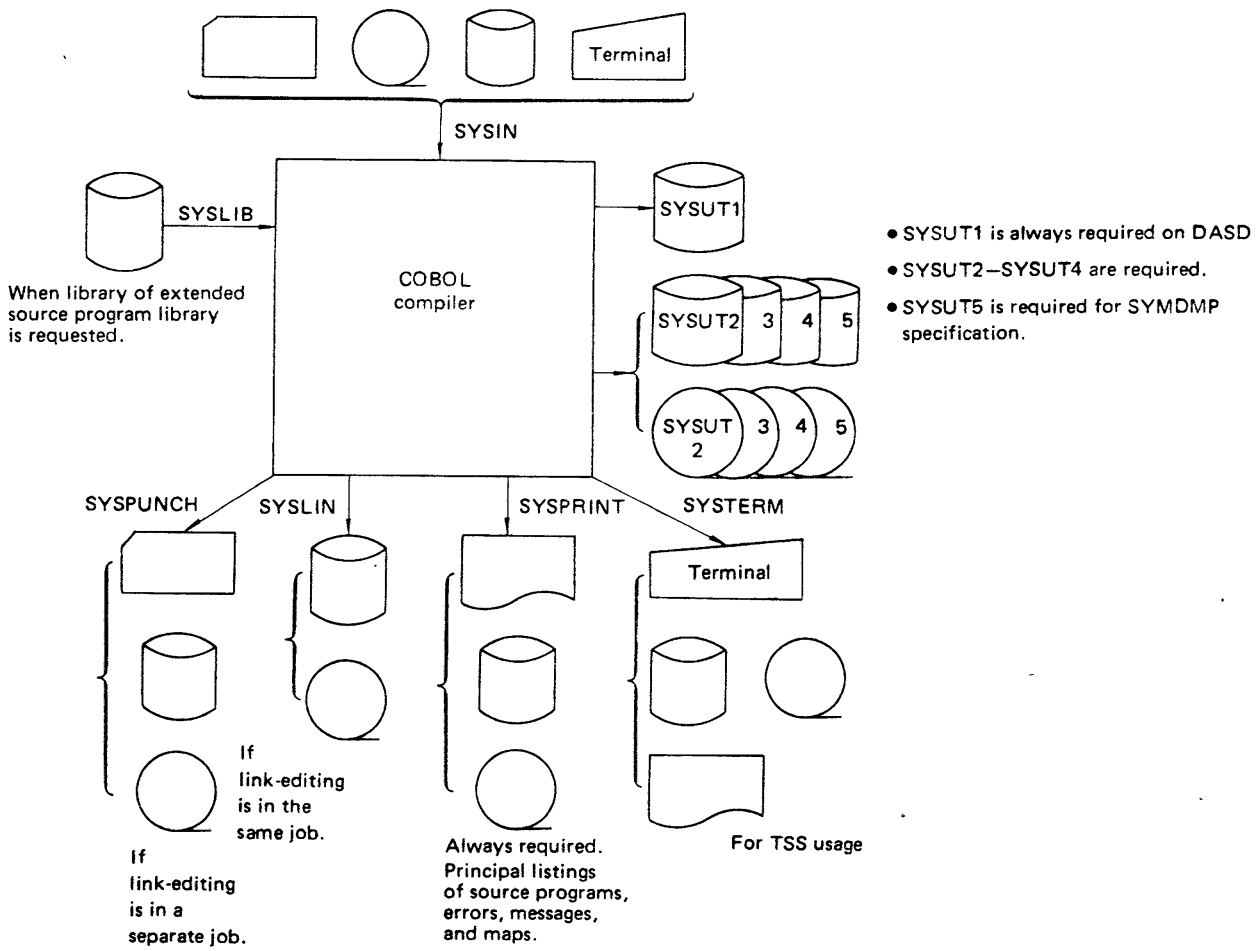


Fig. 1.4 COBOL compiler unit configuration diagram

CHAPTER 2
FORTRAN

2.1 OVERVIEW

The FORTRAN language is especially appropriate for mathemetical computations and other manipulations of numerical data. OS IV/F4 provides two FORTRAN compilers:

- OS IV/F4 FORTRAN IV GE compiler.
- OS IV/F4 FORTRAN IV HE compiler.

This chapter provides a highlight description of the GE and HE compilers. Additional information concerning FORTRAN can be found in the following publications:

- **FACOM OS IV FORTRAN Reference Manual.**
- **FACOM OS IV/F4 FORTRAN HE User's Guide.**
- **FACOM OS IV/F4 FORTRAN GE User's Guide.**

The GE and HE levels of the FORTRAN language offer many extensions to standard FORTRAN. The GE language is a subset of the HE language with the exception of the following two features:

- Debug packet.
- Free-form source statements.

Object modules generated by the GE and HE compilers are compatible and can be linked to each other. A single FORTRAN subroutine library is provided for both compilers. However, an installation not needing the HE compiler may exclude corresponding specialized subroutines from this library during system generation.

2.1.1 GE Compiler

The GE compiler is especially convenient for developing FORTRAN programs; it has the following highlights:

- Minimal compilation times.
- Comprehensive debugging facilities.
- Simple operation from a terminal.

The GE compiler is itself a reentrant program. Therefore, when a large number of users use the compiler simultaneously, only one copy of the compiler is loaded, saving real-storage page frames.

By specifying the GO option, the loader retrieves one or more object modules into virtual storage, and the resulting object program is immediately executed. When compile and load steps bypass linkage editing in this way, execution efficiency is significantly raised.

2.1.2 HE Compiler

The HE compiler is appropriate for compiling efficient FORTRAN object programs; it has the following characteristics:

- Object code optimized for both size and execution speed.
- Wide range of source-program documentation options, such as structured listings and cross-reference listings.

2.2 HIGHLIGHTS

2.2.1 Reentrant Programs

A user can compile a reentrant FORTRAN program by furnishing the following specifications:

- For either the GE or HE compiler, the RENT option requests a reentrant object module.
- In addition, the programmer must specify RENT for link-editing or loading to create an executable reentrant program.

2.2.2 Linkages with Object Modules from Other Languages

When linking programs written in different languages such as FORTRAN and COBOL, programmers need take no special precautions if data formats, array elements, etc. are compatible at the

machine-language level. In some cases, the programmer can link incompatible subprograms by using a locally-written PL/I interface program, as shown in Fig. 2.1. In this method, array elements and data structures can be converted simply by specification in a PL/I subprogram. Also, if linkages are made via PL/I, the programmer can more readily handle his own interruptions and other exceptional conditions.

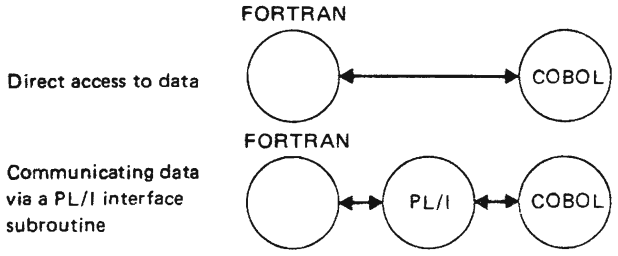


Fig. 2.1 Example of communication between different languages

2.2.3 Program Structures

Programs with simple, overlay, and dynamic link structures can be created in the FORTRAN language. Source-program coding is independent of structure; the user specifies the desired structure during link-editing.

In a simple structure or overlay structure, all required modules must be link-edited together. For a dynamic link structure, modules can be loaded dynamically during execution; frequently-accessed subroutines can be maintained as separate load modules, as described in Section 8.6.4 of Part 2 of this publication.

2.2.4 Optimization Procedures

The HE compiler optimizes execution speed of user programs as follows:

- Deletion of logically-unnecessary assignment statements, elimination of common subexpressions, and relocation of loop-invariant computations, as shown in Fig. 2.2(a).
- Replacing multiplications of loop variables by additions (when possible) and replacing small integral powers of a variable by repeated inline multiplications, as shown in Fig. 2.2(b).
- Optimization of sequential I/O as shown in Fig. 2.2(c).
- In-line code is compiled for statement functions in the same way as for built-in functions. This code is then optimized as shown in Fig. 2.2(d).
- Initial linkages to/from subprograms are optimized for execution speed rather than program size. This optimization is especially efficient when subprograms contain many statements and are within DO-loops.

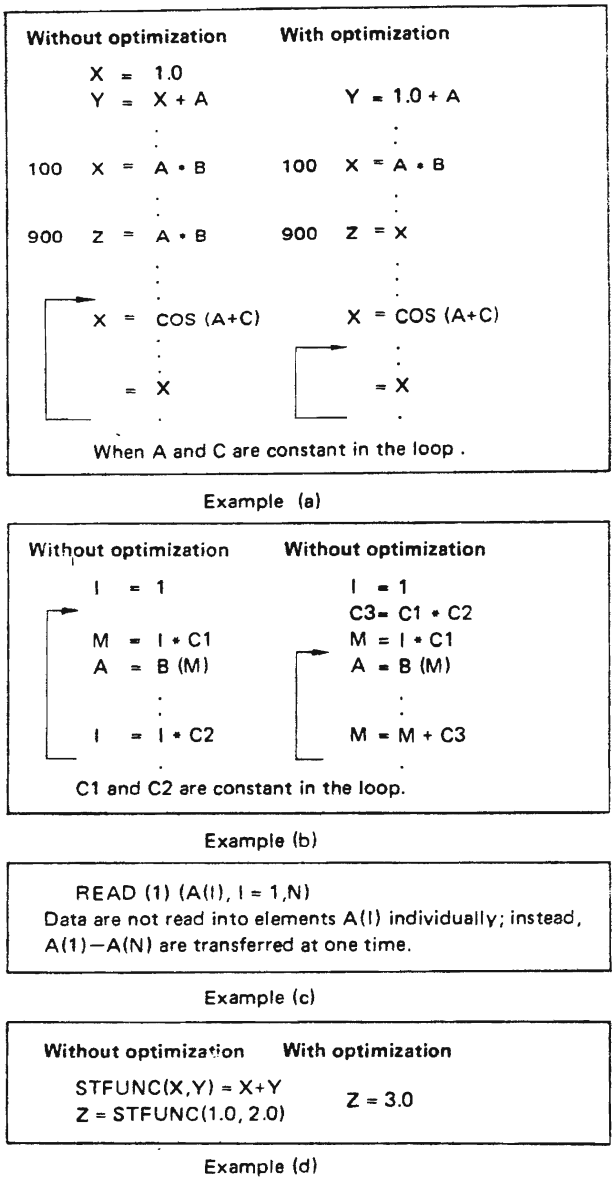


Fig. 2.2 Examples of optimization

- In general, each use of a mathematical function requires a distinct function call. The trigonometric functions are often used with one another; therefore, the compiled code reduces by saving both function values obtained by one call to a library routine. For example, the consecutive statements

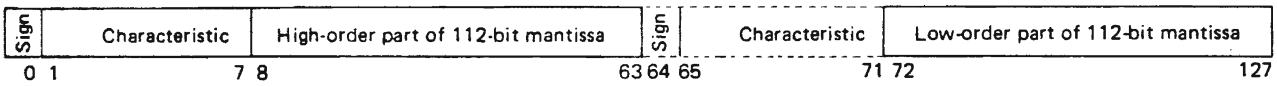
  Y = SIN (X)
  Z = COS (X)

  require only one call to the sine/cosine routine.
- By the DPROF option, a load module can be divided into **basic blocks** (statement sequences without branches). The programmer can request that entries to these blocks be counted during execution to determine which blocks are most important for job-step performance.

Table 2.1 Precision comparison table

Precision	Data type					Mantissa precision		Characteristic precision	
	Byte	Bit	Bit structure			Hexa-decimal	Decimal (approx)	Hexa-decimal	Decimal (approx)
			Mantissa sign	Charac-teristic	Mantissa				
Single precision	4 (1 word)	32	1 bit	7 bits	24 bits	6 digits	7.2 digits	16^{-65}	10^{-78}
Double precision	8 (2 words)	64	1	7	56	14	16.8	. . .	. . .
Extended precision	16 (4 words)	128	1	7	112	28	33.6	16^{63}	10^{75}

Extended floating point format



2.2.5 Conversational Processing

Since the FORTRAN compilers can operate under TSS control, the user can generate a program, debug it, and execute it from a terminal. Data sets and programs generated in batch and TSS modes are identical and may be freely intermixed. For details on TSS facilities, the reader should review Part 4 of this publication.

To perform conversational processing in FORTRAN, the following features are particularly convenient:

- FORTRAN syntax checker
  This separate TSS processor checks for syntactical errors in a FORTRAN source program. The checker operates under the TSS editor; whenever the terminal user enters a syntactically-erroneous line, he can correct it at once.
- FORTRAN prompter
  This separate TSS processor is activated by a command from a terminal. The prompter provides a simple user interface for invoking either FORTRAN compiler and for submitting execution-time JCL information.
- FORTRAN interactive debug packet
  If the programmer specifies the TEST option while compiling his FORTRAN program, his source-program line numbers and variable names are displayed selectively during execution. By controlling the flow of his program, the programmer can test his program conversationally. The debug packet can only be used with the GE compiler.
- Free-form source statements
  With the GE compiler, a programmer can enter FORTRAN programs without regard for "card columns"; he uses the tab key on his terminal to space from one field of a statement to the next field.

2.2.6 Extended Precision For Real and Complex Arithmetic

In some scientific and technical computations, quadruple precision arithmetic is necessary. Both the HE compiler and GE compiler can compile quadruple-precision data and appropriate M series instructions.

Since the characteristic of each M series floating point number has 7 bits, it can range from 0 to 127. To accommodate both positive and negative exponents, 64 is added to the actual exponent to form the **characteristic** for each floating-point number. Therefore, the magnitude of a floating point number ranges from 16^{-65} to 16^{63} in hexadecimal (approximately 10^{-78} to 10^{75}in decimal).

An **extended floating point number** comprises two long floating point numbers. The sign and characteristic of the low order number are ignored and assume, respectively, the sign of the high-order number and characteristic of the high-order number minus 14. The mantissa of the low-order number contains the 14 (unnormalized) low-order digits of the 28-digit fraction.

2.2.7 Automatic Precision Increase

By specifying the AUTODBL option for the HE compiler, the programmer requests **automatic precision increase** (API): constants, variables, functions, and array data (real, double-precision real, complex, and double-precision complex) are converted to the next higher level of precision. Storage areas defined by DIMENSION, EQUIVALENCE and COMMON statements are appropriately increased. In the GE compiler, precision can be increased by specifying the DOUBLE option or GUARD option. However, in this case, redefining storage areas with EQUIVALENCE statements, etc. is not corrected automatically. See Table 2.1 for precision comparisons.

2.2.8 Asynchronous Input/Output Statements

By issuing asynchronous READ/WRITE statements, program execution can be overlapped with I/O processing. When the programmer uses asynchronous READ/WRITE statements, his WAIT statements synchronize I/O with computation.

Fig. 2.3 shows an example of asynchronous READ/WRITE and WAIT statements. This program reads data into array A. It processes the data and calculates new values for array B, from which a record is written. After the first asynchronous READ statement, input and output of data are performed concurrent with computation. The values of the COND parameter in the WAIT statement are as follows: "1" for a normal end, "2" for an unrecoverable I/O error, and "3" for an end-of-file condition.

```
      DIMENSION A(200), B(200), C(200)
      INT = 0
 5    READ (7, ID=1) A
      IF (INT. EQ. 0) GO TO 25
15    WRITE (9, ID=2) B
25    WAIT (7, ID=1, COND=K)
      GO TO (40,77, 88), K
35    INT = 1
      GO TO 45
         .
         .
40    IF (INT. EQ. 0) GO TO 35
      WAIT  (9, ID=2, COND=L)
      IF (L. GT. 1) GO TO 88
45    DO 55 I = 1, 200
      B (I) = A(I) + C(I)
55    C (I) = B(I)
      GO TO 5
         .
         .
         .
77       .
         .
         .
         .
88       .
         .
         .
99    STOP
      END
```

Fig. 2.3 Example of asynchronous input/output statements

2.2.9 Data Set Organizations

FORTRAN programs can access data sets with:
- Sequential organization, or
- Direct organization (without keys).

In addition, the programmer can access a member of a partitioned data set (PDS) as a sequential data set. For instance, member B of PDS A is specified by the following DD statement:

```
//INDATA  DD  DSN=A(B), . . . . .
```

2.2.10 Debugging Aids

FORTRAN debugging aids help the user to locate errors in his source programs:
- Tracing flow within programs.
- Tracing flow between programs.
- Displaying values of variables and arrays.
- Checking validity of subscripts.

To utilize these aids, the following statements are provided:
- DEBUG statement.
- AT statement.
- TRACE ON statement.

In addition, the following compiler options can be specified:
- GOSTMT/NOGOSMT.
- MAP/NOMAP.
- XREF/NOXREF (only in the HE compiler).
- DEBUG/NODEBUG (only in the GE compiler).
- ASTER/NOASTER.

2.2.11 Miscellaneous Features

- ENCODE/DECODE statements
  These statements transfer information from one area to another in the user's address space, performing data conversions specified by FORMAT statements just as for READ/WRITE statements.
- Implied DO loops in DATA Statements
- Assignment statements for character strings
  These are coded similar to arithmetic assignment statements, with the character data preceded by "nH" or enclosed in apostrophes.
  A = 4HABCD
  A = 'ABCD'
- Asterisk lines
  When the programmer puts an asterisk into the first column of a FORTRAN statement, he can specify at compilation time whether this statement is to be treated as a comment or as an executable statement.
- External functions and subroutines such as DATE, TIME, EXP2, and ALOG2.

2.3 REQUIRED UNIT CONFIGURATION

The configuration of I/O functions (and associated DD names) required by the FORTRAN IV GE and HE compilers is shown in Fig. 2.4. The optional configuration required by the FORTRAN IV HE compiler is enclosed in broken lines.

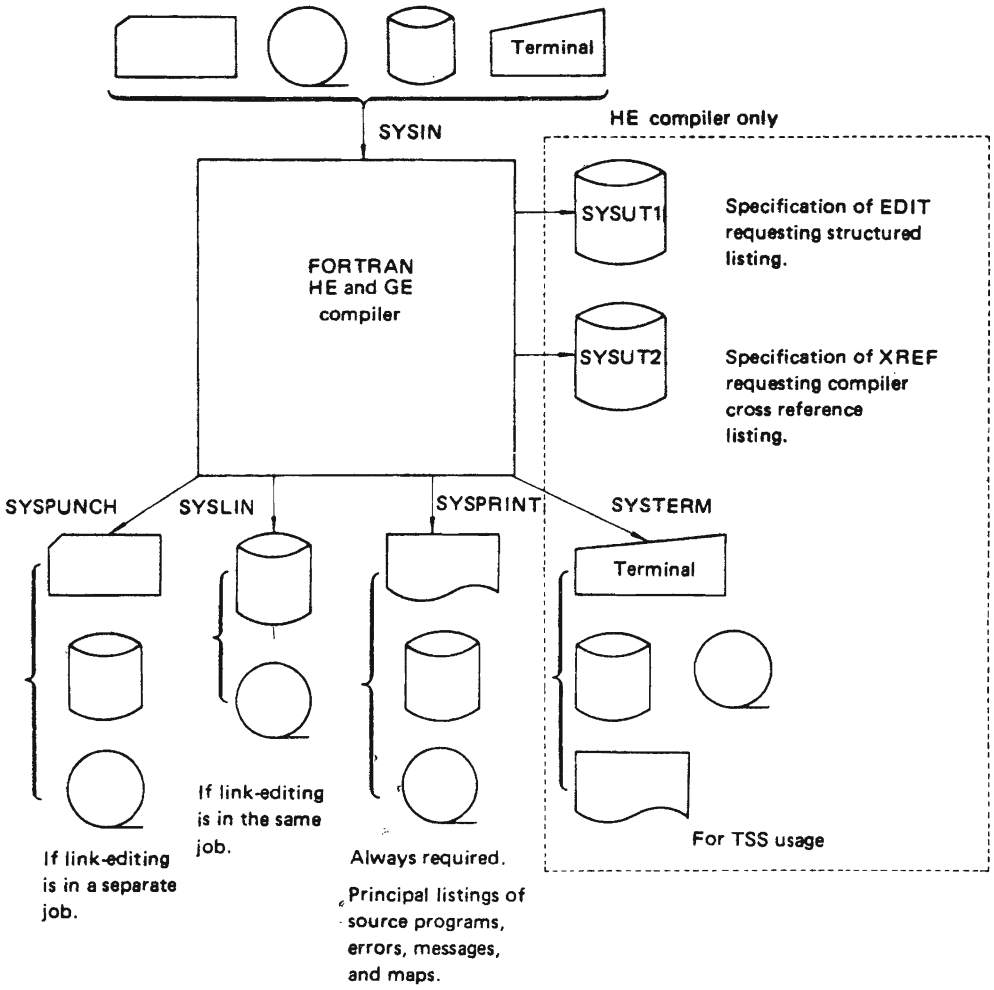


Fig. 2.4 FORTRAN HE and GE compiler unit configuration diagram

CHAPTER 3
PL/I

3.1 OVERVIEW

PL/I is a multipurpose programming language designed for both business and scientific applications. This section briefly describes the OS IV/F4 PL/I compiler. For further information on the PL/I language and how it is used with OS IV/F4, the reader is referred to the following publications:

- **FACOM OS IV/F4 PL/I Reference Manual.**
- **FACOM OS IV/F4 PL/I User's Guide.**

3.1.1 PL/I Subroutine Libraries

All PL/I programs use two libraries of load-module subroutines: (a) a **resident library** containing subroutines link-edited or loaded with user object modules, and (b) a **transient library** containing subroutines loaded as required during execution. This pair of libraries provides more efficient use of main storage than would a single resident library.

3.2 HIGHLIGHTS

The following sections describe basic features and selected unusual features of the OS IV/F4 PL/I compilers.

3.2.1 Reentrant Programs

The user may generate a reentrant program by the following procedures:

- He must specify REENTRANT in the OPTIONS list of his PROCEDURE statement. The compiler will generate code that is reenterable for both machine instructions and compiler-created work areas.

  An example of this is presented below:

```
EX: PROCEDURE OPTIONS
    (REENTRANT);
    .
    .
    .
    END EX;
```

- He must insure that any static storage in his procedure is read-only, not updatable.

3.2.2 Linkages Between PL/I and Other Languages

To exchange information between programs written in different languages, data formats must be compatible. PL/I provides the following facilities for this purpose:

- The programmer can specify a non-PL/I language with his OPTIONS attribute in the entry name declaration.

```
DECLARE SUBI ENTRY OPTIONS
(FORTRAN);
```

  or specify the language in the program invoking the PL/I procedure:

```
SUB2: PROCEDURE OPTIONS (COBOL);
    .
    .
    .
    END;
```

- The programmer can specify by means of OPTION attributes (NOMAP, NOMAPIN, NOMAPOUT) whether COBOL structures and FORTRAN data arrays should be converted to PL/I data structures and arrays.
- He can specify whether PL/I interruption processing (INTER) should be used for interruptions occurring during execution of non-PL/I subprograms.
- He can specify ENVIRONMENT attributes to map PL/I file structures onto COBOL data sets.

3.2.3 Program Structures

The PL/I programmer can create simple, overlay, dynamic link, and dynamic program structures. A program is processed as a dynamic program structure when FETCH and RELEASE statements are used to initiate loading and deletion of procedures. To create more complicated program structures, the user specifies the DYNAMIC option for the linkage

editor. Modules are then loaded only as required at execution time, facilitating changes to subroutines during link-editing as described in Section 8.6.4 of Part 2 of this publication.

3.2.4 Multitask Facilities

In a single job step, several tasks can be attached, detached, and processed asynchronously. Data sets can be shared among asynchronously-executing tasks.

The programmer attaches a subtask by furnishing one or more of the multitasking attributes (TASK, EVENT, and PRIORITY) in the corresponding CALL statement. A WAIT statement synchronizes execution of a subtask. An EXIT, RETURN, END or STOP statement ends a subtask.

A task and its subtasks can share a data set if corresponding PL/I files are open when the subtasks are attached. The EXCLUSIVE attribute locks a file to prevent interference from other tasks.

3.2.5 Dynamic Storage Management

Dynamic storage management in a PL/I program facilitates the following processing:

- Determining the length of a string variable, upper and lower bounds of an array, and the size of an area variable.
- Allocating a storage area and returning it during execution by ALLOCATE and FREE statements.

3.2.6 Optimization Procedures

The PL/I compiler provides optimization facilities which increase execution speeds and reduce the sizes of object modules.

Global Optimization

- Common subexpressions are calculated only once.
- Invariant expressions are transferred out of loops.
- Within a loop, multiplication by the induction variable is changed to repeated additions of the corresponding argument.
- Constants replace expressions whenever possible.
- Unnecessary assignment statements are eliminated.

Assignment optimization

When arrays with the same structure and data attributes are assigned or moved, the compiled instructions assign aggregate rather than individual elements whenever possible.

Inline coding

Data conversion, record input/output (CONSECUTIVE organization), string-processing and built-in functions are performed by in-line instructions whenever possible, reducing the number of library subroutine calls.

3.2.7 Conversational Processing

Since the PL/I compiler can be used under TSS control, a user can generate, debug, and execute a program from a terminal. The PL/I compiler offers the TERMINAL, LMESSAGE, and SMESSAGE options during compilation for conversational processing. The TERMINAL option specifies that error messages are immediately displayed at the terminal; the LMESSAGE/SMESSAGE option specifies whether long (detailed) or short diagnostic messages are to be displayed.

OS IV/F4 PL/I provides the following special facilities for conversational processing:

- PL/I syntax checker
  This independent TSS processor checks for syntactical errors, line by line. Since the syntax checker operates under the TSS editor, the user can make necessary corrections and continue entering his source program.
- PL/I prompter
  The user can request this facility from his terminal; it invokes the PL/I compiler and executes the module after compilation, prompting the terminal user for necessary link-edit and JCL parameters.

3.2.8 The PL/I Preprocessor

The OS IV/F4 PL/I compiler consists of a preprocessor and the body of the compiler.

A percent sign (%) precedes each preprocessor statement. The PL/I **preprocessor** scans each source program for preprocessor statements, which are executed as soon as they are encountered. Output from the preprocessor is an altered version of the source pro-

```
% DECLARE I FIXED;
% I = 0;
% LAB: I = I + 1;
  Z(I) = X(I) + Y(I);
% IF I < = 10 % THEN % GO TO LAB;
% DEACTIVE I;

        ⇩ Preprocessor

Z(1) = X(1) + Y(1)
Z(2) = X(2) + Y(2)
        .
        .
        .
Z(10) = X(10) + Y(10)
```

Fig. 3.1 Example of using preprocessor statements

gram, which becomes the principal input to the body of the compiler. An example of preprocessing is shown in Fig. 3.1.

3.2.9 Data Communications

An OS IV/F4 PL/I program can initiate linkage to a remote terminal, as described in Chapter 6 of Part 2 of this publication. A PL/I program can be designed to respond to terminal-initiated dialogs. The PL/I program need not be designed for a particular type of terminal or communications linkage; it is generally device-independent.

3.2.10 Data Sets

A PL/I program can create, access, and process any of the common file organizations for magnetic tape or DASD:

- Sequential data sets on magnetic tape, including seven-track as well as nine-track format, decimal (ANS) as well as EBCDIC format, etc.
- Sequential data sets on DASD, including members of partitioned data sets.
- Direct data sets, with/without keys.
- Virtual sequential data sets.

VSAM data sets can be accessed in two fundamentally different ways, as specified by the ENVIRONMENT attribute:

- ENV (VSAM)
  This permits the programmer to use the full capabilities of VSAM, as described in Chapter 5 of Part 2 of this publication.
- ENV(INDEXED CONSECUTIVE)
  This permits a PL/I program to perform ISAM-like functions on entry-sequenced and key-sequenced VSAM data sets, as described in Chapter 5 of Part 2.

3.2.11 Program Testing Aids

OS IV/F4 PL/I offers the following facilities for detecting and diagnosing errors with source-language displays of identifier names and conversion of execution-time data to their originally-specified formats (binary, hexadecimal, character, packed decimal, etc.):

- Tracing execution flow within each PL/I procedure.
- Tracing execution flow between PL/I procedures.
- Displaying names and values of selected variables whenever their values change.
- Checking the ranges of subscripts for designated variables, to avoid overwriting other data areas and/or programs.

The following PL/I statements generate diagnostic printouts, as selected by the programmer:

- PUT DATA statements.
- Condition prefixes (SUBSCRIPT RANGE, SIZE, etc.).
- ON, SIGNAL and REVERT statements.
- CHECK conditions.
- Asterisked lines, used under programmer control either as comments statements or debugging statements.

The following source-program listing and debugging options are available to programmers:

- FLOW listing.
- COUNT option.
- STMT/GOSTMT option.
- NUMBER/GONUMBER option.
- OFFSET option.
- DEBUG statement.

3.2.12 Other Features

User handling of interruptions
By furnishing one or more ON conditions—specifying various types of interruptions or other unusual events he wishes to handle himself—the programmer can execute procedures of arbitrary size and sophistication. This type of procedure is essentially an "exit routine" in the sense of Chapter 8 of Part 2 of this publication. After this procedure has completed, the programmer can return to his interrupted mainline program, or he can redirect control permanently to another procedure.

Recursive calls
By specifying the RECURSIVE attribute for a procedure, the programmer can execute it recursively. PL/I library subroutines and ON-units can always be used recursively.

List processing
Based variables and pointer variables permit the programmer to manipulate chained lists of data. He has such functions as ADDR and NULL to assist manipulation of based, pointer, offset, and area variables. ALLOCATE and FREE statements permit him to control when and how much storage is allocated to lists as well as for other dynamic-allocation needs.

Built-in subroutines and functions
Certain common functions such as SUBSTRING, CHARACTER, ABS, and ADDR are expanded into inline instructions by the OS IV/F4 PL/I compiler whenever feasible.

Invocation of other subsystems and OS IV/F4 facilities
The PL/I programmer can easily invoke such

packages as sort/merge. He can take checkpoints in the middle of his program. He can request snapshot dumps as well as terminal dumps, which are described in Chapter 8 of Part 2 of this publication. At the end of each job step, he can set a return code for the OS IV/F4 step terminator in his RETURN statement.

3.3 REQUIRED CONFIGURATION

The configuration of I/O functions (and associated DD names) required for the PL/1 compiler is shown in Fig. 3.2.

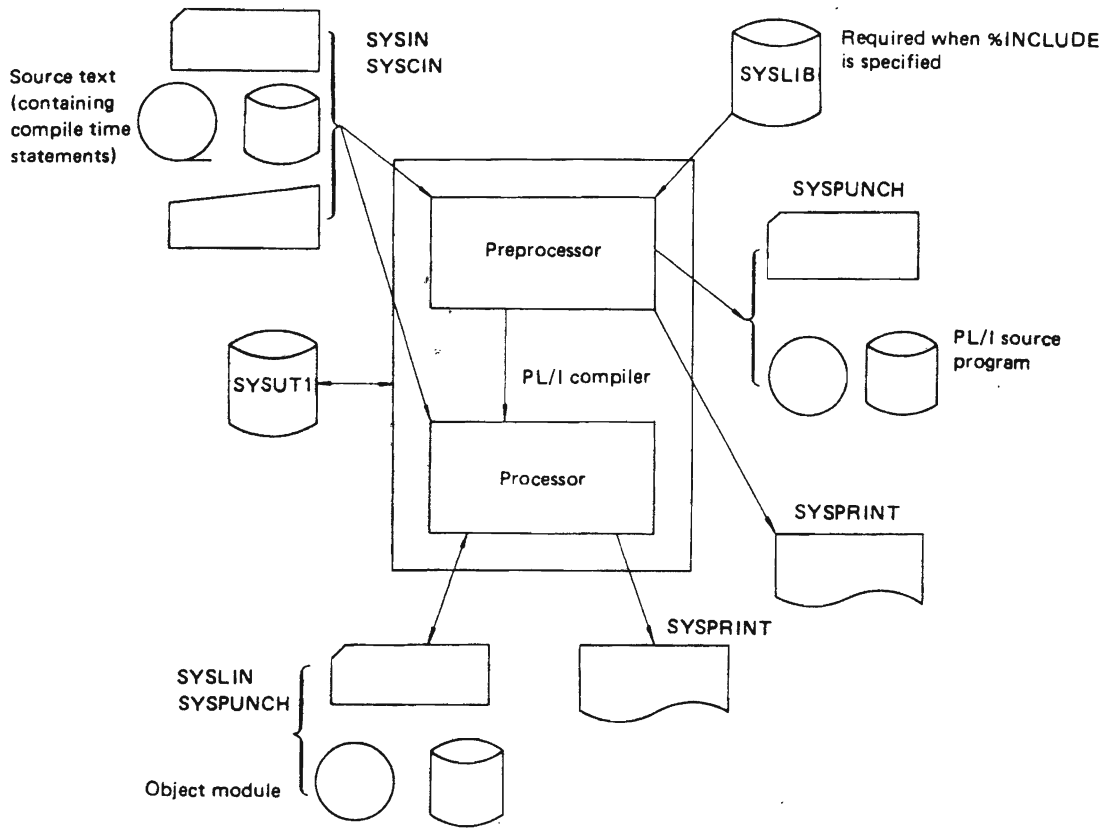


Fig. 3.2 PL/I compiler unit configuration diagram

CHAPTER 4
ALGOL

4.1 OVERVIEW

ALGOL is an algorithmic language for describing computational processes, especially suitable for scientists and engineers. The OS IV/F4 ALGOL compiler provides a number of extensions to the JIS ALGOL 5060 standard. This section presents a general description of OS IV/F4 ALGOL. For a more complete treatment, the user should consult the following publications:

- **FACOM OS IV ALGOL Reference Manual.**
- **FACOM OS IV/F4 ALGOL User's Guide.**

The main characteristics of the OS IV/F4 ALGOL compiler are as follows:

- Punched cards and paper tape can be used for entering source programs.
- Upper-case and lower-case letters can be used in identifiers and character strings.
- ALGOL can manipulate many different data types: integers, real and logical constants, double-length integers, double-precision real numbers, complex numbers, double-precision complex numbers, and character strings.
- ALGOL provides arithmetical, logical, address, and character expressions.
- OS IV/F4 ALGOL furnishes all standard functions and variable functions in addition to those required by Japan Industrial Standard (JIS) ALGOL.
- OS IV/F4 ALGOL provides a number of I/O processing functions.
- OS IV/F4 ALGOL furnishes a debugging facility.
- OS IV/F4 ALGOL can manipulate columns, rows, and cross-sections of data arrays.
- OS IV/F4 ALGOL provides a full range of program and subprogram linkages.

4.2 HIGHLIGHTS

4.2.1 Program Linkages

When the programmer wishes to link several ALGOL programs or procedures, he can use either global symbols or external symbols. A **global symbol** is an identifier which is defined in one procedure and can be referenced in one or more **inner procedures** or other **external procedures.** An **external symbol** is one which identifies a scalar variable, an array name, or a procedure name defined in another procedure or program. Each external symbol must be identified in procedures where it is used, except that a subprogram name is automatically and implicitly defined as a global symbol.

4.2.2 Standard and Variable Functions

Many OSIV/F4 ALGOL functions are predefined, so that the programmer need not define them as external symbols.

Standard functions
The standard functions of OS IV/F4 ALGOL includes most of the frequently used arithmetical functions:

ABS, SIGN ARCCOS, ARCSIN, ARCTAN,
COS, SIN, TAN, COSH, SINH, TANH
EXP, LN, LOG, MAX, MAXn,MIN, MINn,
MOD, REM, NRANDOM, RANDOM
SQRT EQUIV, LENGTH
CLOCK, CLOCKM

Variable functions
The variable functions of OS IV/F4 ALGOL perform various conversions of identifiers or expressions:

ENTIER, ENTIEL, FLOAT, FLOATL
IMAGINARY, COMPLEX, CONJUGATE
FIX, FIXL

Character handling functions
Standard facilities for character strings are REPLACE, INSERT, and DELETE functions.

4.2.3 I/O Facilities

OS IV/F4 ALGOL provides several I/O procedures to process information read/written to data sets (sequential organization and direct organization).

Two sets of procedures are provided: standard procedures prescribed in JIS ALGOL, and nonstandard procedures implemented specially for OS IV/F4 ALGOL. In addition, INLIST/OUTLIST direct data sets can be accessed via LNOTE and LPOINT verbs. These procedures and verbs are shown in Table 4.1.

Table 4.1 I/O procedures

Datum/control field	Input	Output
Line feed	INLINEFEED	OUTLINEFEED
Integer item	ININTEGER	OUTINTEGER
Real item	INREAL	OUTREAL
Array	INARRAY	OUTARRAY
Symbol string	INSYMBOL	OUTSYMBOL
Formatted items	INPUTn (n=0–9)	OUTPUT (n=0–9)
	INLIST	OUTLIST
Complex item	INCOMPLEX	OUTCOMPLEX
Standard I/O control	READ READA READ ARRAY	CRLF EJECT LFEED PRINT PRINTA PRINTFIX PRINTI PRINTR PRINTSTRING SPACE WRITEARRAY
Device-dependent control	INPUT REWIND	OUTPUT REWIND
Direct I/O position	POINT	POINT
Direct I/O feedback	NOTE	NOTE
SYSIN/SYSOUT stream	GET GETA GETD GETMT	PUT PUTA PUTD PUTMT
Change standard format	SFORM	SFORM

4.2.4 Debugging Facilities

To facilitate program debugging, OS IV/F4 ALGOL provides four procedures. Each procedure can display a character string corresponding to the formal parameter STRING. Additional outputs from each function are described below:

- CHECKPOINT(STRING)
  When an error occurs during execution, an error message is displayed.
- SNAP(STRING,E)
  The value of an arithmetic expression (E) is evaluated and displayed.
- SNAPARRAY(STRING,ARRAY)
  This procedure displays the value of STRING followed by all elements of the array in lexicographic order.
- SNAPLIST(STRING,LIST)
  This procedure displays the value of STRING followed by the elements named in the list.

4.2.5 Other Features

Asterisked lines
If a statement contains an asterisk in its first position, the programmer can decide at each compilation whether to treat the statement as a comment or as a compilable statement.

Automatic precision increase (API) facility
API permits the programmer to double the precision of all arithmetical operations with a single compile-time option. Integers are processed as double-precision integers, real and complex fullword numbers as their doubleword counterparts, and real and complex doubleword numbers as their extended-precision counterparts.

Constants and work areas are automatically doubled in size when API is invoked.

CHECK option
The programmer can designate that some or all of the following conditions be checked during execution of his ALGOL program:
- Subscripts within declared ranges.
- Subscript ranges declared correctly.
- Agreement of number and modes of parameters in a procedure call with the declared number and modes.

4.3 REQUIRED UNIT CONFIGURATION

The configuration of I/O functions (and associated DD names) required for the ALGOL compiler is shown in Fig. 4.1.

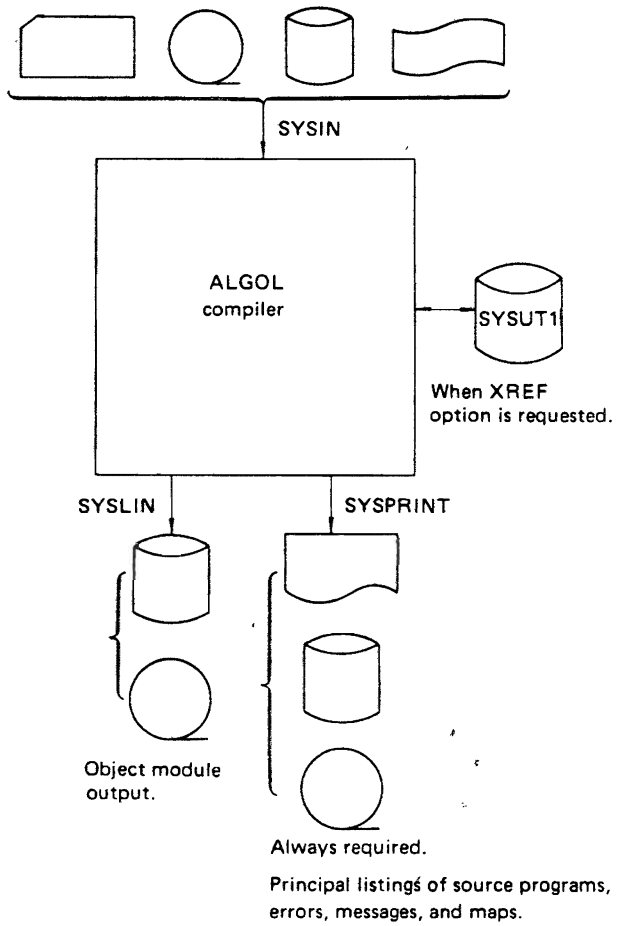


Fig. 4.1 ALGOL compiler unit configuration diagram

CHAPTER 5
SL/100

5.1 OVERVIEW

SL/100 is an OS IV/F4 language used primarily for implementing systems software rather than applications packages. It includes all features and syntax of the OS IV/F4 Assembler Language, including its macro language facility. In addition, it furnishes many statement types of higher-level languages such as assignment statements ("A=B+C"), IF, DO, and GOTO statements. SL/100 is therefore language which retains full flexibility at the machine-language level yet gives programmers several conveniences of higher-level languages.

This chapter presents a brief description of the OS IV/F4 SL/100 facility. For more detailed information, the user is referred to the following publications:

- **FACOM OS IV SL/100 Reference Manual.**
- **FACOM OS IV/F4 SL/100 User's Guide.**

The SL/100 compiler comprises a compiling phase and an assembly phase. In the **compiling phase,** non-assembler statements are converted into assembler source statements; ordinary Assembler statements are copied unchanged into the compiled output. In the **assembly phase,** output from the compiling phase is converted into machine language, and an object module program listing and other optional outputs are produced. The assembly phase is functionally identical to the OS IV/F4 assembler.

5.1.1 Highlights

(1) SL/100 can create all M series data formats: bit, character, halfword, etc.
(2) Using SL/100—either alone or in conjunction with certain other languages—the programmer can create load modules in any of the four OS IV/F4 structures: simple, overlay, dynamic program, or dynamic link.
(3) SL/100 programs can be more concisely coded and more easily maintained than assembler-language programs.
(4) SL/100 automatically optimizes allocation of general registers to indexing and integer-arithmetic calculations. The programmer can designate which register variables are to be preferentially allocated general registers.
(5) Like the assembler, SL/100 has access to all facilities of OS IV/F4 Job Management, task management, and data management.

5.2 DATA FORMATS

5.2.1 Declarations

Data items are declared by assembler pseudo-instructions: "DC" for constants, "DS" for storage areas. Registers are declared by special SL/100 statements which have no assembler counterparts. Control sections of various types are specified by START, CSECT, DSECT, PSECT, DXD, and COM pseudo-operation codes, just like their assembler counterparts. The reader should refer to Section 6.3.1 for a discussion of dividing programs into control sections.

5.2.2 Types of Operands

Operands are classified as constants, variables and functions.

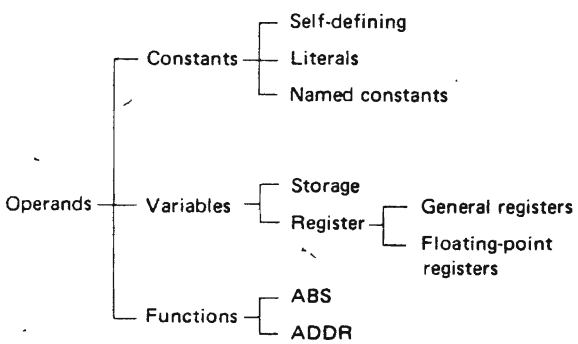


Fig. 5.1 Types of SL/100 operands

Constants

Self-defining constants can appear as displacements or immediate operands in instructions. Constants can be defined and referenced by usages in **literals**, self-defining constants preceded by "=". A constant can also be defined with the DC pseudo-instruction, then referenced by name. SL/100 constants can have the following formats and (parenthesized) type codes:

- Character (C) – 8-bit code for any of the 256 EBCDIC characters.
- Bit constants – represented in binary or hexadecimal notation.
  - binary (B) – binary digit of 0 or 1
  - hexadecimal (X) – 4-bit code in the range 0–9 and A-F.
- Fixed-point binary
  - halfword (H)
  - fullword (F)
- Floating point format
  - short floating point (E)
  - long floating point (D)
  - extended floating point (L)
- Packed decimal (P) – two decimal digits packed into each byte with the sign in the right-most four bits
- Zoned decimal (Z) – each byte contains a four-bit zone or sign and one decimal digit.

Variables

Storage operands can be expressed as scalar variables or as subscripted array elements (one or two subscripts). Register operands can be referenced by previously-declared symbols. Storage addresses can also be obtained by the ADDR function.

Functions

SL/100 provides two functions of particular convenience for creating system programs.

- The ABS function obtains the absolute value of a binary or floating point argument. For example, the SL/100 statement

  A = ABS(B)

  generates the following assembler instructions if A and B are fullword integers:

```
L     0,B
LPR   0,0
ST    0,A
```

- The ADDR function obtains the 24-bit virtual storage address of an argument.

  For example, when the statement

  C = ADDR(D)

  is compiled, the following assembler instructions are generated (if D is within the range of a current USING statement):

```
LA    0,D
ST    0,C
```

5.3 PROCEDURAL STATEMENTS

The SL/100 procedural statements fall into three categories:

(1) Executable statements
- assignment
- IF
- DO and END
- GOTO

(2) Auxiliary statements
- DECLARE
- RELEASE
- RESTRICT

(3) Compile-time statement
- INCLUDE

Assignment statements

This type of statement is used for internal computations and movements of data, of which the following are examples:

- The expression on the right side of the assignment statement is evaluated, and its result is stored into the variable (or array) specified on the left side.
- A specified bit in a particular byte is set ON or OFF.
- A specified field is assigned the value of a storage variable of up to four bytes or a constant.
- As in (3), a one-byte value can be propagated into all bytes of a field.
- The contents of several contiguous general registers are stored into one storage variable (equivalent to a store multiple instruction); conversely, contiguous registers can be loaded from a storage variable (equivalent to load multiple).
- Three integers are added together, and their sum is placed into the designated register variable (equivalent to a load address instruction).
- Sum/difference of operands is computed either arithmetically or **logically** (using high-order bit as a datum rather than a sign), then stored into the indicated variable.
- Quotient is assigned to a scalar variable, if division is requested; quotient and remainder are both assigned if two variables appear on the left side of the assignment statement.
- Logical AND, OR, EXCLUSIVE-OR, and NOT operations are performed, the result being assigned to the designated bit variable (or bit array).

IF statement

Just as for a high-level language, an SL/100 IF statement tests one or more conditions (using AND and OR for conjunction and disjunction) and conditionally branches according to the result.

- Comparisons
  Arithmetical and/or logical comparison
- Bit tests
- Condition-code tests
  Equivalent to branch-on-condition instructions.

DO and END statements
These bracket one group of SL/100 statements executed consecutively. They provide a means for visual verification of procedural units, thus assisting debugging and documentation.

GOTO statement
This statement generates an unconditional Branch instruction.

DECLARE statement
This statement declares the attributes of a register variable, constant, or bit variable.

RESTRICT and RELEASE statements
RESTRICT places a designated general register under the control of the programmer, so that he can allocate it to a specific arithmetical or pointer functions. The programmer issues RELEASE to permit SL/100 to assign this general register according to its own algorithm.

INCLUDE statement
This statement has the same function as the Assembler-language COPY statement.

5.4 FLOATING-POINT FACILITIES

SL/100 provides several floating-point functions as well as data definitions comparable to those of the assembler-language DC and DS statements.

Floating-point data
Each floating-point number is represented by a seven-bit characteristic and a fractional part whose length depends on the requested precision. SL/100 furnishes three floating-point formats:
- short (single precision) format,
- long (double precision) format, and
- extended (quadruple-precision) format.

Their storage requirements are, respectively, 32, 64, and 128 bits. Floating-point formats can be specified for constants, storage variables, register variables, and functions.

Floating-point assignment statement
The floating-point assignment statement has the following functions:
- Move a floating-point number between storage and one or more floating-point hardware registers.
- Conversion of floating-point numbers to/from fixed point numbers.
- The value of a floating-point expression is assigned to the indicated variable.

Floating-point comparisons
Floating-point expressions can be compared in IF statements.

Register declarations
A DECLARE statement can assign a mnemonic symbol to a (hardware) floating-point register.

5.5 DECIMAL ARITHMETIC FACILITIES

Decimal data
Decimal data are decimal constants and decimal variables in either packed or zoned format.

Decimal assignment statement
The decimal assignment statement has the following capabilities:
- Conversions of packed-decimal numbers to/from zoned decimal numbers.
- Transfers between packed decimal variables and register variables.
- The results of arithmetic operations with packed decimal numbers are assigned to the left side of the statement.

Decimal arithmetic comparisons
Packed decimal expressions can be compared in IF statements.

5.6 REQUIRED CONFIGURATION

The configuration of I/O functions (and associated DD names) required for the SL/100 compiler is shown in Fig. 5.2.

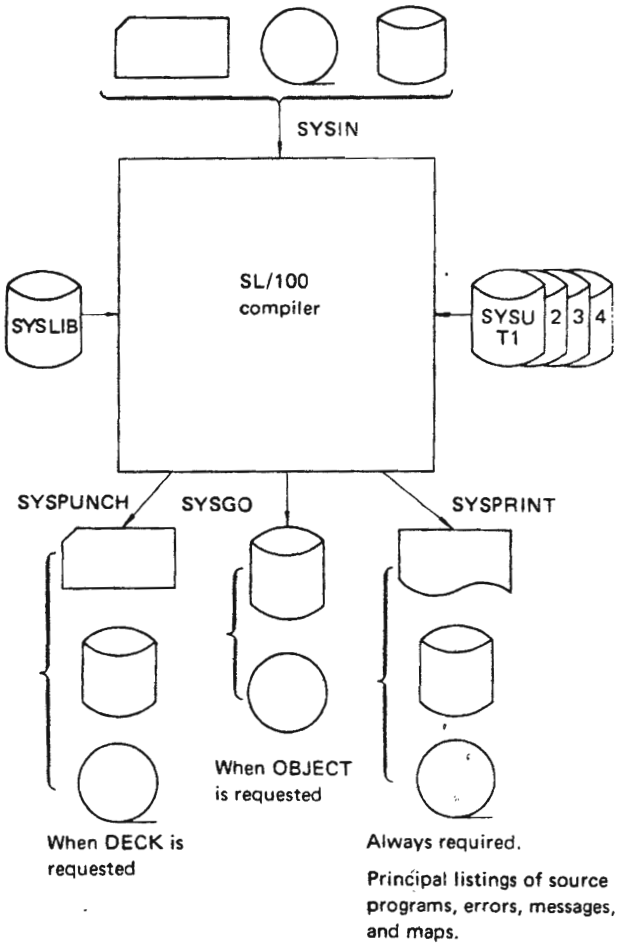


Fig. 5.2 SL/100 compiler unit configuration diagram

CHAPTER 6
ASSEMBLER

6.1 OVERVIEW

The assembler language is a symbolic programming language used to write programs close to the level of machine language in format and content. Mnemonic symbols are defined for machine language operation codes, and symbolic names can be used for storage addresses. By using these symbols, the programmer greatly reduces his level of effort and exposure to errors. Nonetheless, he can directly access all machine-language facilities, many of which are not available to him in higher-level programming languages (except SL/100).

This chapter presents a brief description of the OS IV/F4 assembler language. For more detailed information, the user should study the following publications:

- **FACOM OS IV Assembler Reference Manual.**
- **FACOM OS IV/F4 Assembler User's Guide.**

Assembler-language statements can be divided into three groups: machine instructions, assembler instructions, and macro instructions.

Machine instructions
The assembler language provides mnemonic operation codes for all machine instructions of FACOM M series computers. These codes are translated into machine language by the assembler.

Assembler instructions
The assembler language also provides mnemonic **assembler instruction** (or **pseudo-instruction**) operation codes for special functions performed by the assembler, such as storage allocation, base-register assignment, and displacement computations. With a few exceptions, assembler instructions do not generate machine-language instructions.

Macro instructions
A macro definition is a collection of assembler language statements. When invoked by a macro instruction, part or all of the macro definition is automatically copied into the user's program. This allows an Assembler-language programmer to generate instruction sequences without coding all necessary instructions each time he requires them. Macro definitions fall into two categories:

- **System macro definitions** provided by OS IV/F4, such as data management, job management and supervisor macro definitions.
- **User macro definitions** created by a programmer either for use in his current program or for incorporation into a library.

6.2 MACHINE INSTRUCTIONS

Machine instructions can be classified by their symbolic formats. Table 6.1 illustrates formats for the six classes of instructions.

Functions performed by machine instructions include status switching, input/output, arithmetical, logical, and branching operations.

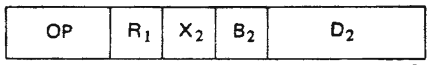
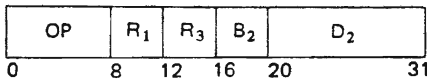
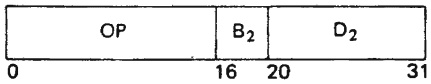
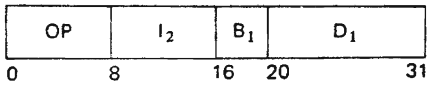
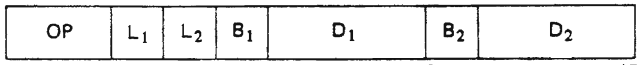
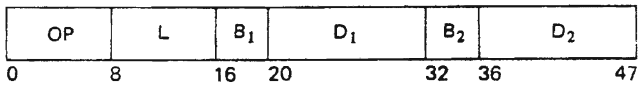
Status switching
A set of instructions is provided to switch the status of the CPU, main storage, virtual storage, and communications between systems. The choice between supervisor state and problem state determines whether the full set of instructions is valid. In the supervisor state, all instructions are valid. Privileged instructions are not valid in the problem state.

Input/output
The OS IV/F4 assembler language contains several input/output instructions, which provide the user with direct control of input/output operations.

Arithmetic operations
Additions, subtractions, multiplications, divisions, rounding, and comparisons can be performed upon one or two operands. Arithmetic instructions differ according to their data formats, which fall into three major classes:

- Fixed-point arithmetic
  The basic operand is a signed binary integer of fixed length. The operation is performed upon one operand in a register and another operand either in

Table 6.1 Machine instruction formats

Type	Operation	Instruction formats
RR	Register to register	
RX	Register to storage, indexed	
RS	Register to storage	
S	Implicit storage	
SI	Immediate to storage	
SS	Storage to storage	 

Key: OP – Operation code
R – Register
B – Base register
X – Index register
D – Displacement
L – Length
I – Immediate

a register or retrieved from main storage.

- Decimal arithmetic
  Operations are performed on packed or zoned decimal data of variable lengths. One or two lengths are specified in the instruction. Both operands are located in main storage.
- Floating-point arithmetic
  Operations are performed on floating-point numbers having short, long, or extended formats. These formats differ according to the lengths of their fractions. Hardware floating-point registers permit operations to be either register-to-register (RR) or storage-to/from-register (RX).

Logical operations
A logical operation can be performed on fixed-length or variable-length data: comparison, translation, editing, bit-testing, and bit-setting.

Branching instructions
Branching instructions provide conditional changes of instruction sequence, linkages to subroutines, or repetitions of loops. Conditional branches depend on arithmetical calculations concurrent with the Branch instructions (BXLE, BXH, BCT, BCTR) or more general tests performed earlier.

6.3 ASSEMBLER INSTRUCTIONS

Assembler instructions are requests for special services from the assembler. These statements do not always cause machine instructions to be generated. They include instructions for program sectioning, data definition, base register assignments and program control.

6.3.1 Program Sectioning and Linking

It is often convenient to divide a large program into control sections which can be compiled independently. The Assembler provides facilities for creating multisection programs.

CSECT, START, and END
A CSECT instruction identifies the beginning (or continuation) of a control section.

A START instruction assigns a name to the first

control section of a program. An END instruction terminates a program.

COM (COMMON)
A COM instruction identifies and reserves a common area of storage (COMMON data in FORTRAN, for example) that may be utilized by independent assemblies that have been link-edited (or loaded) together for execution.

DSECT
The DSECT instruction identifies the beginning (or resumption) of a **dummy control section.** It is assumed that storage is reserved by some other section of this assembly or else by another assembly. Address displacements for symbols defined in a dummy section are relative to the initial statement of the section.

External dummy section
An external dummy control section can be defined as a work area by a program. The area is not reserved by the assembler but is loaded dynamically when the programmer issues a GETMAIN macro instruction. Initial values cannot be assigned to external dummy control sections, which effectively define work areas used in common by several main programs, therefore generating a reentrant program. To define an external dummy section, Q-type address constants must be used with the following instructions:

- DXD (Define External Dummy)
  This instruction defines the size of an external dummy section.
- CXD (Cumulative External Dummy)
  This instruction defines the cumulative length of an external dummy section. The sum is stored by the linkage editor.
- Q-type address constant
  This constant defines displacement from the beginning of a dummy section defined either by a DXD or DSECT instruction. The displacement is stored by the linkage editor.

PSECT (prototype control section)
A PSECT instruction identifies the beginning (or continuation) of a prototype control section. Since storage is reserved by the assembler for a prototype control section, initial values can be defined for reenterable programs. The linkage editor segregates PSECTS in a load module, and the supervisor moves their contents into dynamically-acquired virtual storage Different from control sections defined by DSECTs, the programmer need not explicitly allocate storage for a PSECT; the supervisor allocates this storage automatically while loading his program, as described in Section 8.6.5 of Part 2 of this publication.

6.3.2 Addressing

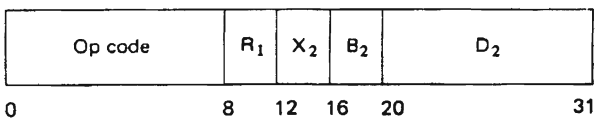
Each assembler-language storage address requires a **base register** (which contains a base address) and a **displacement,** which is added to the base address to obtain the virtual storage address. The programmer may specify a symbolic address and ask the assembler to determine its storage address in terms of its base register and displacement. The programmer must inform the assembler what base registers are available and what virtual storage address each register is assumed to contain throughout his program.

Addressing in the source program
USING and DROP instructions enable programmers to use base registers implicitly, leaving assignment of base registers and calculation of displacements to the assembler.

The USING instruction indicates that one or more general registers are available for use as base registers. The user guarantees that certain base address values will be in these registers for indicated sections of his program during its execution.

The DROP instruction specifies that a previously available general register may no longer be used as a base register.

The following example illustrates hardware registers for an RX-type instruction, where the second operand (designated by the triple "$X_2/B_2/D_2$") is a storage address:



The logical address of the storage operand is determined by adding the displacement D_2 to the base address in base register B_2 and the indexing value in the index register X_2. Only the low-order 24 bits (bits 8-31) of the registers are used in address calculation.

Base address + Indexing + Displacement
= Logical Address

A dynamic storage address is represented by a displacement in the page. If each page size contains 4K bytes and each segment contains 64K bytes, M series dynamic addresses will have the following format:

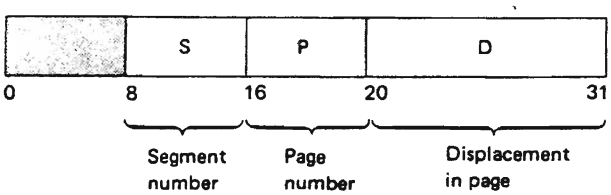
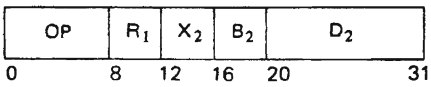
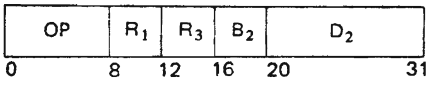
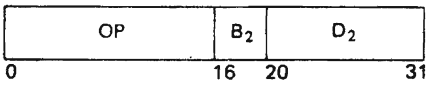
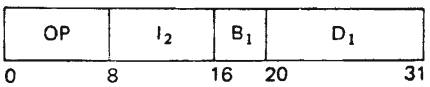
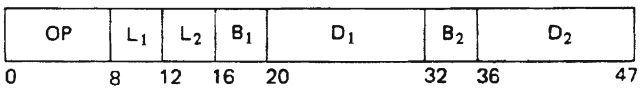
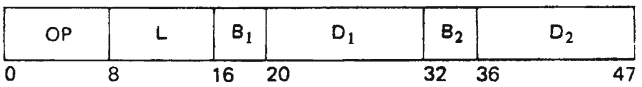


Table 6.1 Machine instruction formats

Type	Operation	Instruction formats
RR	Register to register	
RX	Register to storage, indexed	
RS	Register to storage	
S	Implicit storage	
SI	Immediate to storage	
SS	Storage to storage	 

Key: OP – Operation code
R – Register
B – Base register
X – Index register
D – Displacement
L – Length
I – Immediate

a register or retrieved from main storage.

- Decimal arithmetic
  Operations are performed on packed or zoned decimal data of variable lengths. One or two lengths are specified in the instruction. Both operands are located in main storage.
- Floating-point arithmetic
  Operations are performed on floating-point numbers having short, long, or extended formats. These formats differ according to the lengths of their fractions. Hardware floating-point registers permit operations to be either register-to-register (RR) or storage-to/from-register (RX).

Logical operations
A logical operation can be performed on fixed-length or variable-length data: comparison, translation, editing, bit-testing, and bit-setting.

Branching instructions
Branching instructions provide conditional changes of instruction sequence, linkages to subroutines, or repetitions of loops. Conditional branches depend on arithmetical calculations concurrent with the Branch instructions (BXLE, BXH, BCT, BCTR) or more general tests performed earlier.

6.3 ASSEMBLER INSTRUCTIONS

Assembler instructions are requests for special services from the assembler. These statements do not always cause machine instructions to be generated. They include instructions for program sectioning, data definition, base register assignments and program control.

6.3.1 Program Sectioning and Linking

It is often convenient to divide a large program into control sections which can be compiled independently. The Assembler provides facilities for creating multisection programs.

CSECT, START, and END
A CSECT instruction identifies the beginning (or continuation) of a control section.

A START instruction assigns a name to the first

control section of a program. An END instruction terminates a program.

COM (COMMON)

A COM instruction identifies and reserves a common area of storage (COMMON data in FORTRAN, for example) that may be utilized by independent assemblies that have been link-edited (or loaded) together for execution.

DSECT

The DSECT instruction identifies the beginning (or resumption) of a **dummy control section.** It is assumed that storage is reserved by some other section of this assembly or else by another assembly. Address displacements for symbols defined in a dummy section are relative to the initial statement of the section.

External dummy section

An external dummy control section can be defined as a work area by a program. The area is not reserved by the assembler but is loaded dynamically when the programmer issues a GETMAIN macro instruction. Initial values cannot be assigned to external dummy control sections, which effectively define work areas used in common by several main programs, therefore generating a reentrant program. To define an external dummy section, Q-type address constants must be used with the following instructions:

- DXD (Define External Dummy)
  This instruction defines the size of an external dummy section.
- CXD (Cumulative External Dummy)
  This instruction defines the cumulative length of an external dummy section. The sum is stored by the linkage editor.
- Q-type address constant
  This constant defines displacement from the beginning of a dummy section defined either by a DXD or DSECT instruction. The displacement is stored by the linkage editor.

PSECT (prototype control section)

A PSECT instruction identifies the beginning (or continuation) of a prototype control section. Since storage is reserved by the assembler for a prototype control section, initial values can be defined for reenterable programs. The linkage editor segregates PSECTS in a load module, and the supervisor moves their contents into dynamically-acquired virtual storage Different from control sections defined by DSECTs, the programmer need not explicitly allocate storage for a PSECT; the supervisor allocates this storage automatically while loading his program, as described in Section 8.6.5 of Part 2 of this publication.

6.3.2 Addressing

Each assembler-language storage address requires a **base register** (which contains a base address) and a **displacement,** which is added to the base address to obtain the virtual storage address. The programmer may specify a symbolic address and ask the assembler to determine its storage address in terms of its base register and displacement. The programmer must inform the assembler what base registers are available and what virtual storage address each register is assumed to contain throughout his program.

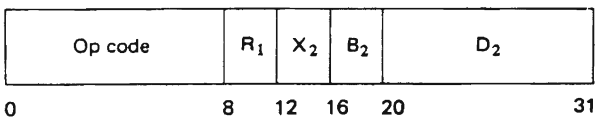
Addressing in the source program

USING and DROP instructions enable programmers to use base registers implicitly, leaving assignment of base registers and calculation of displacements to the assembler.

The USING instruction indicates that one or more general registers are available for use as base registers. The user guarantees that certain base address values will be in these registers for indicated sections of his program during its execution.

The DROP instruction specifies that a previously available general register may no longer be used as a base register.

The following example illustrates hardware registers for an RX-type instruction, where the second operand (designated by the triple "$X_2/B_2/D_2$") is a storage address:



The logical address of the storage operand is determined by adding the displacement D_2 to the base address in base register B_2 and the indexing value in the index register X_2. Only the low-order 24 bits (bits 8-31) of the registers are used in address calculation.

Base address + Indexing + Displacement
= Logical Address

A dynamic storage address is represented by a displacement in the page. If each page size contains 4K bytes and each segment contains 64K bytes, M series dynamic addresses will have the following format:

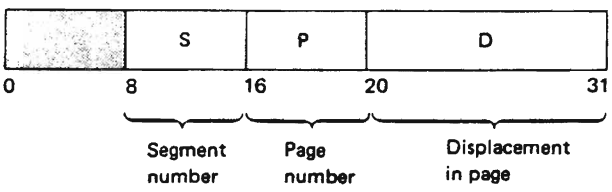
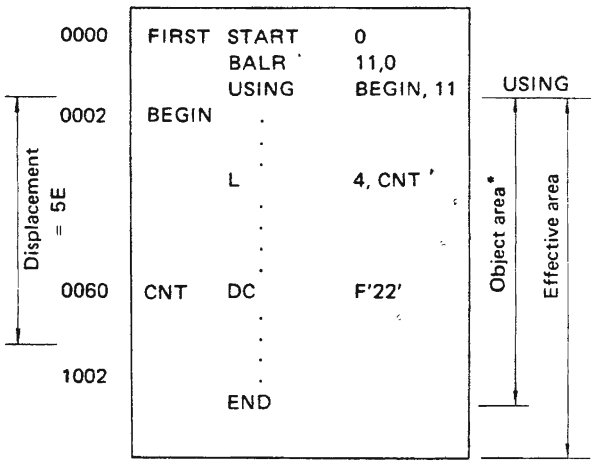


Fig. 6.1 illustrates a USING instruction designating a base register and the result of assembling a LOAD instruction.

Symbolic linkages

Symbols may be defined in one program and referenced in another program, thus linking independently-assembled programs. Linkage symbols are defined by ENTRY, EXTRN and WXTRN instructions.

An ENTRY instruction identifies one or more symbols in this program which will be referenced by other programs. These symbols are entered into the external symbol dictionary (ESD) of the assembled object module.



* The assembler instruction is translated into the following machine instruction

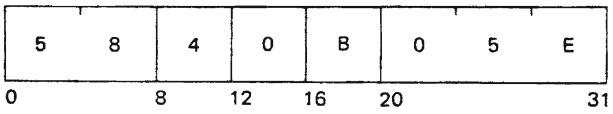


Fig. 6.1 Example of the USING instruction

An EXTRN (external symbols) instruction identifies one or more symbols used by this program but defined in another program. Examples of these instructions appear in Fig. 6.2.

A WXTRN (weak external) instruction identifies one or more **weak external symbols,** which differ from ordinary external symbols as follows: the linkage editor attempts to resolve all of the latter among object modules, load modules, and libraries which the user furnishes, but it only conditionally attempts to resolve weak external symbols. These conditions are based on parameters specified by the user to the linkage editor.

```
A       START
        .
        .
        EXTRN       B
        .
        L           10, NXT
        BR          10
        .
        .
        .
        L           4, ENT
        BR          4
        .
        .
        .
NXT     DC          A(B)
ENT     DC          V(SUB)
        .
        .
        .
        END
```

```
B       START
        ENTRY       SUB
        .
        .
SUB     .
        .
        .
        END
```

Fig. 6.2 Example of using ENTRY and EXTRN instructions

6.3.3 Symbol and Data Definitions

Some assembler instructions cause storage areas to be set aside for constants and data areas.

- EQU (Equate Symbol)
  An EQU instruction defines a symbol by assigning to it the attributes of the expression in the operand field. The EQU instruction can be used to name general or floating-point registers, immediate operands, and other program elements.
- DC (Define Constant)
  A DC instruction defines one or more constants in virtual storage, as shown in Table 6.2
- DS (Define Storage)
  A DS instruction reserves one or more storage areas and assigns names to these areas.
- CCW (Channel Command Word)
  A CCW instruction defines the four fields of one eight-byte Channel Command Word.

6.3.4 Assembler Control Instructions

Assembler control instructions set the Location Counter, control the program listing, and indicate the statement formats.

Address alignment control

- ORG (Set Program Origin)
  An ORG instruction sets/alters the value of the Location Counter for the current control section.

Table 6.2 Summary of constants

Code	Length (bytes)		Format
	Implied	Allowable	
B	–	1 – 256	Binary digits
C	–	1 – 256	Characters
X	–	1 – 256	Hexadecimal digits
F	4	1 – 8	Fixed-point binary (signed)
H	2	1 – 8	Fixed-point binary (signed)
P	–	1 – 16	Packed decimal
Z	–	1 – 16	Zoned decimal
E	4	4	Short floating-point
D	8	8	Long floating-point
L	16	16	Extended floating-point
A	4	1 – 4	A type address constant (value of address)
Y	2	1 – 2	Y type address constant (value of address)
S	2	2	S type address constant (address in base-displacement form)
V	4	3 – 4	V type address constant* (externally defined address value)
Q	4	1 – 4	Q type address constant (symbol naming external dummy section)

* The linkage editor resolves all external symbols of static structures. External symbols belonging to dynamic structures remain unresolved and DALTAB is generated. (Refer to 8.2.3 Processing of Program Structure)

- LTORG (Set Origin for a Literal Pool)
  A LTORG instruction causes all literals which the Assembler has scanned thus far (or since the last LTORG instruction in this assembly) to be assembled at appropriate boundaries, starting at the first doubleword boundary following the LTORG statement.
- CNOP (Conditionally Generate No-Operation Instructions)
  A CNOP instruction allows the programmer to align an instruction at a halfword boundary whose address is a prespecified offset from a word or doubleword boundary.

Input format and sequence control

- ICTL (Input Format Control)
  An ICTL statement allows the programmer to alter the normal format of his source-program statements by specifying the beginning, ending, and continuation columns for each source-program statement.
- ISEQ (Input Sequence Checking)
  An ISEQ instruction requests the Assembler to check the sequence of input statements. Sequence-checking begins with the first statement following the ISEQ statement and terminates with an ISEQ statement bearing a blank operand.

Listing control

- PRINT
  A PRINT instruction controls printing of the assembly listing. The following operands may be specified:
  ON/OFF– listing is (is not) printed.
  GEN/NOGEN–macro-generated instructions are (are not) printed
  DATA/NODATA–full (first eight bytes) constants are printed.
- TITLE
  A TITLE instruction enables the programmer to print his own heading on an assembly listing. Also, the contents of the name field of the TITLE statement are punched into columns 73–76 of any assembly output cards. Any character string in the operand field is printed at the top of each page of the assembly listing thereafter.
- EJECT
  An EJECT instruction causes the next line of the assembly listing to appear at the top of a new page.
- SPACE
  A SPACE instruction inserts one or more blank lines into the listing.

Punch control

- PUNCH
  A PUNCH instruction requests that data in its operand field be punched in edited format.
- REPRO
  A REPRO instruction requests that data on the following statement be punched without editing.

Redefinition of operation codes

An OPSYN (Operation Code Synonym) instruction allows a programmer to define a new operation code for an existing code or to redefine an existing operation code.

Saving and restoring status

- PUSH
  A PUSH instruction saves current PRINT and USING status.
- POP
  A POP instruction restores the former PRINT and USING status.

```
         MACRO
&NAME    MOVE     &TO,&FROM
         AIF      (N' &TO EQ 0) .M1
         AIF      (N' &FROM EQ 0) .M2
&NAME    ST       2, AREA
         L        2, &FROM
         ST       2, &TO
         L        2, AREA
         MEXIT
.M1      MNOTE    'NO FIRST OPERAND'
         MEXIT
.M2      MNOTE    'NO SECOND OPERAND'
         MEND
```

MACRO statement
PROTOTYPE statement
Conditional assembler instruction statements
Model statements
MEXIT statement
MNOTE statement
MEND statement

Fig. 6.3 Example of macro definition

6.4 MACRO LANGUAGE

The OS IV/F4 macro language enables the programmer to define a frequently-used sequence of assembly-language statements as a macro definition. This facility simplifies coding of programs and reduces the incidence of coding errors.

6.4.1 Macro Instructions

A macro instruction is processed by the Assembler just like an assembler statement. The Assembler substitutes a sequence of Assembler-language statements for each macro instruction. These generated statements are then processed like other Assembler-language statements. The Assembler-language statements are obtained either from a library macro definition or from a macro definition submitted with this particular source program.

Three types of parameters may be defined for each macro instruction:

- Positional parameters
  The programmer must furnish values for positional parameters in a prespecified sequence, separated by commas.
- Keyword parameters
  The programmer can furnish values for keyword parameters in any sequence he chooses. Keyword parameters are most useful when their number is large and many parameters assume default values most of the time.
- Mixed-mode parameters
  Many macro definitions define a sequence of positional parameters followed by a collection of keyword parameters; these correspond to **mixed mode macro instructions.**

6.4.2 Macro Definitions

A macro definition is a set of statements that provides the Assembler with the mnemonic operation code and format of corresponding macro instructions and the sequence of statements the Assembler selectively generates to replace each occurrence of the macro instruction. See Fig. 6.3.

A macro definition consists of the following statements:

- MACRO
  The **macro definition header statement** indicates the beginning of a macro definition.
- Macro instruction prototype
  Specifies the mnemonic operation code and the format of all macro instructions that refer to the macro definition.
- Model statements
  These are Assembler statements from which other sequences of Assembler-language statements are generated.
- MEXIT
  The **macro definition exit statement** indicates to the Assembler that all necessary statements have generated from this macro definition.
- MNOTE
  An MNOTE statement requests the Assembler to write an error message during the macro-generation process.
- Conditional assembly instructions
  These allow the programmer to generate different sequences of statements from the same macro definition.
- MEND
  The **macro definition trailer statement** indicates the end of a macro definition.

6.4.3 Conditional Assembler Instructions

The conditional assembly instructions allow the programmer to vary parts of generated statements and the sequence of the statements. All of the conditional assembly instructions may be used anywhere in an Assembler language source program. The primary use of these instructions, however, is in macro definitions.

Macro instruction

Label field	Operation field	Operand field
	MOVE	FL1, FL2

Macro definition

```
         MACRO
&NAME    MOVE    &TO, &FROM
         LCLA    &A, &B, &C, &D
&A       SETA    10
&B       SETA    15
&C       SETA    &A-&B
&D       SETA    &A+&B
&NAME    ST      2, AREA
         L       2, &FROM&D
         ST      2, &TO&C
         L       2, AREA
         MEND
```

} Arithmetic expressions

Generated statements

```
         ST      2, AREA
         L       2, FL225
         ST      2, FL15
         L       2, AREA
```

Fig. 6.4 Example of SETA instruction

SET symbols

SET symbols are variable parameters which are assigned values by the SETA, SETB or SETC instructions. LCLA, LCLB and LCLC instructions may be used to define and assign initial values to SET symbols.

- SETA—the Set Arithmetic instruction is used to assign an arithmetic value to a SETA symbol.
- SETB—the Set Binary instruction is used to assign the binary values of 0 or 1 to a SETB symbol.
- SETC—the Set Character instruction is used to assign a character value to a SETC symbol.

Sequence symbols

Sequence symbols provide the programmer with the ability to vary the sequence in which statements are processed by the assembler. A **sequence symbol** is used in the operand field of an AIF or AGO statement.

- AIF—conditional branch instruction used to alter the sequence in which source statements are processed.
- AGO—unconditional branch instruction used to alter the sequence in which source statements are processed.
- ANOP—Assembly No Operation instruction facilitates conditional and unconditional branching to statements named by symbols or variable symbols.

Examples of conditional assembly instructions are shown in Figs. 6.4 to 6.7.

6.5 CONVERSATIONAL PROCESSING

Since the Assembler can operate under TSS (time sharing system) control, a user can generate, debug, and execute a program from the terminal.

The SYSTERM or TEST option specifies conversational processing to the Assembler. The SYSTERM option directs output of error messages to the SYSTERM data set. The TEST option permits debugging in a conversational mode. The user can trace the flow of his program by requesting displays of addresses and variable names in his TEST commands.

OS IV/F4 provides an Assembler Prompter to assist the user in conversational processing. When invoked by a terminal command, the Prompter helps the user define all necessary program elements, DD statements, etc. to assemble and execute his program successfully.

Macro instruction

Label field	Operation field	Operand field
	MOVE	FLA, FLB

Macro definition

```
            MACRO
&NAME       MOVE    &TO, &FROM
            LCLA    &A1
            LCLB    &B1, &B2
            LCLC    &C1
&B1         SETB    ('&TO' EQ 'FLA')
&B2         SETB    (L'&FROM EQ 10)
&A1         SETA    &B1
&C1         SETC    '&B2'
&NAME       ST      2, AREA
            L       2, &FROM&A1
            ST      2, &TO&C1
            MEND
```

Generated statements

```
            ST      2, AREA
            L       2, FLB1
            ST      2, FLA0
```

Fig. 6.5 Example of a SETB instruction

Macro instruction

Label field	Operation field	Operand field
	MOVE	

Macro definition

```
            MACRO
&NAME       MOVE    &TO, &FROM
            LCLC    &A
&A          SETC    'FL'
&NAME       ST      2, &A.A
&A          SETC    'FLE' 'DE' (1,3)
            L       2, &A
&A          SETC    'AFLEF' (2,3) 'D'
            ST      2, &A
            MEND
```

Generated statements

```
            ST      2, FLA
            L       2, FLEDE
            ST      2, FLED
```

Fig. 6.6 Example of a SETC instruction

Macro instruction

Label field	Operation field	Operand field
FLB	ADD1	A1, A2, A3

Macro definition

```
            MACRO
&NAME       ADD1    &A, &B, &C
            AIF     ('&A' EQ 'A0').END
            AIF     (T'&C EQ N).END
&NAME       ST      2, A0
            L       2, &A
            A       2, &B
            ST      2, &C
.END        MEND
```

Generated statements

```
FLB         ST      2, A0
            L       2, A1
            A       2, A2
            ST      2, A3
```

Fig. 6.7 Example of an AIF instruction

6.6 REQUIRED CONFIGURATION

The configuration of I/O functions (and associated DD names) required by the Assembler is shown in Fig. 6.8.

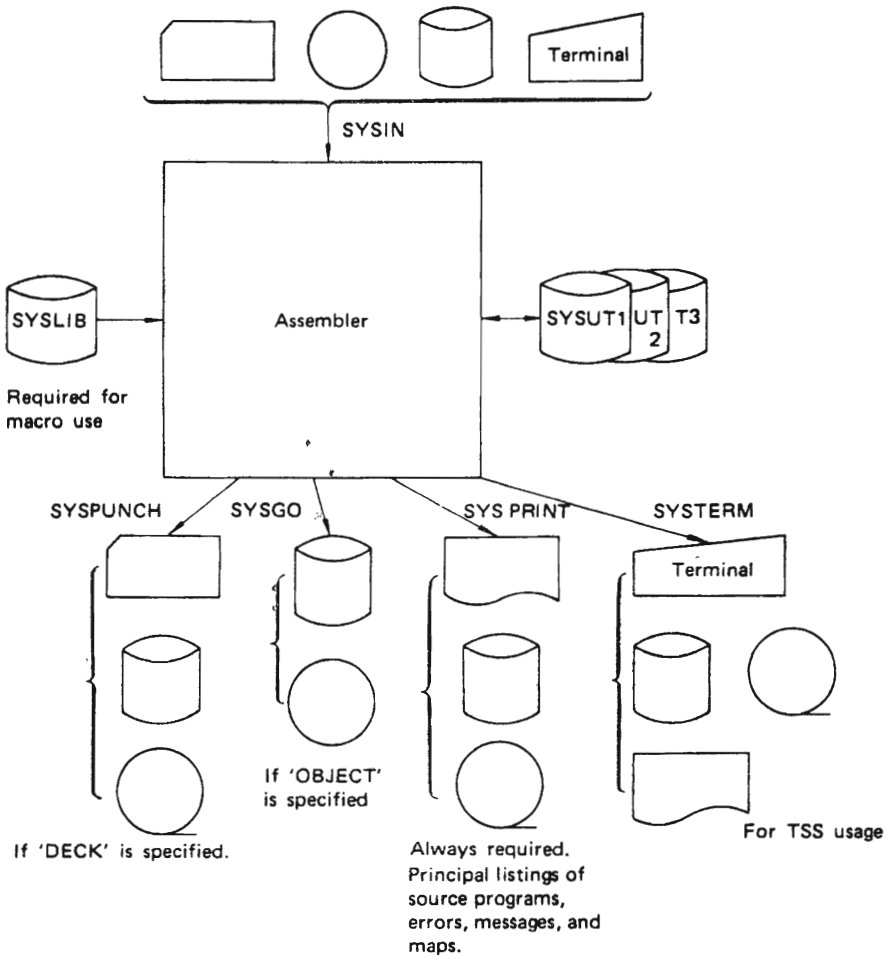


Fig. 6.8 Assembler unit configuration diagram

CHAPTER 7
SORT/MERGE

7.1 OUTLINE OF SORT/MERGE

The sort/merge program of OS IV/F4 arranges object data into a specified order (**sort** function) or merges several presorted data sets into one data set retaining the original order (**merge** function).

The sort/merge program provides the user with a generalized, efficient, and flexible program to handle his sorting and merging applications.

The sort/merge program is generalized in the sense that it can handle a wide range of sorting and merging applications. It can, for example, accept a wide variety of inputs:

- EBCDIC or ASCII data sets.
- Concatenated data sets having unlike characteristics or residing on unlike devices.
- Variable or fixed-length records.
- Data sets residing on punched cards, magnetic tape, and direct access storage devices.

The OS IV/F4 sort/merge program selects the most efficient sort or merge algorithm appropriate to any collection of input, output, and work data sets. Specifically, it will:

- Automatically choose the most efficient sorting or merging method for a given input-output-intermediate storage configuration and set of file characteristics.
- Determine, at execution time, the maximum amount of real storage in the system that can be made available to it, and use it.
- Support as intermediate work devices the F478B and F479B disk storage devices, the F6625A drum storage device, and the F610 series and F611 series magnetic tape units.

Sort/merge is flexible in that it can:

- Be installed to include all of its functional capabilities, or only those capabilities needed by the user's installation.
- Perform a sort operation on as many as 64 control fields, arranging the data in either ascending or descending order.
- Perform an intermediate merge operation on as many as 16 previously-sorted data sets.
- Include at execution time as many as 18 different types of exit routines. (This makes it possible for the user to use his own I/O error-handling routines.) Using these specially written routines the user can also perform common sort-related tasks such as generating summary records, deleting and inserting records, and general file maintenance.

Fig. 7.1 shows the sort unit configuration and Fig. 7.2 shows the merge unit configuration. Data definition names (DDNAMES) for sort/merge are as follows:

- SORTIN — input file(s) to be sorted or merged.
- SORTOUT — output file of sorted or merged records.
- SORTLIB — library of sort/merge program modules.
- SORTMODS — data sets furnishing any modules required by the exit routines.
- SYSIN — program control statements.
- SORTWKnn — work area(s) used by the sort program.
- SORTCKPT (optional) — data set for checkpoint records.
- SYSOUT — messages produced by Sort/Merge.

7.1.1 Sort/Merge Phases

The OS IV/F4 sort/merge program consists of an initialization phase, an internal sort phase, an intermediate merge phase and a final merge phase. The flow of sort/merge is shown in Fig. 7.3.

Initialization phase

In this phase OS IV/F4 checks the SYSIN control statements for sort/merge, determines the sort technique, calculates program area and work area, and checks the SORTIN and SORTOUT units used for sort/merge.

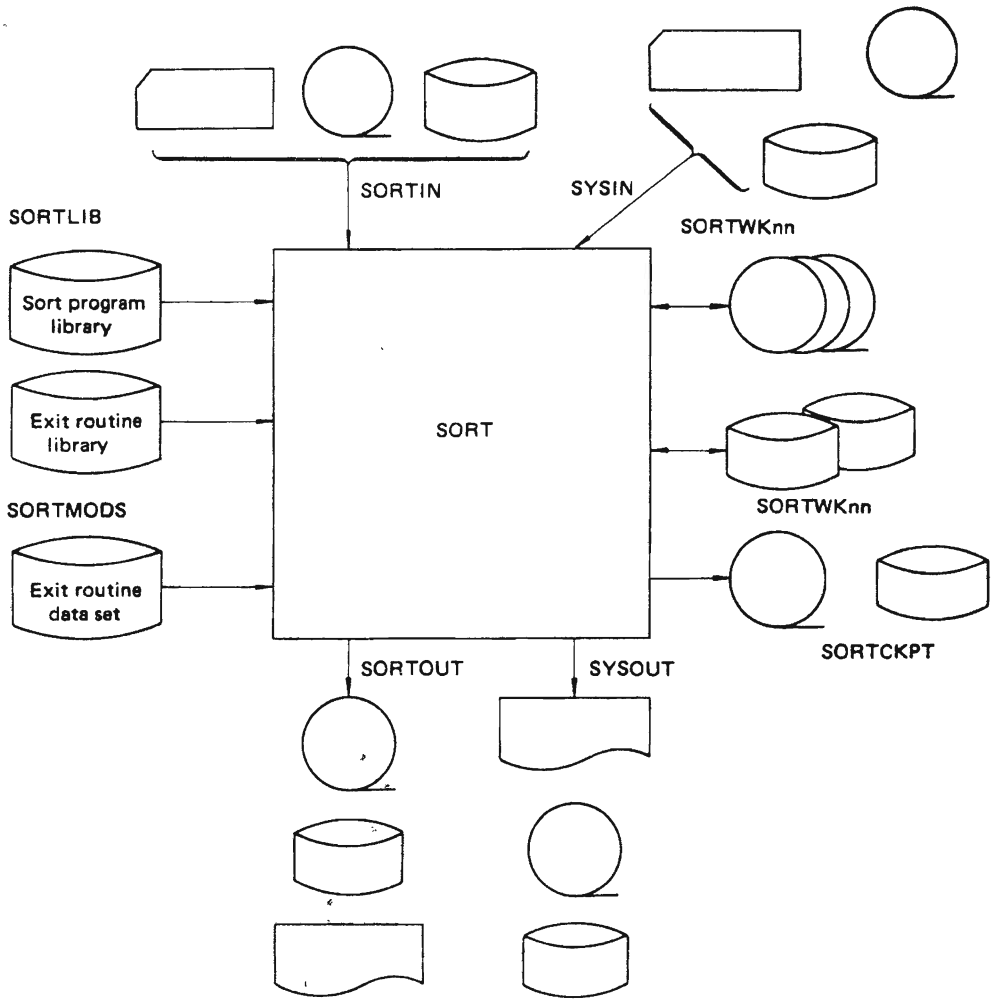


Fig. 7.1 Sort unit configuration diagram

Internal sort phase
In this phase, input data are internally sorted and written onto the SORTWKnn data sets. Data at one or more specified positions (**control fields**) in the records are compared, rearranged, and written as sequences of partially sorted records (**strings**), in either ascending or descending order. When all input data have been processed in this way, control passes to the next phase.

Intermediate merge phase
Strings generated in the internal sort phase are read back from the SORTWKnn data sets, merged into longer strings, and written back onto the work data sets. This operation is continued until the number of strings reaches the target number for the final merge phase. If the number of strings generated in the internal sort phase is already less than the target number of strings for the final merge phase, the intermediate merge phase is skipped.

Final merge phase
Fully-sorted (or fully-merged) data are written onto the SORTOUT unit(s).

7.1.2 Sort Processing Flow

When the user requests sorting, the four phases are normally executed in the indicated sequence. Under some conditions (e.g., if input data sets are substantially presorted), the intermediate merge phase may not be needed.

7.1.3 Merge Processing Flow

When the user requests merging, only the initialization and final merge phases are performed. Input data must have been presorted into the required sequence.

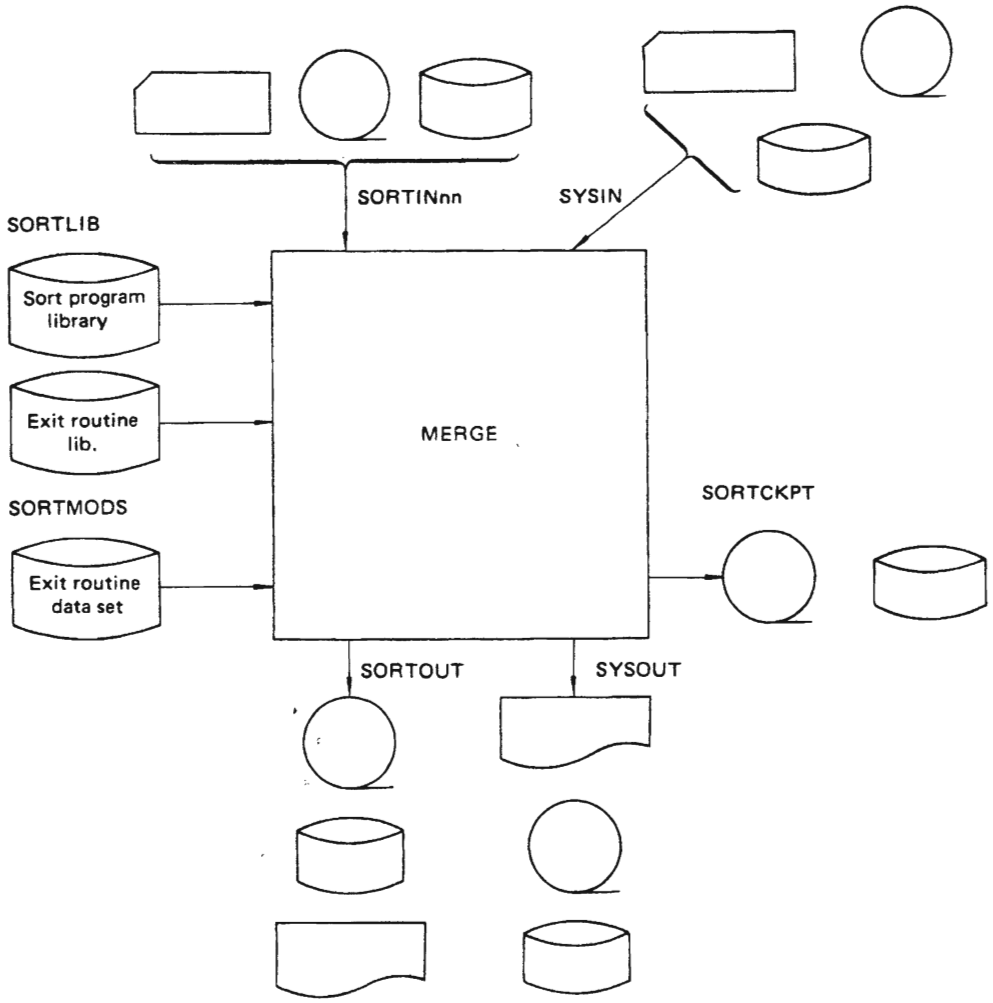


Fig. 7.2 Merge unit configuration diagram

7.2 FUNCTION OF SORT/MERGE

7.2.1 Control Field Comparison Method

A maximum of 64 control fields can be specified. Each control field must be within 4092 bytes from the beginning of each record, and the sum of the lengths of all control fields must not exceed 4092 bytes.

The control functions are as follows:

- Order specification
  The order of arranging the records, either ascending or descending, can be specified for each control field.
- Control field comparison
  The format of each control field is specified on the Sort/Merge control statement.
- Change of control field collating order
  When the characters of the control field are compared, the user can request a collating sequence other than EBCDIC. This is accomplished by using the ALTSEQ command on the sort/merge control statement. Even if this function is used, the value of the control field is not changed for output.
- Processing control fields of identical value
  When records whose control fields have identical values are sorted, the records are usually output without preserving the input sort order. However, the input sort order can be retained in output records having equal control fields by specifying a parameter on the sort control statement.

Table 7.1 Organization of sort/merge data sets

Characteristics \ Data set	SORTIN	SORTOUT	SORTWKnn
Unit	CR, MT, DASD*	LP, MT, DASD*	MT, DASD**
Organization	Sequential organization VSAM data set	Sequential organization VSAM data set	Sequential organization
Record format	Fixed or variable length	Fixed or variable length	—

* Any DASD unit supported by QSAM and VSAM can be used as the SORTIN or SORTOUT unit.

** Mixed use of MT and DASD is impossible. For DASD, the SORTWKnn units must be the same type and the work area on each DASD must be contiguous (i.e., a single DASD extent).

7.2.2 Input/Output Data Sets and Work Data Sets

Table 7.1 shows characteristics of the SORTIN, SORTOUT, and SORTWKnn data sets. The following sections describe block lengths and record lengths permitted for the SORTIN and SORTOUT data sets.

Block Length
When several SORTIN data sets are concatenated, their block lengths may differ.

- During merging, block lengths of SORTIN data sets may differ.
- For both sorting and merging, block lengths of the SORTIN and SORTOUT data sets may differ.
- Any DASD unit supported by QSAM and VSAM can be used at the SORTIN or SORTOUT unit.
- Mixed use of MT and DASD is impossible. For DASD, the SORTWKnn units must be the same type and the work area on each DASD must be contiguous (i.e., a single DASD extent).

Record length
During the execution of sort/merge, the user can change the record length by using an exit routine. The length is specified through the LENGTH parameter of the RECORD control statement.

7.2.3 User Exit Routines

The user can optionally perform special processing during Sort/merge by using exit routines written in assembler language. All exit routines are linked to the Sort/Merge program by the MODS control statement.

Data set opening and initialization exit routines
The user can open any/all data sets and perform other initializations by providing the following exit routines:

E11 ..... Internal sort phase
E21 ..... Intermediate merge phase
E31 ..... Final merge phase

Data set close exit routines
These routines can be used to close any/all data sets in the indicated phase(s).

E17 ..... Internal sort phase
E27 ..... Intermediate merge phase
E37 ..... Final merge phase

Record change exit routines
These routines can be used to add or delete input records and to change their contents.

E15 ..... Internal sort phase
E25 ..... Intermediate merge phase
E35 ..... Final merge phase
E32 ..... In the case of merge processing

Error processing exit routines for record input/output
These routines can be used to incorporate an error recovery routine into the Sort/merge program for each user or for an entire computation center. This exit routine is automatically executed when an uncorrectable input/output hardware error is encountered.

During input

E18 ..... Internal sort phase
E28 ..... Intermediate merge phase
E38 ..... Final merge phase

During output

E19 ..... Internal sort phase
E29 ..... Intermediate merge phase
E39 ..... Final merge phase

Control field change exit routine
This routine (E61) lengthens, shortens, or alters any control field within a record. The control-field-change option must be specified on the Sort/Merge control statement for each control field to be changed.

Work data set overflow exit routine
This routine (E16) specifies the procedure to be taken by the Sort/merge program when available main storage for any work data set is insufficient.

7.3 SORT/MERGE TECHNIQUE

The Sort/merge program selects the sort technique for the internal sort phase and intermediate merge phase which uses the least CPU and I/O time.

Internal sort technique

The **internal sort technique** is called **tournament replacement selection technique.** This technique simultaneously compares as many input records as will fit in main storage and outputs as long a string as possible.

Intermediate merge technique

The **intermediate merge technique** depends on the type of external storage (DASDs or magnetic tape units) used for the work data sets.

When work data sets are on DASDs, the **balanced merge technique** is used, which alternates main storage processing with input/output and distributes the output strings across as many tracks of the work data set as possible thereby lessening the head motion.

When three or more magnetic tape units are used for the work data sets, the **read backward polyphase merge technique** is applied. After the internal sort phase has distributed strings to all work units but one, the reels on these units are read backwards; longer strings are then written onto the last work unit.

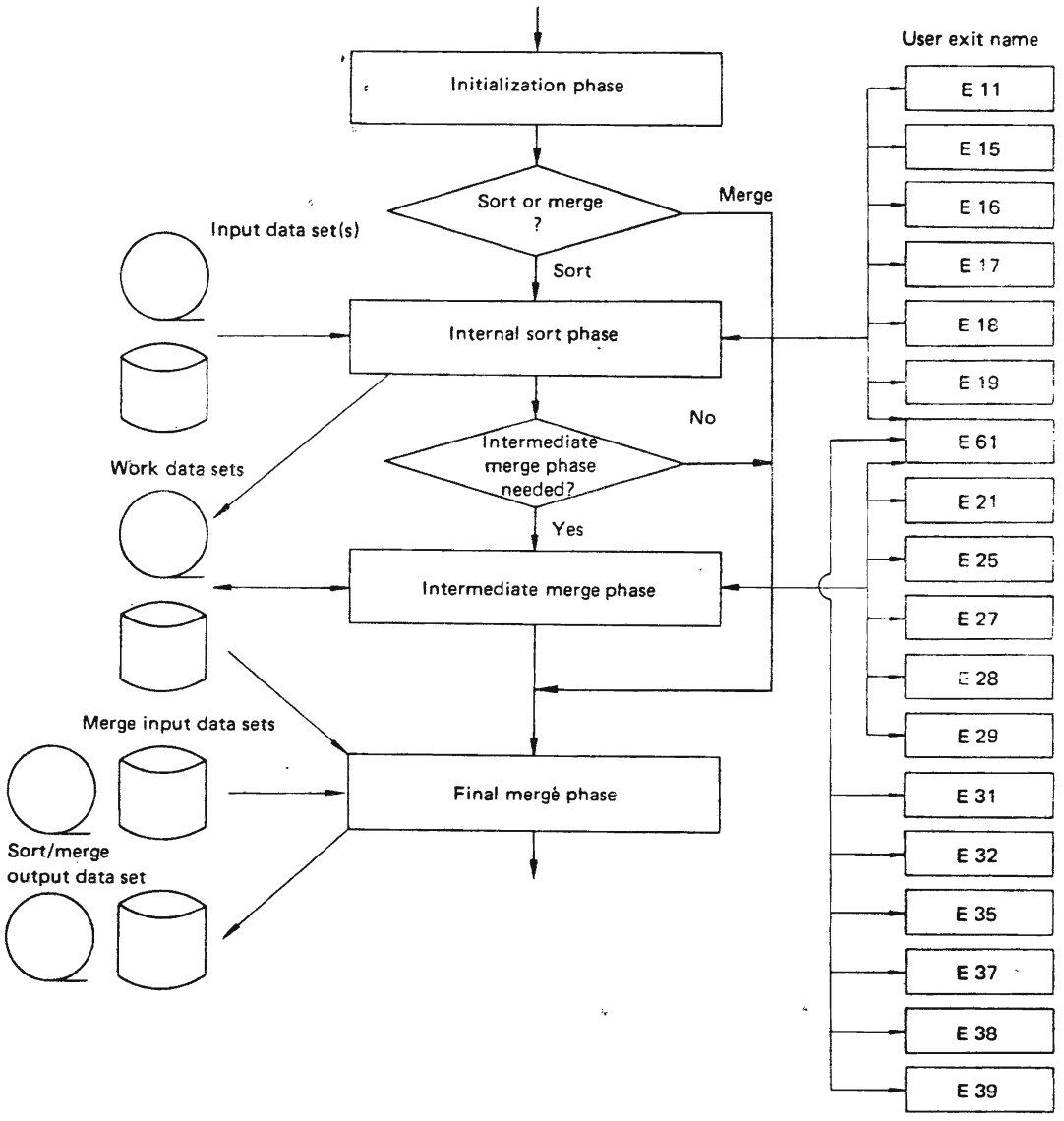


Fig. 7.3 Flow-diagram of sort/merge

CHAPTER 8
LINKAGE EDITOR/LOADER

8.1 OUTLINE OF THE LINKAGE EDITOR/LOADER

The linkage editor and the loader are two OS IV/F4 processing programs for preparing the outputs from language translators for execution. The linkage editor prepares a load module, which a separate job step can bring into main storage for execution. The loader prepares an executable program in main storage and passes control to it directly during the same job step.

The linkage editor provides several additional processing facilities for creating overlay programs, selectively replacing program sections, and building and editing system libraries. The loader provides fast loading of programs that need not be stored for production use or do not require the special processing facilities of the linkage editor. One invocation of the Loader is comparable to linkage editing and immediate execution of a program.

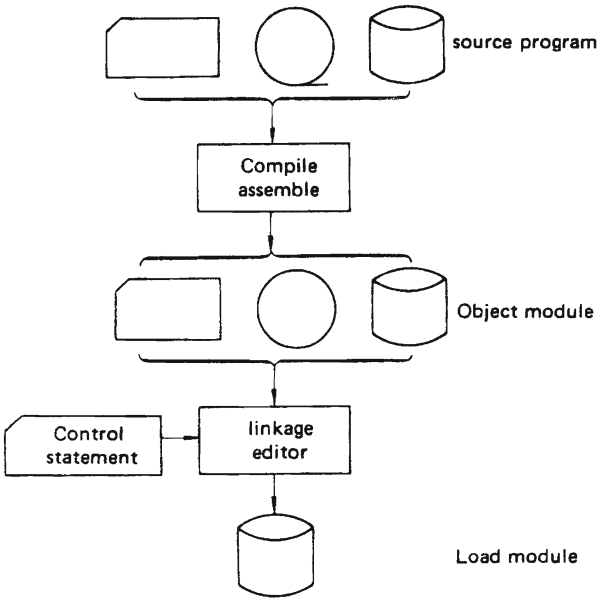


Fig. 8.1 Source, object, and load modules

8.2 FUNCTIONS OF THE LINKAGE EDITOR

8.2.1 Combining Object Modules and Load Modules

The user compiles object modules with an assembler and/or compiler. He creates an executable program by furnishing these object modules to the linkage editor or loader, which links them together by resolving references to external symbols.

Linking Modules
Input modules (object modules and load modules) specified by linkage editor control statements are linked into one load module, as shown in Fig. 8.1.

Editing load modules
If the user wishes to change an executable program, he compiles (or assembles) and edits only object modules which need updating rather than his entire source program. Specified modules can be replaced, renamed, deleted, or reordered via linkage editor control statements as shown in Fig. 8.2.

Automatic call function
To resolve external symbol references, the optional automatic call function can retrieve load modules from a specified library (the **automatic call library**) in order to resolve them. Data sets specified by INCLUDE or LIBRARY control statements (except any libraries named on the SYSLIB DD statement) can be used as automatic call libraries. The following functions can be used to specify external references that are not to be resolved by the automatic call function:

- Restricted nocall function (LIBRARY control statement parameter)
  The specified external symbol references are not to be resolved during the current linkage editor step.

- Never-call function (LIBRARY control statement parameter)
  The specified external symbol references are not to be resolved during any linkage editor step.
- NCAL option (EXEC statement parameter)
  All external symbol references are not to be resolved during the current job step.

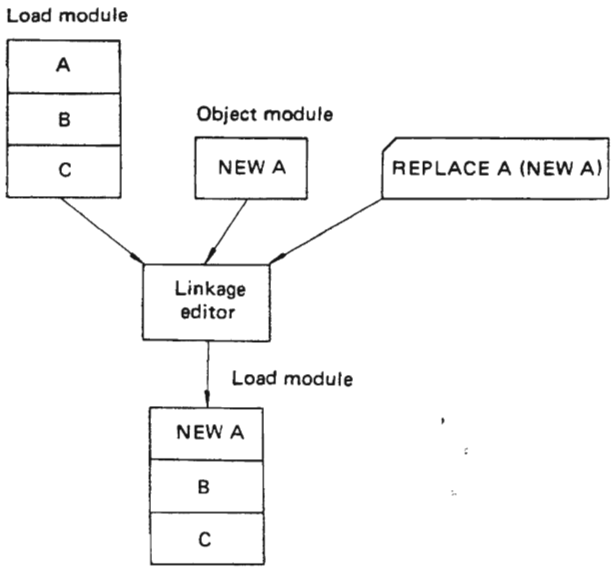


Fig. 8.2 Replacing a control section in a load module

8.2.2 Address Allocation

Allocating addresses comprises the following four functions: aligning page boundaries, reserving COMMON and static external storage areas, processing of pseudo-registers, and processing of prototype control sections.

Page boundary alignment for control sections and labeled common areas
Control sections and labeled common areas of the load module are aligned to a 4K-byte page boundary. Careful alignment of page boundaries can reduce paging activity by OS IV/F4 virtual storage architecture.

Reservation of specialized storage areas
This function processes the unlabeled common control section (COMMON) generated by FORTRAN and assembler-language processors. It also processes any static external storage areas generated by a PL/I compiler. The linkage editor reserves this common area in the output module.

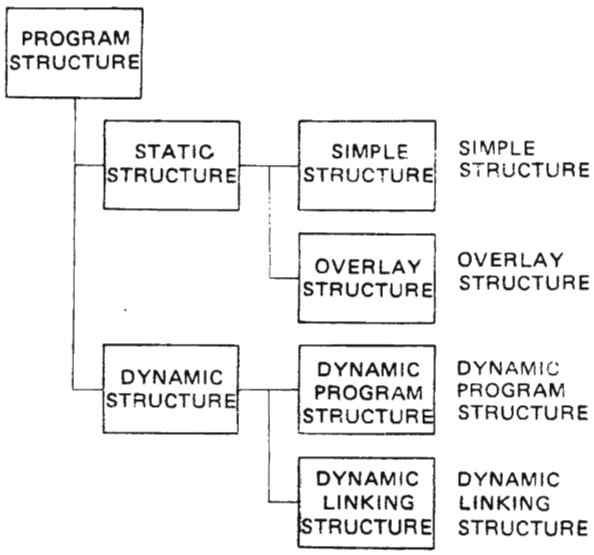
Pseudoregister processing
Pseudoregisters are useful for coding reentrant programs. The storage area for these registers must be secured dynamically during execution. The linkage editor processes the pseudoregisters by calculating the amout of storage required for each pseudoregister, the displacement of each pseudoregister from the beginning of the dynamic storage area, and the total storage area required for all pseudoregisters.

Processing prototype control section
A prototype control section (PSECT) may be requested for a reentrant program containing data to be changed, work data, and address constants. When a prototype control section is furnished for a reentrant load module, the linkage editor links it with any other prototype control sections. If the reentrant attribute is not specified, PSECTs are processed in the same way as ordinary control sections (CSECTs).

Each time the load module is attached or linked, the OS IV/F4 Supervisor copies the PSECT into a Supervisor-allocated work area.

8.2.3 Program Structure Processing

The program structures permitted by OS IV/F4 are the following:



The principal difference between a **static structure** and a **dynamic structure** is the time at which the modules required for execution of the program are linked. A static structure has all required control sections linked into a single load module. A dynamic structure contains two or more load modules. At execution time, the OS IV/F4 control program loads the proper program into the user's virtual storage area and passes control to it. The conceptual flow of control for each structure is shown in Fig. 8.3.

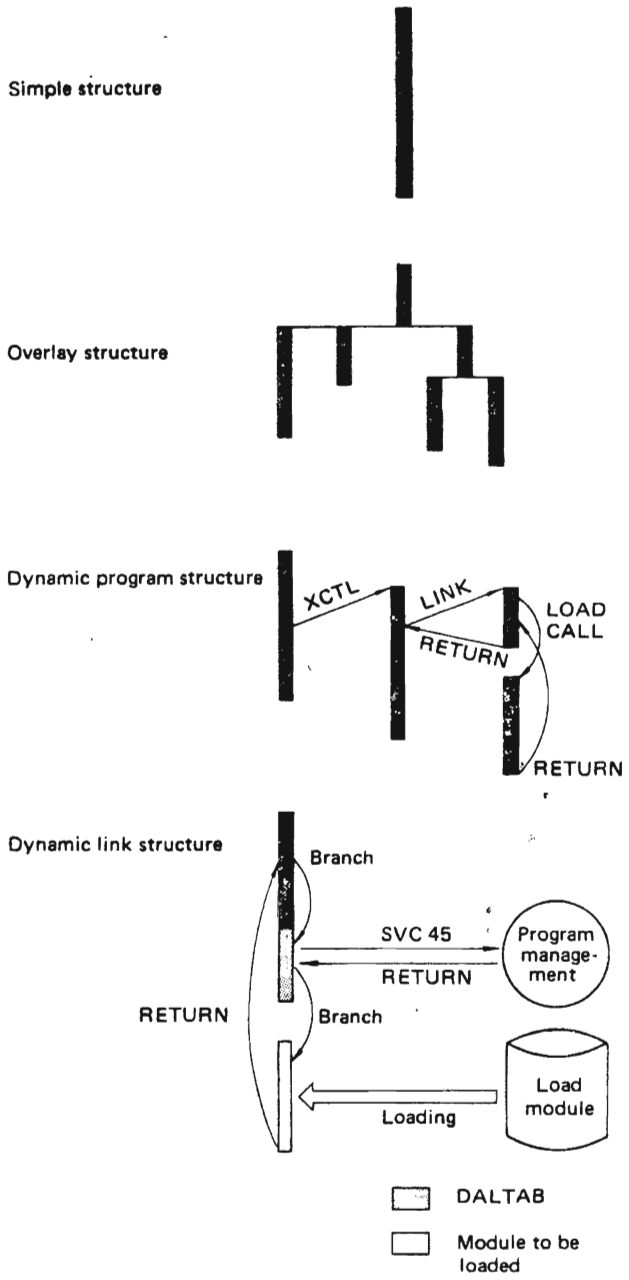


Fig. 8.3 Conceptual figure of the program structure

Simple structure
All programs for one job step are loaded into virtual storage at one time and immediately executed.

Overlay structure
An overlay structure consists of several segments which overlay one another upon request by the user program.

Dynamic program structure
A dynamic program structure retrieves other load modules depending on execution flow. These modules are loaded – and control is passed to them – via various supervisory macro instructions: ATTACH, LINK, XCTL, LOAD, CALL and RETURN.

Dynamic link structure
This structure dynamically links to load modules managed by the OS IV/F4 control program as well as those specified explicitly by the user in his address space. A dynamic linking table (DALTAB) is generated by the linkage editor to resolve external symbols possibly required during each execution of the program. Control is passed to each module only when actually required by the executing program.

In a dynamic program structure, the user furnishes program-control macro instructions and the linkage editor need not perform special processing. The static and dynamic link structures differ in how they resolve external symbol references. The linkage editor can process either structure, as requested by the user.

8.3 TIME-SHARING CONSIDERATIONS

So that programs can be developed, debugged, and executed under TSS control (for details see **Part 4**, "Time Sharing System"), not only compilers and the assembler but also the linkage editor and loader are available under TSS.

Table 8.1 Allowable program structures

Program \ Structure	Simple structure	Overlay structure	Dynamic program structure	Dynamic link structure
Linkage editor	X	X	X	X
Loader	X		X*	X

* (Some restrictions)

Table 8.2 Usage attributes

Program \ Attribute	Refreshable	Reenterable	Serially reusable	Nonreusable
Linkage editor	X	X	X	X
Loader				X

TERM option
The PARM parameter of the EXEC statement designates the output unit for error and warning messages from the linkage editor and loader. Output is to the unit (terminal, line printer, DASD, or magnetic tape) specified by the SYSTERM DD statement.

Prompter
Prompters are provided for the linkage editor and loader, called directly or indirectly through a TSS terminal command to allocate data sets required by the linkage editor or loader, to invoke the linkage editor or loader, and when processing is completed, to release these data sets.

8.4 REQUIRED UNIT CONFIGURATION

The unit configurations used by the linkage editor and loader are shown in Figs. 8.4 and 8.5, respectively.

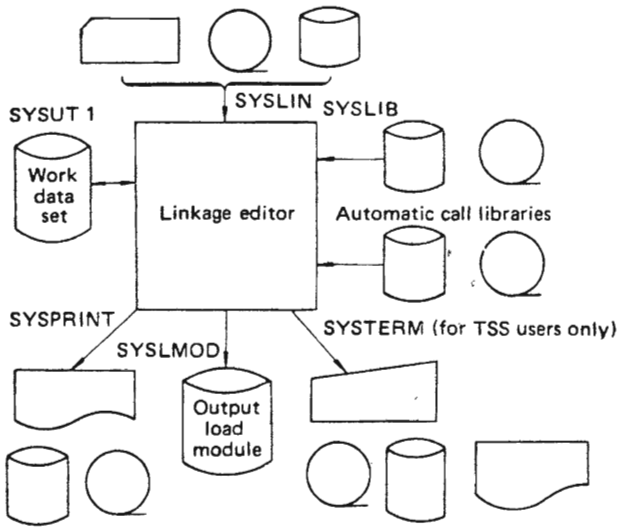


Fig. 8.4 Linkage editor unit configuration diagram

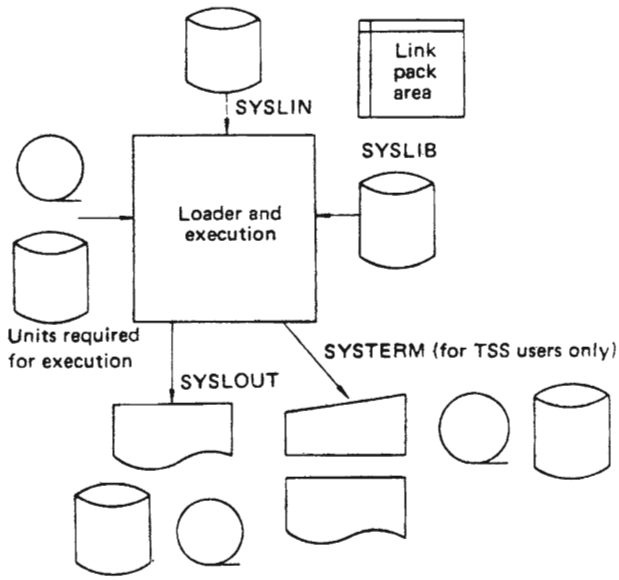


Fig. 8.5 Loader unit configuration diagram

CHAPTER 9
UTILITY PROGRAMS

9.1 OVERVIEW

OS IV/F4 offers system utility, data set utility and independent utility programs. **System utility programs** are used to maintain system control information – volume tables of contents (VTOCs), catalogs (SYSCATLG), PASSWORD data sets, etc. **Data set utility programs** perform various transcription and validation operations on sequential, partitioned, or direct access data sets. They reorganize, change, or compare records at the data set and/or record level. Most data set utility programs offer various user exit facilities to permit each installation to customize its operation. System and data set utility programs can be invoked by other application programs.

9.2 SYSTEM UTILITY PROGRAMS

9.2.1 Alternate Track Assignment and Recovery–JSGATLAS

This utility program allocates an alternate track as a logical replacement for a hardware-defective DASD track. Salvageable data records on the defective track are moved to the alternate track, after analysis of any records which are difficult or impossible to read.

9.2.2 DASD Initialize, Dump, or Restore – JSGDASDR

The functions of this utility program are to initialize a DASD volume, assign one or more alternate tracks, and save or restore contents of a DASD volume, specifically:

- To write a volume label and to create a volume table of contents on a new or reconditioned DASD volume.
- To perform thorough surface testing for a DASD volume, verifying its capability to read/write data from each track without hardware errors.
- To change the serial number of a DASD volume.
- To allocate an alternate track as a replacement for a defective track.
- To write part or all of the contents of a DASD volume onto another volume.
- To write ("dump") a DASD volume onto magnetic tape, or to restore the tape contents back onto a DASD volume.
- To write a diagnostic image of one or more DASD tracks onto the system output unit.

9.2.3 Initialize Magnetic Tape Reels–JSGINITT

This utility program can initialize one or more magnetic tape reels. Volume and header labels are created; either ASCII or EBCDIC can be selected as the character code.

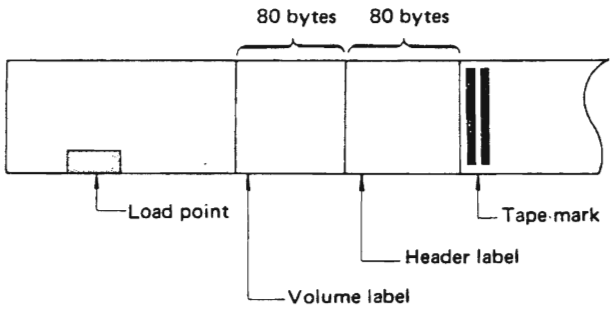


Fig. 9.1 Tape format after initialization

9.2.4 List Data Sets or Control Information – JSGLIST

This utility program can perform the following services:

- Display the contents of a system catalog (SYSCATLG data set).
- Display a partitioned data set directory.
- Display selected information from a DASD volume table of contents. The information can describe the entire volume or a selected data set.

9.2.5 Edit and Print SMF Statistics—JSGSTATR

This utility program edits and prints certain information collected by the OS IV/F4 system management facilities.

9.2.6 Move or Copy a Data Set—JSGMOVE

This utility program can copy or move a sequential, partitioned, or direct access data set. The source data set is usually deleted when processing has been completed—unlike the JSECOPY utility program—so the user must invoke JSGMOVE carefully so a useful data set is not prematurely deleted.

- To copy or move an arbitrary data set.
- To copy or move a group of cataloged data sets.
- To copy or move part or all of a system catalog (SYSCATLG data set).
- To include or exclude the specified data sets (or members of a partitioned data set) while copying or moving the contents of a DASD volume.
- To unload a partitioned data set onto a magnetic tape as a sequential data set, i.e., to back up the data set. Likewise, the program can reload the magnetic tape copy back onto a DASD volume in partitioned format.

9.2.7 Program for VTOC and Catalog Management—JSGPROGM

This utility program has the following functions:

- To delete data sets or partitioned data set (PDS) members.
- To change the names of data sets or PDS members.
- To catalog or uncatalog data sets.
- To create or delete indices or index names from the system catalog.
- To link or delink portions of the system catalog.
- To create or maintain the indices for generation data groups.
- To create or change the passwords for the DASD volume.

9.3 DATA SET UTILITY PROGRAMS

9.3.1 Compare Two Data Sets—JSDCOMPR

This utility program can compare pairs of records from two sequential data sets or two partitioned data sets. Sequential data sets are considered identical if they contain the same number of records and corresponding records and keys are identical. Two partioned data sets are considered identical if they contain the same number of records in corresponding members, contain note lists in the same positions of corresponding members, and have identical records and keys.

9.3.2 Copy a Data Set—JSECOPY

This utility program can copy one or more partitioned data sets:

- To copy, delete, or replace specified members from the target data set.
- To change names of selected members.
- To copy all members—or specified members—onto magnetic tape in order to back up the data set, and vice versa.
- To compress and reorganize a partitioned data set, in order to reclaim unusable DASD space.

9.3.3 Generate a Test Data Set—JSDDG

This utility program creates data sets for program debugging. Test data are generated by specifying output record fields and desired characteristics or by selecting given fields from input records for the output records.

9.3.4 Edit a Job Stream—JSEEDIT

This utility program copies and edits data sets by selecting them from a job stream; it creates a new input job stream.

9.3.5 Generate a New Data Set—JSDGENER

This utility program generates one or more sequential data sets or members of a partitioned data set. It can also create a partitioned data set from a sequential data set.

9.3.6 Print or Punch a Data Set—JSDPTPCH

This utility program has the following functions:

- To print or punch a sequential data set or an entire partioned data set.
- To print or punch specified members of a partitioned data set.
- To print or punch specified records from a sequential or partitioned data set.
- To print or punch the directory from a partitioned data set.

9.3.7 Update a Source Library—JSEUPDTE

This utility program has the following functions:

- To create a source-program library.

- To update a source program stored in a sequential or partitioned data set.
- To change a data set organization (from partitioned to sequential, or vice versa).

PART 4
TSS
(TIME SHARING SYSTEM)

CHAPTER 1
OUTLINE OF TSS

The time sharing system (TSS) of OS IV/F4 allows the user to make use of a computer from a terminal. A terminal is a typewriter-like device connected through telephone or other communication lines to a computer. A terminal can be any distance from the computer — in the same room or in another city. Because OS IV/F4 processes instructions much faster than the user can enter them through the terminal, it can process input from many remote terminals at the same time it is processing jobs entered locally at the computer center. However, due to the speed of the system, the user will be able to work almost as though he had exclusive use of the computer.

The user can tell the system what work to perform by typing in one or more commands in the TSS command language. The command language can be used to:

- enter, store, modify, or retrieve data at the terminal.
- solve mathematical problems.
- develop programs in Assembler, FORTRAN, COBOL, PL/I, or other languages.
- execute programs.

When the user enters a command into the system, the system performs the work requested by that command and sends messages back to the user's terminal. The messages tell him the status of his program and whether the system is ready to accept another command. He can interrupt the processing of a command at any time to enter a new command.

If the user makes a mistake typing in a command, or if he fails to include some necessary information with the command, the system sends him a message prompting him for correct input. He may then respond by typing in this information. Table 1.1 lists the terminals supported by TSS.

1.1 TSS DESIGN

TSS has been designed with the following characteristics:

- ease of input.
- efficiency.
- high reliability.

To accomplish these goals, the following considerations have been given to the system:

- A variety of commands allow the remote user to use most OS IV/F4 capabilities.
- TSS provides the user with both diagnostic and informative messages.
- OS IV/F4 batch environment is compatible with TSS. Therefore, programs developed under TSS can be executed in batch mode and batch-developed programs can be run under TSS.
- The system automatically schedules jobs by means of its optimization algorithm.
- The modular structure of the control program confines system failures to one module.
- The secrecy of the user's programs and data is assured.

For additional details the reader should consult the **FACOM OS IV/F4 TSS General Description** and other related **FACOM OS IV/F4 TSS** Manuals.

Table 1.1 Terminals supported by TSS

Terminal *	TY	LP	PT	CRT
Fujitsu F1510	X		X	
F1520	X	X	X	
F1530	X			
F9520	X	X		X
F9525	X			X

TY Typewriter I/O
LP Line printer
PT Paper tape read/punch
CRT Display terminal
* Other terminals in common use as TSS terminals may also be supported in the future provided they meet certain common standards for control and interface.

1.2 TSS FEATURES

Simultaneous operation of TSS and batch
TSS and batch jobs operate simultaneously in the same system. Allocation of CPU control among batch and TSS jobs is easily controlled. Since this allocation is a function of the number of concurrent TSS and batch jobs, the optimum distribution is achieved at all times.

Multiprocessor support
TSS jobs can be executed on either a uniprocessor or multiprocessor configuration,

Multiple virtual storage support
Each terminal user is given a virtual address space of 16 million bytes (16 MB). Since the user is not restricted by real-storage region sizes, he can readily execute most programs.

Number of concurrent jobs
The maximum number of concurrent TSS and batch jobs is essentially infinite (approximately 1,500).

Number of users
The number of authorized TSS users for one installation is unlimited.

Communication access
Virtual Telecommunications Access Method (VTAM) controls communication between the terminal user and TSS. A communication access management routine processes incoming messages. The configuration of each remote terminal is defined for VTAM when TSS is installed at a center.

Job swapping
If an executing job must await terminal input/output, it is swapped out to external storage. The job is brought back into the system when the input/output is complete. Hence, a greater number of users can use real storage and other system resources.

Response times
OS IV/F4 allocates CPU time according to an algorithm which assures adequate response time for TSS jobs. The standard algorithm can be altered for each installation.

Compatibility with batch
The environments of TSS and batch are **mutually compatible** so that it is possible to execute TSS-developed programs in the batch environment and batch-developed programs by TSS. A data set generated by batch may be accessed by TSS and a TSS-registered data set may be used in the batch environment.

Service routines
The service routines contain control functions required by all terminal users. The I/O service routine dynamically controls the I/O device buffers. The parse/scan service routine checks the validity of input commands and their syntax.

Debug aids
Interactive debug commands allow the user to debug his programs efficiently by setting breakpoints or displaying the contents of registers and storage areas. A **breakpoint** is either a source statement to be executed or a condition to be met which will suspend program execution and return control to the user's terminal. A program may be executed from one breakpoint to any other breakpoint or from the beginning to any breakpoint.

Prompters
Prompters provide a conversational method of defining jobs for compilers or the linkage editor/loader. The prompter accepts input commands from the user, analyzes the input for correctness and completeness, prompts the user to input any missing information, allocates the necessary data sets, and invokes the proper processor. The terminal user can activate the following prompters by command:
- FORTRAN.
- COBOL.
- PL/I.
- Assembler.
- Linkage editor.
- Loader.

Dynamic allocation
The user can allocated data sets such as system libraries, user libraries and other data sets required by his jobs, then release them when they are no longer needed. This **dynamic allocation** should be employed for data sets which cannot be allocated until job execution.

Conversational remote job entry (CRJE)
CRJE allows the user to execute programs developed under TSS in the batch environment. Users may generate, update and debug programs under TSS and then submit the completed programs to the OS IV/F4 job initiators for execution.

User protection
Since each user has an independent virtual space he can neither destroy nor read from another user's address space. The system automatically applies the user's registered name and password to his data sets so they are protected from any illegal access.

Failure localization
Since each user has an independent space, he can not be affected by a failure in another user's job.

User control of the system
Any authorized user can modify a system parameter

(e.g., permission to use the OPERATOR or ACCOUNT commands). The user can also monitor system operation from his terminal.

TSS Command Language
Table 1-2 lists the types of TSS commands. Additional details are found in the **FACOM OS IV/F4 TSS Command Reference Manual** and the **FACOM OS IV/F4 TSS Command Processor Generation Guide.**

TSS languages
The TSS language processors are as follows:
- ANS COBOL.
- FORTRAN IV GE.
- FORTRAN IV HE.
- PL/I.
- Assembler.

Table 1.2 Types of TSS commands

Purpose	Function	Command
System control	Defines installation or user control information	OPERATOR ACCOUNT
Session control	Requests the system to initiate or terminate a session.	LOGON LOGOFF
Program development data operation	Creates or updates data or source programs.	EDIT MERGE
Program operation	Controls translation, linkage and execution of programs.	CALL, RUN LINK
Data set management	Allocates data sets, releases or modifies data set names, prints data sets, and deletes data sets.	ALLOCATE LIST DELETE
Debug aid	Allows conversational debugging of programs.	TEST
Terminal control	Defines terminal characteristics, such as the attention, character erase and line erase characters.	TERMINAL PROFILE
Conversational remote batch	Submits TSS-generated programs to the OS OS IV/F4 job initiators.	SUBMIT OUTPUT
Connect time	Requests terminal connect time.	TIME
Message transmission	Sends messages to the console operator or to other users.	SEND
Command description	Request the function, syntax, and operands for any TSS command.	HELP

1.3 SYSTEM CONFIGURATION

The hardware and software configurations required for TSS are described below. For additional information, the reader should consult the **FACOM OS IV/F4 TSS General Description.**

1.3.1 Hardware Configuration

The following devices are needed for TSS operation in addition to the standard configuration for batch processing:
- Terminal devices
  Typewriter devices used as TSS terminals are the F1510, F1591, F1592 typewriter terminals and the F9525 display devices.
- Communication control processor (CCP)
  A F2805 or F2803 communication control processor is a required interface between the terminal communication lines and the channel processoc.
- Direct access storage devices
  DASDs for holding the command library and system data sets are the F6625 magnetic drum unit and the F478B/F479B magnetic disk pack unit.

Fig. 1.1 illustrates a representative hardware configuration for TSS.

1.3.2 Software Configuration

The following elements make up the TSS software system:
- Virtual Timesharing Access Method (VTAM).
- Time Sharing Control Task (TSCT).
- Address Space Control Task (ASCT).
- Session Control Task (SCT).
- TSS Message Control Program (TSSMCP).
- Terminal Monitor Program (TMP).
- Command Processor (CP).
- Service routines.
- Language processors.
- Syntax Checkers (FORTRAN, PL/I).
- Prompters (FORTRAN, COBOL, PL/I, Assembler, Linkage Editor, Loader).
- Interactive Debug (FORTRAN, COBOL).

Fig. 1.2 shows the configuration of the TSS control program.

The main functions of each program in the software system are as follows:

Time sharing control task (TSCT)
TSCT performs overall control of TSS. It initializes TSS processing, generates space for users, modifies the job-scheduling algorithm, and terminates TSS processing.

Address space control task (ASCT)
ACST controls swap-out and swap-in of jobs and processes attention interrupts from user terminals.

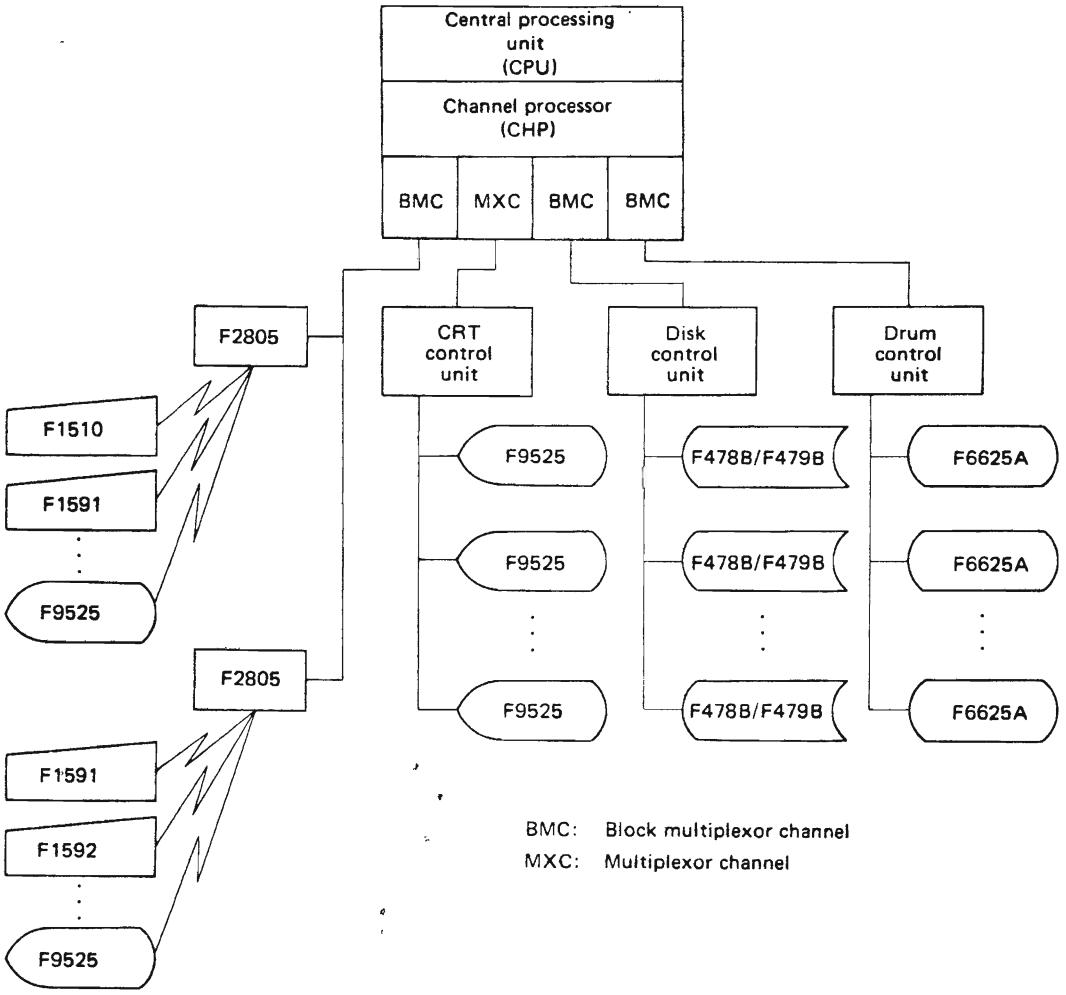


Fig. 1.1 Representative hardware configuration for time sharing

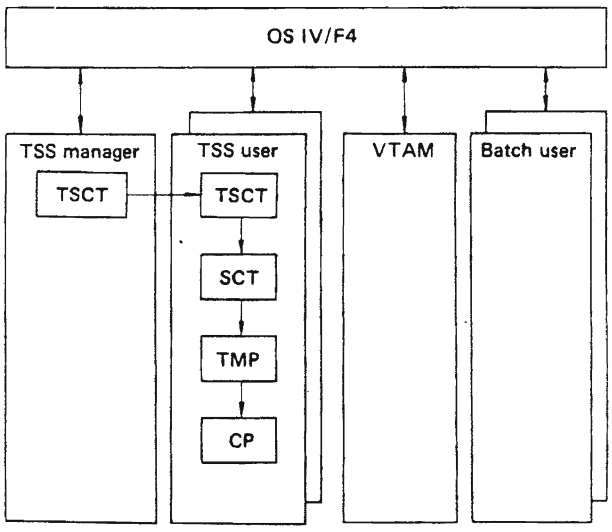


Fig. 1.2 Configuration of TSS control program

Session control task (SCT)
SCT allocates resources when a terminal session is initiated and releases the resources when the session is terminated.

TSS message control program (TSSMCP)
TSSMCP controls terminal I/O by editing messages and attention interrupts.

Terminal Monitor Program (TMP)
TMP processes commands from each terminal and passes control to the appropriate command processor.

Command Processor (CP)
CP executes terminal commands.

Service Routine
The service routines are the programs supplied under TMP and CP which check the syntax of input commands, allocate device buffers for CP, and dynamically allocate data sets for the execution of user jobs.

Syntax Checker
This program checks the syntax of each statement of a FORTRAN or PL/I source program, and prints any required error messages.

Prompter
Prompters provide the user with a conversational method of defining a job. The prompter inputs commands from the terminal, allocates system libraries and user data sets needed for program execution, prompts the terminal user to input any missing operands, invokes the proper processor, and returns all resources when processing is complete. The following prompters are available:
- FORTRAN.
- COBOL.
- PL/I.
- Assembler.
- LINK (for Linkage Editor).
- LOADGO (for Loader).

Interactive debug
Interactive debug facilitates the debugging of FORTRAN or COBOL source programs. The user may set breakpoints in his program, define initial values of variables, execute his program from breakpoint to breakpoint, and display any storage location desired.

Fig. 1.3 illustrates the TSS architecture.

1.4 OUTLINE OF PROCESSING

Initialization of TSS at the Installation
The console operator must perform the following tasks to start TSS:
- Start VTAM.
- Start the Time Sharing Control Task (TSCT) to initiate TSS processing.

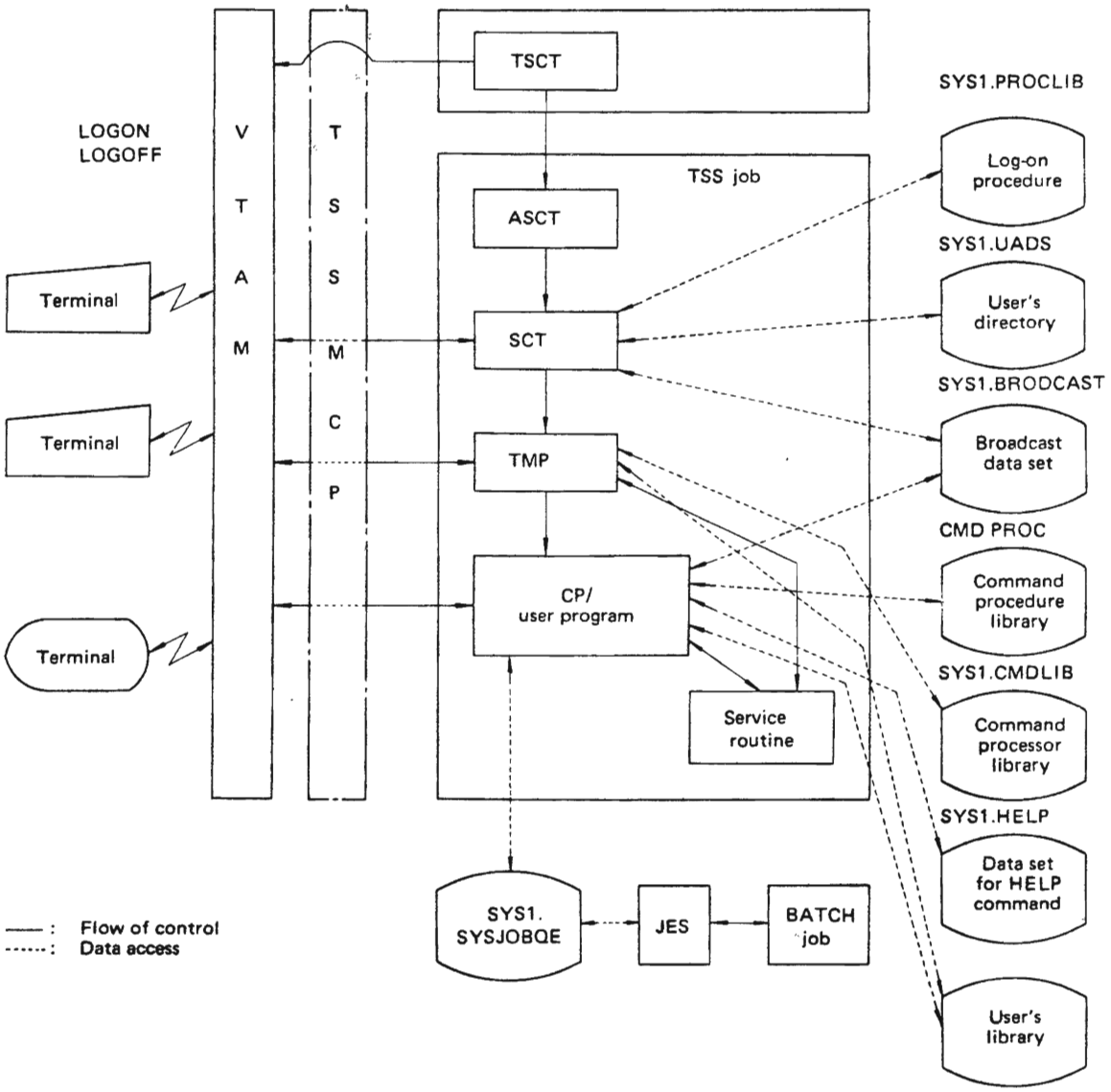


Fig. 1.3 TSS architecture

Starting a Terminal Session
The terminal user must take the following steps to initiate a session:
- Switch on the terminal power source.
- Input the LOGON command with the user's registered name, password, account number, and log-on procedure. The registered name of the user identifies all his data sets. The password is checked to verify that the user is authorized to use the TSS system. All processing is charged to the user's account number. The log-on procedure identifies the cataloged procedure which defines the system resources and libraries needed by the user. Generally, the program designated by the EXEC statement in this procedure is a terminal monitor program (TMP).

Terminal Processing
Input the command for the processing required. The system invokes the proper command processor from the command library. If an operand is omitted from the input command, any default value will be utilized. For any operand with no standard value, a message such as "ENTER XXXXXXX" is displayed at the terminal to prompt the user to input the missing information. Users may obtain information about a command, subcommand or operand by inputting "HELP" together with the name of the command.

Generally, a user must enter a series of commands during a terminal session. Part or all of this series can be optionally cataloged as a **command procedure** and invoked by an EXEC command. Command procedures can be prepared by each installation and/or generated by the user at his terminal.

Stopping a Terminal Session
The user terminates a session by the following steps:
- Enter LOGOFF command.
- Disconnect the terminal. The system updates the user's accounting information, releases the system resources, and disconnects the terminal from TSS. To commence a new session after cancelling the previous session, a LOGON command can be entered without a prior LOGOFF command.

Terminating TSS at the Center
The console operator must perform the following steps to halt TSS when all terminal sessions are complete:
- Stop the time sharing control task.
- Stop VTAM.

1.5 SUPERVISION OF SYSTEM OPERATION

The TSS Control Program uses tuning parameters set by each installation and information from each task to optimally distribute resources to each user.

Tuning parameters greatly affect resource distribution so the operating condition of the system must be monitored to assure optimum system efficiency. The functions which enable the system manager to determine the system condition are:
- OPERATOR command
  Any authorized user can use the OPERATOR command to display the number of active terminals and batch jobs submitted from TSS, modify TSS options that were defined when TSS was initiated, monitor the processing of both TSS and batch jobs, cancel a job submitted by TSS, and send or receive messages from other terminal users.
- SMF (System Management Facilities)
  The SMF data collection routines gather accounting information, such as CPU time, data set activity information, such as EXCP count by device, and volume information, such as the space available on given DASDs.
- GTF (Generalized Trace Facility)
  GTF assists the user in problem determination and diagnosis by tracing system and user events. GTF produces trace event records for the event types the user selects to monitor.

1.6 DATA AND PROGRAM PROTECTION

Data and program protection is important in a time-sharing system in which a number of users access system resources simultaneously. OS IV/F4 TSS has the following protection features include authorization check, program protection, and data set protection.

Authorization check
When the user inputs a LOGON command the system checks his name to verify that it is in the directory of authorized users. If a password is required, it is also checked. The directory defines which functions each user is authorized to use.

Program protection
Each user has an individual address space in TSS and may not access any other address space. His programs and data buffers cannot be destroyed by other users, nor can he destroy other users' programs.

Data set protection
The user may associate a password (which can be identical to his LOGON password) with any data set. Any other user must then input the proper password in order to open this data set.

There are two types of passwords:
- Renewal protection
  The password is required only if the data set is to

be rewritten, updated or erased.

- Read protection.
  The password is required if the data set is to be read.

1.7 SERVICE ROUTINES

The system contains service routines which perform syntax checking, I/O buffer control and data set allocation.

Parse/scan routine
The **scan routine** determines whether the command names in each I/O string are grammatically correct; the **parse routine** checks the syntax of each operand in the I/O string. Default values are provided for missing operands (when defined). The user is prompted to furnish the missing operands having no default values.

I/O service routine
The **terminal monitor program** (TMP) reads in commands from a terminal and transfers control to the proper command processors. The I/O service routine dynamically allocates I/O buffers for each command processor as needed.

Dynamic allocation of data sets
In general, data sets required for a user's job cannot be conveniently allocated until execution. The dynamic allocation facility allocates data sets as required, releasing them when no longer needed. The maximum number of data sets than can be dynamically allocated must be specified by each user in his LOGON procedure.

CHAPTER 2
TSS COMMAND LANGUAGE

2.1 GENERAL CONCEPTS

User attibute data set (SYS1.UADS)
Every user authorized to use TSS has an attribute data set which contains his registered name, password, account number, LOGON procedure and other user attributes. SYS1.UADS is compared with the user's LOGON information to check his authorization to use the system.

Broadcast data set (SYS1.BRODCAST)
This data set contains messages sent by the installation or one terminal user to other terminal users. Messages of general interest to all users are called NOTICES; messages for an individual user are called MAIL.

Terminal mode
A terminal is in **command mode** when it is ready to accept a command. OS IV/F4 indicates this status by writing a READY message to the terminal. When the user inputs a command with one or more **subcommands** such as the EDIT command, OS IV/F4 writes an EDIT message to the terminal, notifying the user that the terminal is in the EDIT subcommand mode. These messages are called **mode messages.**

The **entry modes** of the EDIT command are **editing mode,** which accepts subcommands, and **input mode,** which accepts data. Users generate source programs in input mode and then make corrections in editing mode. Modes can be changed by typing blank lines or Attention characters. Fig. 2.1 shows the relationship of the different modes.

Authorization
Use of the ACCOUNT, OPERATOR, or CRJE commands requires authorization from the installation. SYS1.UADS indicates whether these commands can be used. The ACCOUNT command is used to modify SYS1.UADS.

Data-set naming conventions
A **user identification qualifier** and a **content identification qualifier** should be prefixed and suffixed,

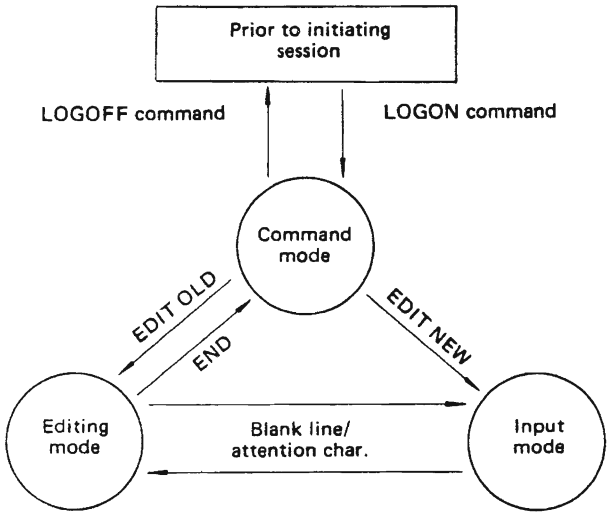


Fig. 2.1 Mode conversion

respectively, to the name of each user data set. The user identification qualifier is typically his registered name; the content identification qualifier describes the purpose and/or type of data set. For example, if a user is named "USER", his data set is named "DSN," and the data set is an assembler program, an appropriate name for the data set might be:

USER.DSN.ASM

All user data sets in TSS are cataloged in the above format.

Second-level messages
If a system message ends with "+", the user may demand more detailed information by inputting "?". The resulting expanded message is called a **second level message.**

Attention interrupt
The user can interrupt any executing program by pressing the attention key. Processing will continue if a blank line is input. However, if a new command is input, the current program is cancelled and the new command is processed. Any messages waiting to be sent to the user are erased.

Command procedure
A series of frequently executed commands can be cataloged as a command procedure. The procedure is invoked by the EXEC command which causes the series of commands to be executed as if it were input.

Text editing
A terminal may have character or line erase characters. The system converts all lower case letters into capital letters. If the user would input "a*Bb*de**e" when "*" is designated as the character erase character, "BE" would be accepted by the system.

2.2 SYSTEM CONTROL COMMANDS

The ACCOUNT command is used to add or delete a TSS user or to alter an attribute of a user (e.g., authorization to use the OPERATOR, SUBMIT, or ACCOUNT commands). Any designated user may employ the OPERATOR command to supervise the TSS system. The user may output the availability of system resources, output the number of TSS users, cancel jobs, and send messages to any terminal.

2.3 SESSION CONTROL COMMANDS

The LOGON command requests initiation of a terminal session. The registered user name, password, account number and LOGON procedure name are input. Multiple sessions cannot be established using the same registered user name. The LOGON information is compared with SYS1.UADS and if all inputs are valid, the terminal session may begin.

The LOGOFF command terminates a terminal session. When the LOGOFF command is input, the system updates the user's accounting information, releases the system resources allocated to the user, and disconnects his terminal from the system. If the user inputs a LOGON command without inputting a LOGOFF command, the system automatically terminates the prior session and establishes a new session.

2.4 PROGRAM-DEVELOPMENT AND DATA-ENTRY COMMANDS

The EDIT command is used to generate, update or delete source programs and data. The EDIT command has various subcommands and entry modes for input and editing. The input mode generates new programs and data; the editing mode updates or deletes existing files. For additional information, the reader should consult the **FACOM OS IV/F4 TSS Command Reference Manual**, the **FACOM OS IV/F4 TSS Command Processor Generation Guide**; and the **FACOM OS IV/F4 TSS Messages Handbook.**

Input mode
The user puts the terminal into input mode by designating NEW as an operand of the EDIT command. The system outputs the current line number on the left side of the terminal to prompt the user to input a line of data. If the user depresses the new line key after inputting each line, the system outputs the number of the next line. The INPUT subcommand of EDIT will also put the terminal in the input mode. If the user inputs a blank line, the terminal will switch from input mode to editing mode. Each line of a source program can be checked for syntax errors as it is input. The SAVE subcommand of EDIT must be used to save the newly-generated data set.

Editing mode
EDIT has numerous subcommands to update and delete data sets. The user can replace a given character string with another character string with the CHANGE subcommand. The DELETE subcommand deletes given lines in a data set. Both line number and contextual editing can be performed. The user can edit a specific line or group of lines by specifying the line number as an operand of an EDIT subcommand. Contextual editing operates upon character strings. For example, if the user is unsure which line contains the character string 'XXX' he may input FIND'XXX'. The line pointer will be moved to the line containing the given character string. The user may either output the line with the LIST subcommand or correct the line with the CHANGE subcommand. The line pointer can be moved up or down a given number of lines or set at the beginning of a data set. Entire lines can be inserted with the INPUT subcommand. The END subcommand terminates editing.

2.5 PROGRAM OPERATION COMMANDS

Each compiler generates an object module from a source program. The LINK command calls the OS IV/F4 linkage editor to convert the object module to a load module. The CALL command executes the load module. The RUN command translates a source program to an object module, converts the object module to a load module, and executes the load module. However, the object and load modules are not retained. The LOADGO command executes an object module, but the load module is deleted after execution.

2.6 DATA SET MANAGEMENT COMMANDS

Data set allocation, release, and deletion
The ALLOCATE command dynamically allocates output and work data sets when requested by a user. (The EDIT command defines input data sets.) The FREE command releases an allocated data set. The DELETE command deletes a data set which was created by the EDIT command.

Data set availability and other attributes
The LISTALC command lists data sets currently allocated to this user. The LISTDS command outputs for a given data set its creation date, record length, organization, and member names (if partitioned). The LISTCAT command lists the data sets which are cataloged under a given user identification number.

Data set protection
The PROTECT command associates a password with a data set so it can be read, written, or deleted only by authorized users.

2.7 DEBUGGING COMMANDS

The TEST command is used to debug assembler language programs by:

- Setting specific values into storage or general registers.
- Setting or releasing breakpoints.
- Executing a program from its beginning or from any relative address (or labeled statement).
- Printing the contents of storage or registers.
- Printing the contents of an OS IV/F4 control block.

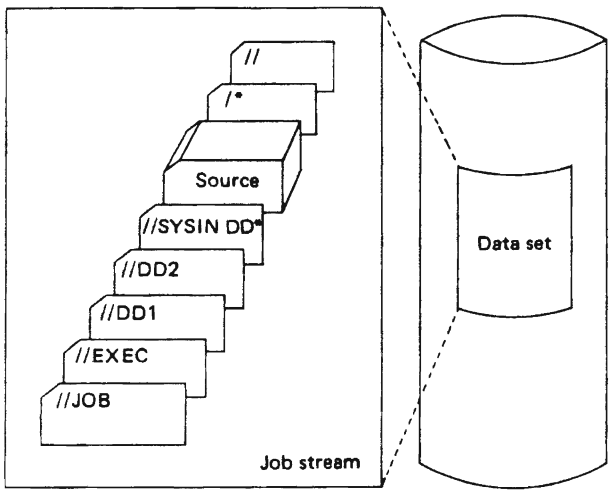
The **interactive debug** feature performs the same functions as the TEST command but is used for FORTRAN or COBOL programs.

2.8 TERMINAL CONTROL COMMANDS

The user can issue a TERMINAL command to designate an **attention character string, if his terminal** does not have an attention interrupt key. The user may also specify the logical line size or screen size of his terminal. A **character erase character** and/or **line erase character** may be defined with the PROFILE command.

2.9 CONVERSATIONAL REMOTE JOB ENTRY (CRJE) COMMANDS

The CRJE commands (SUBMIT, STATUS, OUTPUT, CANCEL) allow the TSS user to submit jobs to the OS IV/F4 batch initiators. The user must first generate a data set containing the job stream as card images.



The SUBMIT command transfers the job stream to the batch input queue. The STATUS command reports the status of a job submitted for batch processing. If "NOTIFY" is specified as a parameter on the JOB card, the user will be notified when his job has been executed provided he is still logged on to TSS. After the job has been executed, it can be printed with the OUTPUT command. The CANCEL command cancels a particular job.

2.10 MISCELLANEOUS COMMANDS

The TIME command displays the connect time and CPU time which has elapsed in the current session. The SEND command transmits a message to the installation or other users. The HELP command clarifies the meaning of any requested TSS command or gives the grammar of its operands.

CHAPTER 3
TSS LANGUAGE

The following TSS conversational language processors are available:
- COBOL.
- FORTRAN.
- PL/I.
- Assembler.

3.1 COBOL LANGUAGE

The user may generate, edit, compile, and execute COBOL source programs from his terminal.

COBOL source program generation
The EDIT input mode is used to generate source programs. Line numbers are automatically assigned to each input line. The line numbers are used to edit a particular line or to identify a line with a diagnostic message. The user may define logical tabs for COBOL statements by defining the column number for each tab.

COBOL program compilation
The user invokes the compiler with the CALL command to compile the source program. If the COBOL prompter is used, the required data sets and compiler options are defined and the compiler is executed. The compiler diagnostic messages are printed at the terminal. Any source program errors detected by the compiler can be corrected with the EDIT command.

COBOL program execution
The object program generated by the COBOL compiler is executed by the loader, using the LOADGO command. Any data sets required for execution must be allocated in advance with ALLOCATE commands. If the load module created from the object program is to be retained, the user can invoke the linkage editor with the LINK command. The user can then execute the load module with a CALL command. The RUN command compiles and executes a program in one step. The COBOL interactive debug facility aids the testing of COBOL programs. The user must first compile his program specifying the TEST parameter on the CALL command which invokes the COBOL compiler.

3.2 FORTRAN LANGUAGE

The following two compilers are provided for FORTRAN:
- FORTRAN IV GE (abbreviated **GE**).
- FORTRAN IV HE (abbreviated **HE**).

GE is appropriate for program development and short executions. It has the following features:
- Compilation time is minimized.
- An interactive debug package is available.

HE is appropriate for long production runs and produces:
- Minimum object program size.
- Comprehensive cross reference listings of source program variables and statement labels.

The GE language is a subset of the HE language with the exception of:
- interactive debug package,
- free-form input of source statements, where verbs and operands can be separated by arbitrary numbers of blanks, rather than having fixed positions.

A conceptual diagram of FORTRAN is given in Fig. 3.1.

The user may generate, edit, compile, and execute FORTRAN source programs from his terminal.

FORTRAN source program generation
The user generates source programs using the EDIT command in the same manner as in COBOL. GE will accept free-form input which does not comply with standard FORTRAN. Free-form input statements may start in column 1 and extend across as many lines as needed, using the hyphen as a continuation character. The **CONVERT command** will convert free-form to standard FORTRAN input (statement

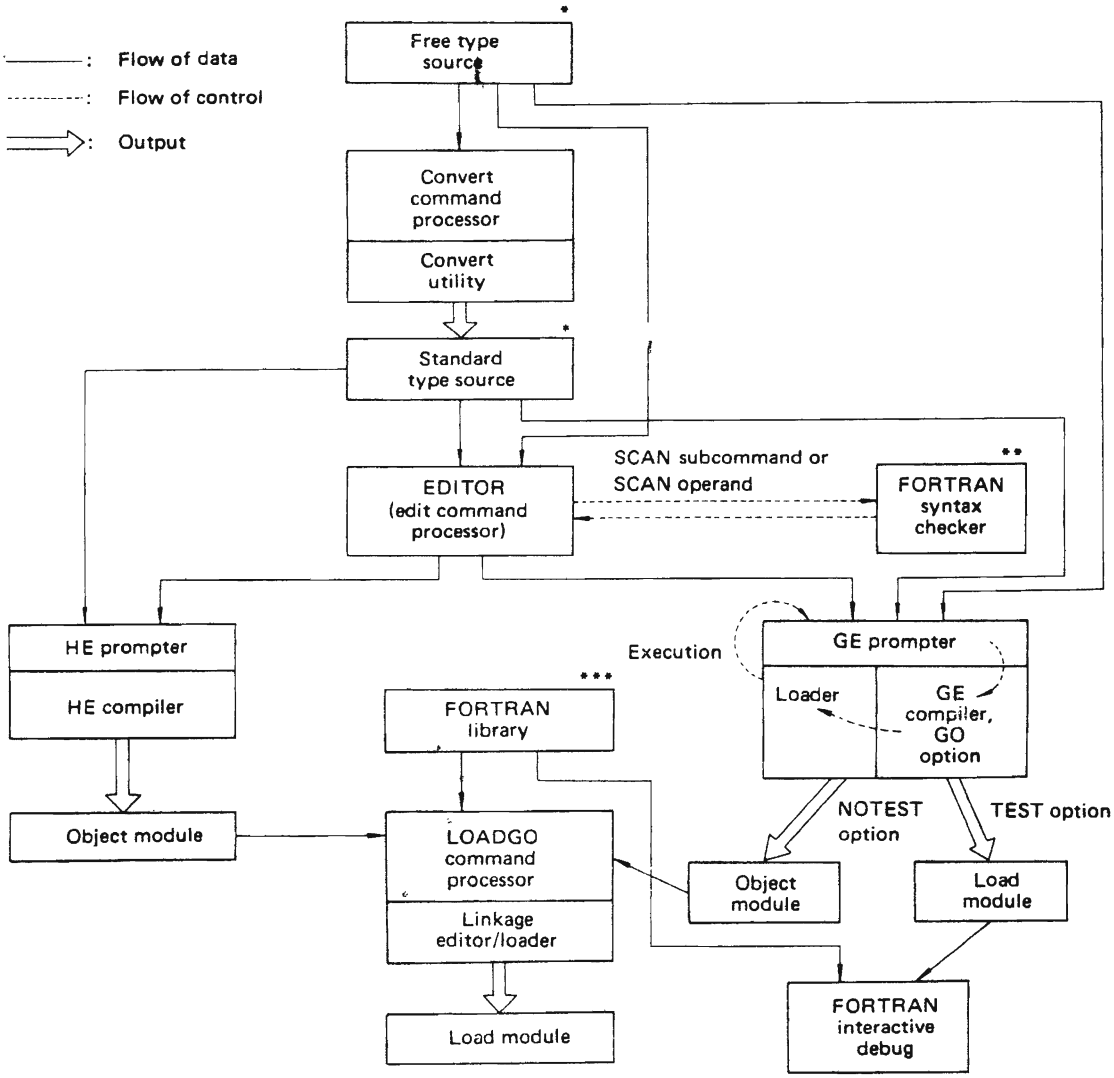


Fig. 3.1 Flow of data and control in FORTRAN

* Free form source has no set column alignment. Standard type source conforms to the proper columns prescribed by ISO.

** Syntax checking can be performed for both GE and HE source statements.

*** The FORTRAN library is common to GE and HE. It contains functions (such as absolute value and minimum) which are called by FORTRAN programs.

labels, continuation characters, and statements are in specific columns).

The Syntax Checker detects syntactical erros in source statements as they are input with EDIT subcommands:

FORTRAN compilations

FORTRAN programs can be compiled by either GE or HE. When the compiler is invoked by the FORTRAN prompter, necessary data sets and default compiler options are automatically provided by TSS. Diagnostic messages are printed at the terminal. Source statements can be corrected with the EDIT command, facilitated by the line numbers of each erroneous line printed in the diagnostic messages. The user may also print the entire program listing at his terminal.

FORTRAN executions

Object modules generated by GE or HE can be executed with a LOADGO command. When the object module is converted to a load module, pertinent libraries can be linked and saved for future use with the LINK command. A load module (i.e., production program) can be executed with the CALL command. GE and HE object modules can be arbitrarily combined into load modules. A RUN command automatically compiles and executes a source program. The FORTRAN interactive debug feature facilitates testing FORTRAN programs. The program must be compiled and link-edited with the TEST option prior to execution.

3.3 PL/I LANGUAGE

PL/I programs can be generated, compiled, and executed at the user's terminal.

PL/I source program generation
The user generates source programs with the EDIT command in the same manner as in COBOL. TSS sets logical tabs for the input statements and automatically assigns line numbers. The syntax checker detects any syntactical errors as the source statements are input.

PL/I compilations
The user invokes the compiler to compile the source program. If the compiler is invoked by the PL/I promter, TSS provides all necessary data sets and compiler options and executes the compiler. The compiler prints diagnostic messages at the terminal and the user can correct any erroneous statements with the EDIT command.

PL/I executions
The object module generated by the compiler is executed with the LOADGO command. The LINK command converts the object module to a load module and saves the load module. The CALL command executes the retained load module. The RUN command compiles and executes a source program in one step.

3.4 ASSEMBLER LANGUAGE

Assembler programs can be generated, compiled, and executed from a terminal.

Assembler source program generation
The EDIT command generates Assembler source programs in the same manner as COBOL. Source statements can be input free format.

Program compilation
The user invokes the assembler to assemble the source program. If the assembler is invoked by the prompter, all the necessary data sets and assembler options are automatically provided and the assembler is executed. The diagnostic messages of the assembler are output at the user's terminal.

Program execution
The LOADGO command executes an object module. The RUN command assembles and executes a source program. The LINK command converts an object module to a load module and saves the load module. The load module is executed with the CALL command. The TEST command is used to debug an assembler program which has been assembled and link-edited with the TEST option.

INDEX

COMMENT FORM

Please use the form below to write whatever comments and suggestions you may have regarding this publication. The completed form should be given to the FACOM representative in your area.

Your opinions please.
Please mark each item below with the appropriate letter representing your frank views on this publication, i.e. E (excellent), G (good), F (fair), P (poor).

☐ Text usefulness	☐ Illustrations/tables	☐ General appearance
☐ Text clarity	☐ Index coverage	☐ Paper quality
☐ Text accuracy	☐ Cross referencing	☐ Printing
☐ Text organization	☐ _____	☐ Binding

Detailed comments:

Name: _____ Position: _____

Company or organization: _____

Address: _____ Reply requested: No

_____ Yes

FOR OFFICE USE ONLY. Do not fill in here.

Local representative: Date received:

Documentation section Date received:

Action:

Seen and checked by_____