Xerox Data Systems

701 South Aviation Boulevard
El Segundo, California 90245
213 679-4511

XEROX

Xerox Operating System (XOS)

Sigma 6/7/9 Computers

Batch Processing

Reference Manual

FIRST EDITION

90 17 65A

December 1971

Price: $9.00

Printed in U.S.A.

NOTICE

This publication documents the A00 version of the Xerox Operating System.

RELATED PUBLICATIONS

Title	Publication No.
Xerox Sigma 6 Computer/Reference Manual	90 17 13
Xerox Sigma 7 Computer/Reference Manual	90 09 50
Xerox Sigma 9 Computer/Reference Manual	90 17 33
Xerox Operating System (XOS)/BP User's Guide	90 17 67
Xerox Operating System (XOS)/UT Reference Manual	90 17 69
Xerox Operating System (XOS)/SM Reference Manual	90 17 66
Xerox Operating System (XOS)/OPS Reference Manual	90 17 68
Xerox ANS COBOL (for XOS)/LN Reference Manual	90 18 37
Xerox ANS COBOL (for XOS)/OPS Reference Manual	90 18 38
Xerox Sort (for XOS)/Reference Manual	90 18 39
Xerox Data Management System (DMS)/Reference Manual	90 17 38
Xerox Extended FORTRAN IV/LN Reference Manual	90 09 56
Xerox Extended FORTRAN IV (for XOS)/OPS Reference Manual	90 18 40
Xerox Meta-Symbol/LN, OPS Reference Manual	90 09 52

Manual Content Codes: BP - batch processing, LN - language, OPS - operations, RBP - remote batch processing, RT - real-time, SM - system management, TS - time-sharing, UT - utilities.

CONTENTS

APPENDIXES

FIGURES

TABLES

EXAMPLES

PREFACE

The purpose of this manual is to

- Describe the functional aspects and capabilities of the Xerox Operating System (XOS) – Chapters 1 and 2.
- Provide complete reference information necessary to the preparation and submission of batch and remote-batch jobs under XOS – Chapters 3 through 8.
- Describe the facilities available for the development of telecommunications applications – Chapter 9.

The primary intended audience of this manual is the applications programmer. For the experienced applications programmer, this manual should provide sufficient reference information when used in conjunction with the XOS Utilities Reference Manual, 90 17 69, and programming language reference manuals (see "Related Publications"). For the device programmer, or one unfamiliar with any comparable operating system, it is recommended that the XOS Batch Processing User's Guide, 90 17 67, be used as a starting point instead of this manual and the XOS Utilities Reference Manual. (The appropriate language publications will still be required, however.)

For the system programmer and system manager, the information in this manual is prerequisite to a thorough comprehension of the XOS System Management Reference Manual, 90 17 66.

In addition to this and the above-mentioned publications, the remote-batch user who needs to act as his own operator at a remote-batch terminal site must refer to the XOS Operations Reference Manual (Chapter 9), 90 17 68.

In this manual, Chapters 1 through 4 are of equal interest to all language users; at least portions of Chapters 5 and 6 (especially the latter) are also applicable to all users. Chapter 6 in particular (File Organization and Management) contains information that is closely related and supplementary to subjects covered in Chapters 3 and 4. (Chapter 6 is also prerequisite to Chapter 7.) Chapters 7 and 8 need be referred to only by users of the Meta-Symbol programming language, and Chapter 9 only by users of the Telecommunications Management (TMS) facilities.

GLOSSARY

access method: any of the file management techniques available to the user for transferring data between main storage and an input/output device. The access methods are

ASAM – assisted sequential access method.

AIAM – assisted indexed sequential access method.

APAM – assisted partitioned access method.

VSAM – virtual sequential access method.

VDAM – virtual direct access method.

BDAM – basic real direct access method.

TAM – telecommunications access method.

account catalog: a catalog that resides on an account volume and contains entries that point to the permanent files the owner has cataloged. The account catalog contains all the information necessary to completely and uniquely describe a file, including the actual location of the file.

account volume: a disk pack or pseudo-volume (an area on the system disk) that belongs to an authorized account and on which resides the account catalog. The system file management software has access to the files of an account via the account catalog.

ACK: the acknowledge character usually used in communications systems to indicate error free receipt of a transmission. (Contrast with "NAK".)

assisted access method: an access method that automatically synchronizes the transfer of data between the program using the access method and an input/output device, allowing logical-record processing by the user program.

asynchronous transmission: transmission in which each information character is individually synchronized (usually by the use of communication terminal determined elements).

attended operation: a remote terminal mode of operation in which individuals are required at both stations to establish the connection and transfer the data sets from talk (voice) mode to data mode. (Compare with "unattended operation".)

automatic volume recognition (AVR): the preprocessing of labels on a magnetic storage volume, initiated by the operator.

AVR: see "automatic volume recognition".

block (records): 1. (v) to group records for the purpose of conserving storage space or increasing the efficiency of access or processing. 2. (n) A physical record defined to be a unit of data transmission.

buffer (input/output): an area of storage into which data is read, or from which it is written by the system, on behalf of the user.

catalog: 1. (n) a directory to locations of magnetic storage media and/or files. 2. (v) to enter an item (e.g., a file) into a catalog.

cataloged procedure: a set of job control commands that has been placed in a special cataloged file, and can be retrieved by naming it in a !EXEC (execute) command.

central station: the central computer site in relationship to remote (telecommunications) stations.

command processing: the reading, analyzing, and performing of commands submitted via an input job stream or the console device.

common dynamic area: dynamically acquirable memory space, allocated downward from the upper end of the user's available dynamic space.

common volume: synonymous with "public volume".

console: the interface, or communication device, between a user or operator and the computer (e.g., operator's console, remote terminal console).

control program: a group of programs that provides such functions as the handling of input/output operations, error detection and recovery, program loading, and communication between the program and the operator.

data control block (DCB): a table that groups the parameters defining the logical structure of a file. This information is used to interpret the I/O requests given by the user.

DCB: see "data control block".

deblocking: isolating the individual logical records within a blocked (see physical record "block").

default option: an option that will automatically be assumed if not overridden by an explicit specification.

device name: the general name for a device, specified at the time the system is generated, and used for all symbolic references to the device.

device type (I/O): a mnemonic symbol used to describe class of I/O media having common physical characteristics, such as magnetic tape (MT), disk media (DM), etc.

diagnostic: 1. pertaining to the detection and isolation of syntactical or procedural errors. 2. a message noting the occurrence of a syntax error, coding error, etc.

direct-access media: file storage space on RAD or disk pack, i.e., file storage space to which access is not inherently sequential.

duplex: in communications, pertaining to a capacity for simultaneous independent two-way transmission on a single line. (Contrast with "half duplex"; synonymous with "full duplex".)

dynamic space: memory space that the user can request dynamically, i.e., during program execution.

ECB: see "event control block".

echoplex: in communications, a mode of operation wherein data is echoed back to the transmitting station by the receiving stations (on a duplex line).

end-of-file mark: a code signaling that the last record of a file has been read.

end-of-message (EOM): the specific character or sequence of characters that indicate the termination of a message of record.

end-of-tape marker: a marker on a magnetic tape used to indicate the end of the permissible recording area; for example, a photo-reflective strip, a transparent section of tape, or a particular bit pattern.

end-of-text character: a communication control character used to indicate the end of a message text.

end-of-transmission (EOT): the specific character or sequence of characters that indicates termination of transmission to or from a remote terminal.

error severity code: a code indicating the severity of errors noted by an assembler, compiler, or link editor and used to determine the possibility of subsequent processing of the resulting object or load module.

event: an occurrence of significance to a task; typically, the completion of an asynchronous operation, such as input/output.

event control block (ECB): a one-word element associated with each event that contains information relative to the status of that event.

file: a collection of related records treated as a unit. The records in a file may or may not be sequenced according to a key contained in each record.

file management: a general term that collectively describes those functions of the operating system that provide access to files, enforce data storage conventions, and regulate the use of input/output devices.

fixed length record: a logical record having the same length as all other records in the same file. (Contrast with "variable-length record".)

file protection: preventing unauthorized access to a file. The protection may be based on read access authorization, and/or write access authorization, or password protection, or any combination of these.

full duplex: in communications, pertaining to a capacity for simultaneous independent two-way transmission on a single line. (Contrast with "half duplex"; synonymous with "duplex".)

group of lines: two or more communication lines allocated as a unit to a program DCB.

half duplex: in communications, pertaining to an alternate, one way at a time, independent transmission. (Contrast with "full duplex".)

header label: the record at the beginning of a file or volume containing control information about the file or volume. (See also "trailer label".)

idle characters: control characters interchanged by a synchronized transmitter and receiver to maintain synchronization during nondata periods.

indexed sequential: a file organization in which record locations are determined from an index of key values contained in the records.

job: a specified group of program steps prescribed as a unit of work for the computer in that they must be scheduled as an entity. By extension, a job includes all necessary programs, files, and instructions to the operating system. A job consists of one or more job steps.

job class: a parameter that permits a measure of control over the relative priority of program scheduling and execution.

job control command: a command in a job that is used to identify the job or describe its requirements to the operating system.

job step: a unit of work for the system from the standpoint of the user, presented as a request for execution of an explicitly identical program, and a description of resources required by it. A job step consists of the external specifications of work that is to be done as a task or subtask.

key: a unique identifier associated with a record or a partition of a file that can be used to locate or access the record or partition.

link editor: a processor that produces a load module by transforming object modules into a format acceptable for execution and resolving all intermodule linkages.

load module: a complete program in a format suitable for loading into memory for execution.

local dynamic area: dynamically acquirable memory space, allocated upward from the lower end of the user's available dynamic space.

logical record: the unit of information processed in a single operation via an assisted access method.

message: in communications, a meaningful unit of transmission usually delineated communication control characters.

multiprogramming: the concurrent execution of two or more independent programs by a single central processing unit.

NAK: the negative acknowledge character usually used in communications to indicate the receipt of a transmission and some form of error indication. (Contrast with "ACK".)

network: a series of points interconnected by communications channels. (A private line network is a network confined to the use of one customer, while a switched telephone network is a network of telephone lines normally used for dialed telephone calls.)

nonswitched line: a private communication line that does not connect with a common carrier switchboard.

operational label: a one-to-four character identifier that associates a logical file and a physical file resource.

owner: the creator of a file, who may control access to the file by other users. (See "file protection".)

parallel class: the highest priority job class. Jobs in this class must contain only one job step. They may be executed as they are brought into the system or may be cataloged when they are brought into the system and executed at a later time.

password: a word used to protect against unauthorized access to standard files.

permanent file: a file, recorded on a magnetic device, that endures, in principle, beyond the execution, period of the program that creates it. A file identifier (i.e., a filename) is associated with the file, which permits it to be located symbolically.

physical record: the unit of information transferred in single operation between memory and an I/O device, which may include more than one logical record.

polling: a technique by which each of the terminals sharing a communications line is periodically interrogated to determine whether it requires servicing.

private volume: a removable volume belonging to the user(s) of a given account number.

production class: jobs of other than the parallel class, that may include several job steps. The possible production classes are named A, B, C, D, E, and T.

pseudovolume: a portion of the system disk allocated to a user account as an account volume.

public volume: a removable volume that can be assigned to any user account for the creation of files (usually temporary). Synonymous with "common volume".

quantum: the minimum unit for disk space allocation (8K bytes).

real address: an instruction address that refers to a physical location in main storage. (Compare with "virtual address".)

record: a collection of related items of data, treated as a unit for processing or I/O transmission.

record length: a measure of the size of a record, specified in bytes (characters).

remote station: a telecommunications terminal, (normally) connected to the control computer site by a communications line.

resource: any facility of the computing system or operating system required by a job. These include main storage, secondary storage, input/output devices, and the central processing unit.

resource allocation: reservation of a system resource for the use of a job or job step.

resource deallocation: freeing of a previously allocated system resource.

scheduler: a system component whose function is to allocate all resources for a job and to select the next job to be entered into the system for execution.

sequential access: a method of accessing data such that consecutive records are processed in the order that they are presented to the system.

shared file: a file which may be accessed by two or more users concurrently.

station: either a remote or central station in a communications network.

supercatalog: a table on the system disk containing a number of entries, each of which associates an account number to the serial number of an account volume, or to the system-disk address of a pseudovolume (an account volume by definition).

superjob: a series of jobs that are logically treated as one job for purposes of scheduling and conditional execution of the jobs in the sequence. This facility is available only to production classes A - E.

switched line: a communications line connected with a common carrier switchboard (i.e., dial-up line).

symbiont: a system program that executes concurrently with user and other system programs and applies to data transfers between disk and low-speed peripherals.

synchronous transmission: transmission in which the sending and receiving units are operating continuously at substantially the same frequency and are maintained, by means of automatic correction, in a desired phase relationship.

SYSGEN: see "system generation".

system generation (SYSGEN): a process that tailors XOS to meet the needs of the specific installation.

system startup: the process (and point in time) of loading and initializing the operating system.

task management: those functions of the control program that regulate the use by independent tasks of the central processing unit.

telecommunication: data transmission between a computing system and remotely located devices via a unit that performs the necessary format conversion and controls the rate of transmission.

temporary file: the same as a permanent file except that no identifier is associated with the file on its recording media. The file is identified only by an operational label, which is significant only during the execution of the job that creates it; the file is therefore "lost" to the system at end of job.

trailer label: a record placed at the end of a file or volume that contains control information about the file or volume. (See also "header label".)

unattended operation: the automatic features of a remote station's operation that permit the transmission and receipt of messages on an unattended basis. (Compare "attended operation".)

undefined-length record: a logical record having an unspecified length that is automatically equated to physical record length.

variable length record: a logical record having a length independent of the length of other records in the same file (contrasted with "fixed-length record").

virtual address: an instruction address that refers to a location in virtual memory and must be algorithmically converted to a real address before the instruction is executed (contrast with "real address").

virtual memory: a conceptual arrangement of main storage achieved via a hardware/software technique that permits memory addressing without regard to the program's physical location in memory. Each address reference is automatically translated (mapped) from where the program appears to be located (virtual address) to an actual location (real address). This technique further allows programs to be broken into noncontiguous blocks (pages) for better memory utilization.

volume: all that portion of a single unit of storage media that is accessible to a single read/write mechanism, e.g., a reel of tape, removable disk pack, etc.

volume protection: protection of a volume against unauthorized multiple concurrent access.

1. INTRODUCTION

The Xerox Operating System (XOS) is a large-scale multiprogramming operation system for Xerox Sigma 6, 7, and 9 computers. It provides both local and remote batch processing, and includes facilities for telecommunications applications. XOS employs the Sigma external priority interrupt system, the memory map, and the access protection feature associated with the map, for highly efficient performance of its basic supervisory functions.

- Up to eight levels of external priority interrupt automatically distribute central processing unit (CPU) time to active jobs (according to user defined priorities).
- Memory mapping hardware performs automatic program relocation, mapping the user's contiguous virtual memory space into noncontiguous physical memory.
- Memory access protection hardware provides selective access control for individual memory pages, protecting the system from the users, one user from another, and a user from himself.

Use of combined hardware and software supervisory techniques based on the facilities described above results in significant economies of system overhead and a high level of overall system efficiency.

Significant functional features of XOS include the following:

- Flexible control of job scheduling through the user's ability to assign his job to one of several (up to seven) job classes. The relative priority of, and resource limitations associated with, each job class are determined by the individual installation during system generation.
- Efficient concurrent utilization of system resources, especially of secondary storage and peripheral I/O devices.
- Efficient handling of large data files.
- A powerful and easy-to-use job control command language, allowing conditional execution of both job-steps and jobs, specification of total job and individual job-step resource requirements, communication between job-steps, etc.
- A full range of file management services.
- Sequential, indexed-sequential, partitioned, and direct file organizations.
- ANS compatible standard volumes and files.
- Symbiont controlled peripheral I/O devices and remote batch terminals.

Figure 1-1 illustrates the capabilities and primary resources of XOS. Some of the major aspects of the system are described in more detail below.

JOB CLASSES

XOS provides for up to seven separate job classes, each having a distinct priority. The classes may be categorized into (1) the parallel job class and (2) up to six production classes. (The class to which a given job is assigned is specified in the !JOB control command.) The number of separate production classes actually used is an option of the individual installation.

The parallel job class inherently has the highest scheduling and execution priority. Jobs in this class may be executed as they are submitted to the system, or cataloged when they are submitted and executed later upon operator command. (The latter option is only available for parallel class.) There is a single restriction on parallel class jobs: they may contain only one job-step. The number of parallel jobs active concurrently is limited only by the available system resources.

The six production classes are identified as follows:

Production class A

Production class B

Production class C

Production class D

Production class E

Production class T

The relative priority of each production class is established by the installation at system generation time, i.e., at the time that XOS is tailored to meet the needs of the particular installation. For example, the classes could be prioritized in alphabetical order, as shown above, with class A having the highest priority and class T the lowest. Alternatively, class T might fall between classes C and D. In general, the priorities need not be established in alphabetical order.

There are actually two types of priorities established at system generation time. One type determines the relative order in which jobs of the production classes are initiated, i.e., brought into memory and prepared for execution: this is the scheduling priority. The other type determines the order in which the corresponding tasks are to be activated, i.e., given control of the CPU, relative to other tasks awaiting activation: this is the execution priority.

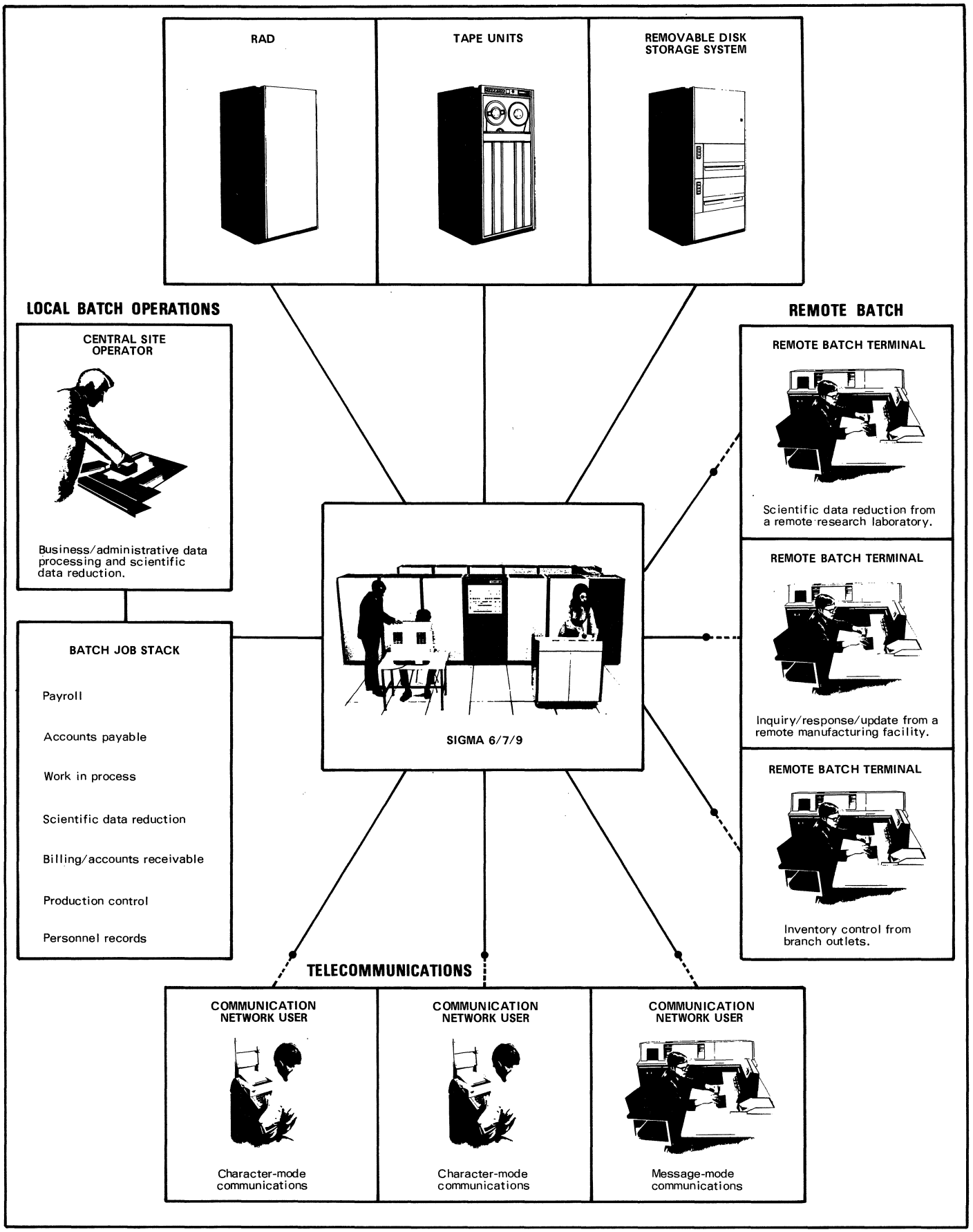


Figure 1-1. Generalized XOS Capabilities and Resources

In addition to the installation determined class priorities, jobs within the T class can be given a user assigned scheduling priority. That is, T class jobs can be prioritized relative to other T class jobs competing for initiation. This priority internal to the T class is specified also on the !JOB command.

The basic scheduling philosophy (described in more detail under "Scheduling") is as follows:

- If all queued parallel class jobs have been initiated (or none are queued), the system will examine the highest priority production class.
- If one job of a given production class has been initiated (or none are queued), the next lower priority production class will be examined. That is, only one job from each production class can be active at the same time.
- If a job of a given class cannot be initiated due to unavailability of required resources, then no job of a lower priority class will be examined for possible initiation.
- Except for the T class, no job other than the first in a production class queue (i.e., the "oldest" job in the class) will be examined for possible initiation. When the T class is examined, the complete queue will, if necessary, be examined – in priority order – to find a possible candidate for initiation.

Limits on the number, type, and extent of resources that a job may request are also established for the classes at system generation time. For each class, both default requirements and maximum limits are established for memory and temporary disk space, number and type of peripherals, maximum CPU time, and amount of output via symbiont devices.

The user selects the class in which his job should be run based upon his resource requirements. If several classes are appropriate, a good rule to follow is that I/O-bound jobs should be assigned to a higher priority class and compute-bound jobs to a lower priority class. In the multiprogramming environment of XOS, this helps increase system throughput.

A convenient feature of XOS is the ability of the user to chain several related jobs of one production class (other than T) into a superjob. The series of jobs constituting a superjob are executed sequentially. Each is executed only upon the proper completion of the preceding job. If any job of a superjob aborts, all remaining member jobs are ignored.

MULTIPROGRAMMING AND TASKS

Multiprogramming is defined as the concurrent operation of two or more jobs that reside simultaneously in memory. Multiprogramming is accomplished essentially by allowing the I/O operations of as many jobs as possible to be performed simultaneously while CPU control for internal operations is distributed among the several jobs on a priority basis. (It is not possible for the CPU to simultaneously execute the instructions of more than one job.) Multiprogramming allows total system resources to be used more efficiently and maximizes job throughput.

When a job has been initiated (i.e., taken from a queue of jobs to be scheduled, initialized for execution, and loaded into memory) it becomes a "task" to the system. Although all tasks within the system do not correspond to user jobs (e.g. system tasks), for our current purpose we will consider only tasks originating from such jobs. The number of tasks allowed in the system at one time depends on the amount of available resources and certain scheduling restrictions (see "Scheduling" below). In effect, a variable number of tasks may be performed. Figure 1-2 illustrates the general flow of job and task management functions, showing the time relationship between the two.

Once a task is activated, it will execute until it requests I/O, voluntarily enters a wait state, terminates, or is interrupted by some event initiated by a higher priority task. In any of these cases, the task becomes inactive and the CPU is free to process the instructions of the highest priority task waiting to execute. When the interrupted task is again activated, it will resume execution at the point at which it was interrupted. This is possible because XOS saves the total environment of each task when it is interrupted and restores that environment when the task resumes execution. Note that a user task corresponds to a complete job, but in the case of jobs consisting of multiple job-steps – i.e., independently processable portions of a job – only the program module required by a single job-step will be active at any given time.

Each job class is associated with one level of the Sigma external priority interrupt system. (The external interrupt system is distinct from interrupts associated with I/O operations, arithmetic overflows, instruction faults, etc.) The tasks automatically derive their execution priorities from the interrupt level with which they are associated. (Two or more production classes may share the same interrupt level.)

Use of the priority interrupt system to distribute CPU time among the tasks significantly reduces the amount of software required for multiprogramming control.

MEMORY MANAGEMENT

The multiprogramming of a variable number of tasks, each of variable size, inherently results in the fragmentation of available physical memory space during system operation. Under conventional operating systems, currently loaded programs must often be physically redistributed in memory in order to provide enough continuous space to load another program, or the loading must wait until enough continuous space becomes available on a contingent basis. Under

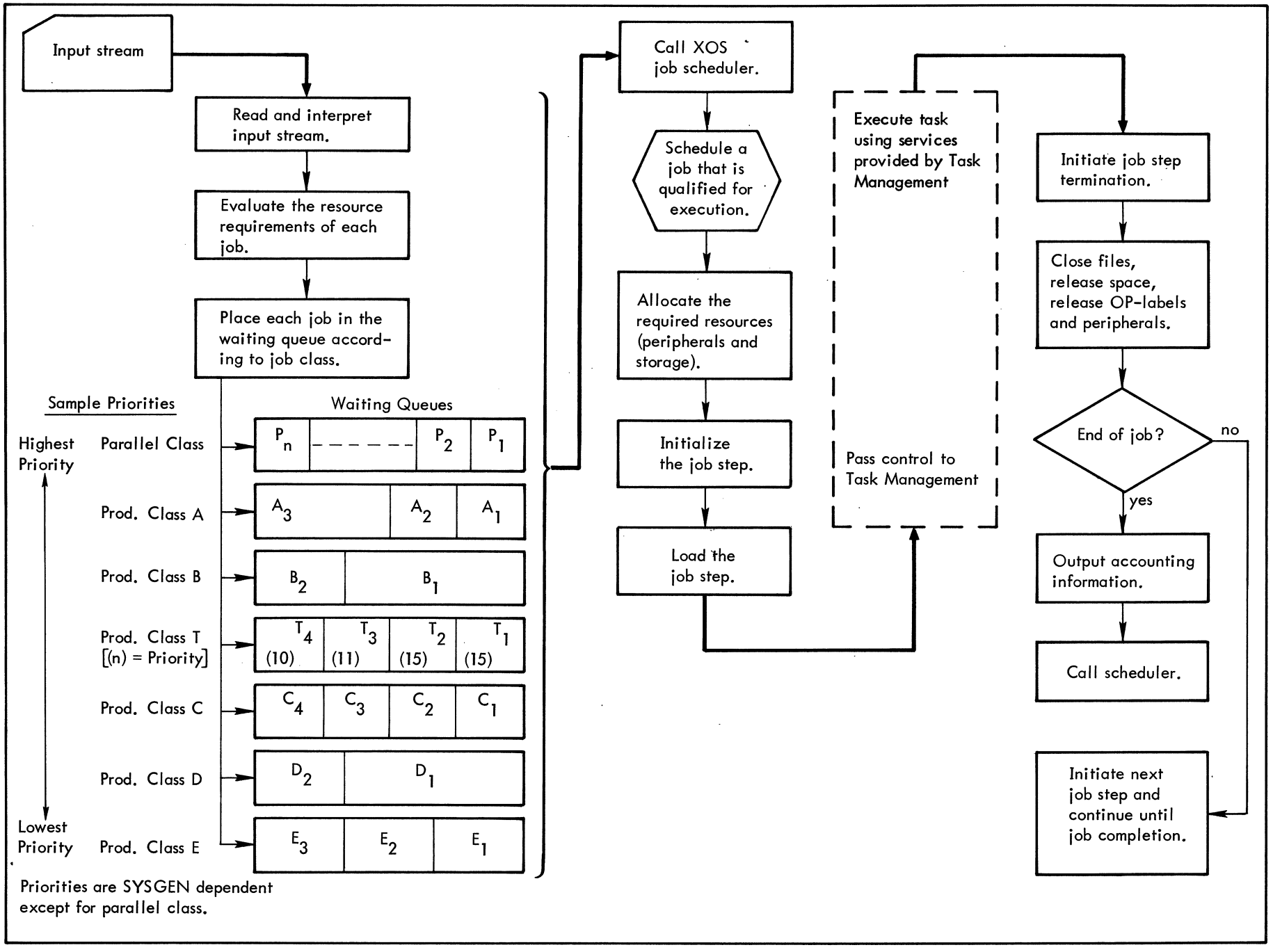


Figure 1-2. Conceptual Illustration of XOS Job and Task Management

XOS, however, a user program can automatically be fragmented at load time, as necessary (that is, be made to occupy noncontinuous memory space). The areas of physical memory occupied by a program module are dynamically determined by the circumstances of memory utilization — by other users and by the system — that prevail at the moment the module is loaded. The ability to "page" programs into many small noncontiguous areas of memory at actual load time is vital to an efficient operating system.

By means of the concept of virtual memory, all user programs (and most of the operating system itself) are planned and programmed as if they were always to operate in the same fixed and continuous area of memory without regard for the probable fragmentation of the programs as actually loaded. Use by XOS of the Sigma hardware memory mapping feature makes possible the concept that all of the user's memory space is logically continuous and, in a sense, multilayered (from the viewpoint of multiple concurrent users).

A brief description of memory usage by the control elements of the system follows, prior to a further discussion of the user's virtual memory space.

MONITOR RESIDENCE

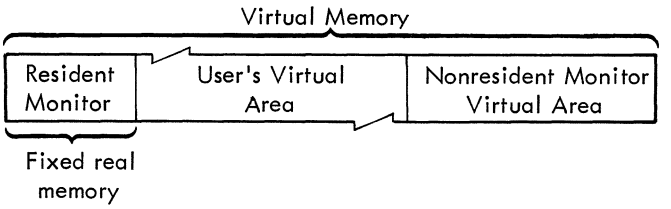
The control elements of the operating system, referred to collectively as the monitor, direct all program processing. During processing they perform many services that are either transparent to the user or explicitly called for by the user. The monitor is organized in two parts (with respect to memory residence); a small resident monitor that remains in memory at all times, and a nonresident portion that resides on secondary storage and is brought into memory as needed.

The XOS monitor is divided into resident and nonresident portions because, due to the large number of services it provides, it is not practical to keep all of it permanently resident in memory. Relatively few of these services are required frequently enough to justify being made resident; the majority are made nonresident, thus saving space for additional user tasks.

The nonresident monitor is physically divided into a number of elements that are independently loaded into memory as required. When one of these elements that was loaded into memory is no longer in use, it remains in memory but is marked "disengaged". The resident monitor maintains statistics on the frequency of use of these "disengaged" elements, and when additional memory is required the least frequently used element(s) are overlaid by the program or element that requires space. Using this technique, the system is able to make the most efficient use of "unused" memory and significantly reduce the number of requests for loading nonresident monitor elements.

The range of virtual addresses of the nonresident monitor area is established when the system is generated for an installation. Although the nonresident monitor area is assigned addresses in high virtual memory, the nonresident monitor elements are actually loaded anywhere in the same area of real memory space also used to contain user program modules.

The resident monitor occupies a fixed area of real memory beginning at real address zero, and also a corresponding amount of one-for-one virtual memory space (i.e., real and virtual addresses are identical in the resident monitor). Virtual versus real memory is organized as illustrated below.



VIRTUAL MEMORY AND THE MEMORY MAP

All user programs and system processors are automatically origined, during link editing or SYSGEN, at a fixed virtual address (normally at the beginning of the user's virtual space). Obviously, if several program modules must occupy memory concurrently (which is commonly the case), they can not be stored in the same physical location.

Memory mapping allows the program modules to be divided into virtual pages, each being 512 words long, and stored by page wherever physical space is available (without regard to logical organization). Physical memory is divided into corresponding physical pages of 512 contiguous words. When the program is loaded into memory, the logical pages of the program are mapped into available physical pages. The contiguous logical pages of the program need not occupy contiguous physical pages of memory. At execution time, the hardware memory map automatically translates the program's virtual address references, instruction by instruction, into an appropriate set of physical addresses so that the program will execute as if it had been stored into one contiguous area. This process is shown in Figure 1-3. This technique enables XOS to maximize the use of memory space, especially for concurrent operation of many small to moderate size programs.

All of memory is mapped in XOS, even that part occupied by the resident monitor. However, the resident monitor area is mapped one-to-one real to virtual so that within the resident monitor, virtual addresses always reference the corresponding physical location.

NONRESIDENT MONITOR AREA

The virtual area that is reserved for the nonresident segments of the monitor must be two pages or larger in size. In addition to a corresponding number of reserved real physical pages, the monitor will use any other free physical memory available for loading additional nonresident

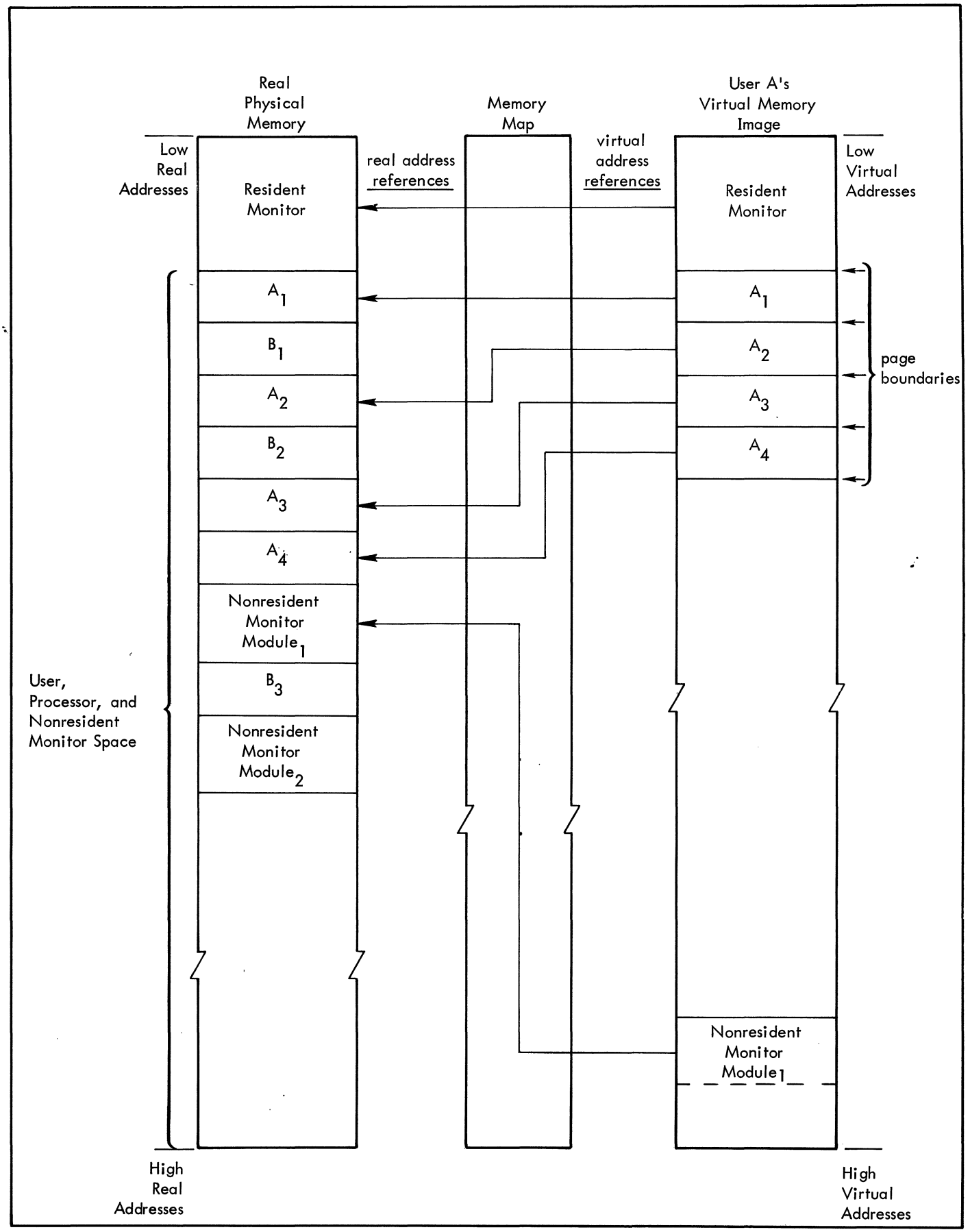


Figure 1-3. Memory Map Operation

segments as required. This free area is made available as follows: when a user's job enters the system, he will indicate either by a !LIMIT command or by class default values the maximum number of 512-word pages that his program may ever use. These pages are not dedicated to him when resources are allocated. Rather, pages are given to the job on a demand basis up to the limits given by the !LIMIT command or default value. Suppose the job's !LIMIT command indicates 50 pages as a maximum available to the job, but the job actually uses only 30 pages. The monitor is aware of the 20 free pages of memory and will load nonresident monitor modules to occupy the free space. The user program in this example may dynamically acquire (and release) up to 20 pages of additional memory. If the user asks for memory that currently contains nonresident monitor elements, the resident monitor will determine whether or not a given element is currently being used and how often in the past it has been used. In this way, the monitor can release to the user the memory occupied by the least used nonresident elements. In certain operating environments, some nonresident elements may become pseudoresident elements in that they are loaded into memory and the memory they occupy may never be subsequently required.

SYMBIONTS

Symbionts are system programs that asynchronously buffer – on disk – I/O operations for the following peripheral devices: card reader, card punch, line printer, and remote batch terminal. When a user job requests symbiont I/O – taking the case of output – the I/O takes place between memory and disk, rather than directly between memory and the peripheral device actually requested. The program is thus freed of the necessity of executing essentially at device speed. The appropriate symbiont independently transfers the data between disk and the peripheral device so that job throughput is not delayed by time consuming I/O on a slow device. Symbionts operate concurrently with other system and user programs, and normally operate one level above the highest user-priority level (P class). They are initiated by the computer operator.

For program initiated input/output, the user makes the choice between symbiont I/O or direct device access in a control command (!ASSIGN) that describes the device assignment for a given data file. Since symbionts greatly decrease job execution time, they should be used whenever feasible.

All jobs are entered into the system through the system card reader (IN) symbiont. This action is initiated and controlled by the operator. Each job is stored as a series of card images in a separate file on disk. A system program called the Control Card Interpreter (CCI) reads the file's control cards and adds information at the end of the file to be used by the system scheduler. The scheduler then determines when the job should be brought into memory for operation. Conversely, all listing-log information (job control output written by the system) is output via the system line printer (OUT) symbiont.

RESOURCE ALLOCATION

The allocatable system resources are categorized under XOS as follows:

- Shared resources. Core storage space and secondary storage space (temporary file space on system disk).
- Common resources. All input/output devices when specified simply by media type (e.g., any 9-track magnetic tape drive, any disk drive, any card reader). Also peripheral devices specified by symbiont name (e.g., IN, OUT, SCP).
- Reserved resources. Any specific input/output device specified by type and logical address, such as a specific line printer, or a group of transmission lines.

The reserved resources are so called because they must be reserved unconditionally for the job before the job can be initiated (i.e., scheduled for execution). The user can exercise no control over the scheduling of his job in regard to this type of resource (if any such is required), and therefore it will not figure in the following discussion. In the case of shared and common resources, however, the user can favorably affect the scheduling of his job – and overall system efficiency – by a judicious combination of job class assignment and control command usage, specifically the !LIMIT, !SLIMIT, and !RESOURCE commands. (These commands are described in detail in Chapter 3.)

In general, a submitted and queued job will not be initiated until (1) it is eligible – by virtue of its relative priority – to be examined for scheduling and (2) when examined, its required resources are available for allocation. The user's means of control over condition 1 is his choice of an appropriate job class; guidelines for making this choice were given under "Job Classes," above. Whenever condition 1 is satisfied but condition 2 is not, the job effectively "loses one turn" in the scheduling cycle, since the next scheduling cycle will start again with the top priority job class, not the last class that could not be initiated (see "Job Scheduling", below). Therefore it is often imperative for best scheduling that the system know (1) the actual extents of shared resources required to initiate the job – rather than the sometimes excessive default values, and (2) the minimum common resources required to initiate the job – rather than the common resources required for the entire job. (The latter is applicable only to multistep jobs.) Each of these cases is discussed separately below.

SHARED RESOURCES

In lieu of user specified values for the actual memory and temporary file space requirements of the job, the system

assumes the default values for its job class (as distinct from the maximum values for the class). The user can specify his actual overall job space requirements (plus maximum CPU time and maximum number of cards to be punched/pages to be printed) on the !LIMIT control command. These specifications can be less than or greater than the class default values, but cannot exceed the maximums imposed on the class.

It is important to note that the shared resource requirements of a later step of the job can exceed the !LIMIT (or default) value if the !SLIMIT command is employed for that individual step. This command allows the requirement limits to be temporarily raised for one or more steps that represent peaks in the overall job requirement profile. Since scheduling and initiation of the job itself depend upon the !LIMIT or default values, use of !SLIMIT commands in conjunction with the !LIMIT command (or default values) can favorably affect job scheduling. (This command must be used with caution, however, especially with regard to releasing temporary disk space exceeding the job limit prior to job-step end.) If the !SLIMIT command for a given job-step requests more resources than are available at that point in time, the job is placed in a wait state until such resources are freed.

COMMON RESOURCES

The system learns the exact total job requirement for common resources directly by inspection of the job control commands (specifically the !ASSIGN commands) for the job. In a multistep job, it may often happen that the number of common resources required to begin the job (e.g., to execute the first two job-steps) is significantly less than the total job requirement. The user can inform the system of this circumstance by means of the !RESOURCE command. This allows the job to be initiated before the common resources required by the whole job are available.

This command must be used with considerable restraint, however. Unjustified use of this command by several concurrent jobs could cause the system to become locked in a condition sometimes referred to as a "fatal embrace". In this condition one job is waiting for resources allocated to other jobs which in turn are waiting for the very resources already allocated to the job waiting for... . Clearly, users should employ this facility with some caution.

JOB CLASSES VERSUS RESOURCE LIMITATIONS

Table 1-1 shows a set of sample values for default and maximum resource limits, by job class, for a system incorporating five job classes. The values shown are purely for purposes of example, and thus are somewhat arbitrary. However, the values illustrate what an installation might establish, given the mix of job types represented in the table.

Table 1-1. Example Job-Class Resource Limits

Scheduling Priority	Job Class	Sample Job Types	CPU Time (minutes)		Temporary Disk Space (quanta)		Core Space (pages)		Peripheral Usage
			Def.	Max.	Def.	Max.	Def.	Max.	
highest	P	Operator initiated utilities (PREP, LISTCTG, FMGE-COPY).	2	5	0	15	15	50	very low
↑	A	Small user programs with high I/O activity (report generators, media conversions).	5	15	0	20	15	50	high
	B	Larger user programs but still largely I/O bound (SORT jobs, accounts receivable application).	5	30	0	50	15	75	high
↓	T	Assemblies, compilations, and link edits (COBOL, FORTRAN, Meta-Symbol, LINK).	10	20	75	200	50	100	moderate
lowest	C	Large assemblies and compilations; compute-bound user programs (scientific analysis programs, inventory/sales trend analysis).	15	no limit	75	200	75	all core	moderate

JOB SCHEDULING

Figure 1-4 is a functional flow diagram of XOS job scheduling. (The production job scheduling priorities shown have been arbitrarily chosen for the sake of example.) As illustrated therein, XOS schedules jobs according to the following rules:

1. Parallel (P) class jobs always have the highest scheduling priority. As many P class jobs as are possible within the constraints of available resources are initiated concurrently. If the scheduler cannot initiate a given queued P class job because of lack of resources, it will wait for the necessary resources to become available. The scheduler will not examine a job of any production class until initiation of all currently queued parallel class jobs has been achieved.

2. Production class job queues are examined for job initiation according to the class scheduling priority established by the installation. (Once initiated, the corresponding tasks are executed, relative to one another, following another priority, which may or may not be the same as the scheduling priority.) Only one job of each class is initiated at a time. However, one job from each of the classes may be initiated concurrently if enough system resources are available.

3. The parallel class queue and all production class queues except T operate on a first-in, first-out basis. If, for example, the first job in the production class A queue cannot be initiated, the scheduler will not look at the next job in the A queue. It will wait for resources to become available for the first job in the queue. For production class T, however, the scheduler will attempt to find any one job in the T class queue for which enough resources are available. It will examine the jobs according to a priority within the T class, which is assigned in the job control command !JOB for each job. (These subpriorities are only available for production class T.)

4. If the scheduler can not initiate a waiting job of a given class, it will not examine any lower priority job class. Rather, it will wait for any resource to be freed, and will then repeat the cycle with the highest priority queue, normally the parallel class.

When a job is loaded into memory, it must be loaded in its entirety (by job-step, if several), rather than by pages as the pages are needed. If core space is not available for all of the pages of a particular job (the requirement being determined by the class default, !LIMIT command, or !SLIMIT command), the job will not be initiated until enough core space becomes available.

GENERAL NOTATION CONVENTIONS

The following conventions are used in control command and procedure syntax descriptions and examples throughout this manual:

1. Uppercase Characters. A symbol in uppercase characters (e.g., KEY) is a keyword and, if used, must appear exactly as shown.

2. Lowercase Characters. A symbol in lowercase characters (e.g., name) is an element to be replaced by a user chosen value (decimal integer, character string, keyword, etc.), as appropriate.

3. Brackets [ ]. An element between brackets is optional.

4. Braces { }. Elements placed one under the other between braces require the selection of only one of those shown if the element group is selected. See also convention 6.

5. Ellipsis ... . An element followed by three successive periods is a repeatable element. The required separator character, if any, is shown preceding the ellipsis.

6. Vertical Stroke |. Elements (generally keywords) separated by the vertical stroke character require the selection of only one of those so separated if the element group is selected. A series of elements so separated is equivalent to the same elements displayed vertically between braces (see convention 4, above).

7. Except as noted in rules 8-10, all special characters such as ! $, . : ( ) are necessary syntactical elements and must appear as shown in the syntax descriptions.

8. The hyphen (-) does not have any syntactical meaning except in the :TREE command (Chapter 4) and, with that exception, should be considered as part of the substitution symbol (e.g., acct-number) in which it appears.

9. The percent sign (%) has a special syntactic meaning; its use is optional, and is described in Chapter 3.

10. The combined plus-minus sign (±) implies that either a plus or minus sign should be employed in its place.

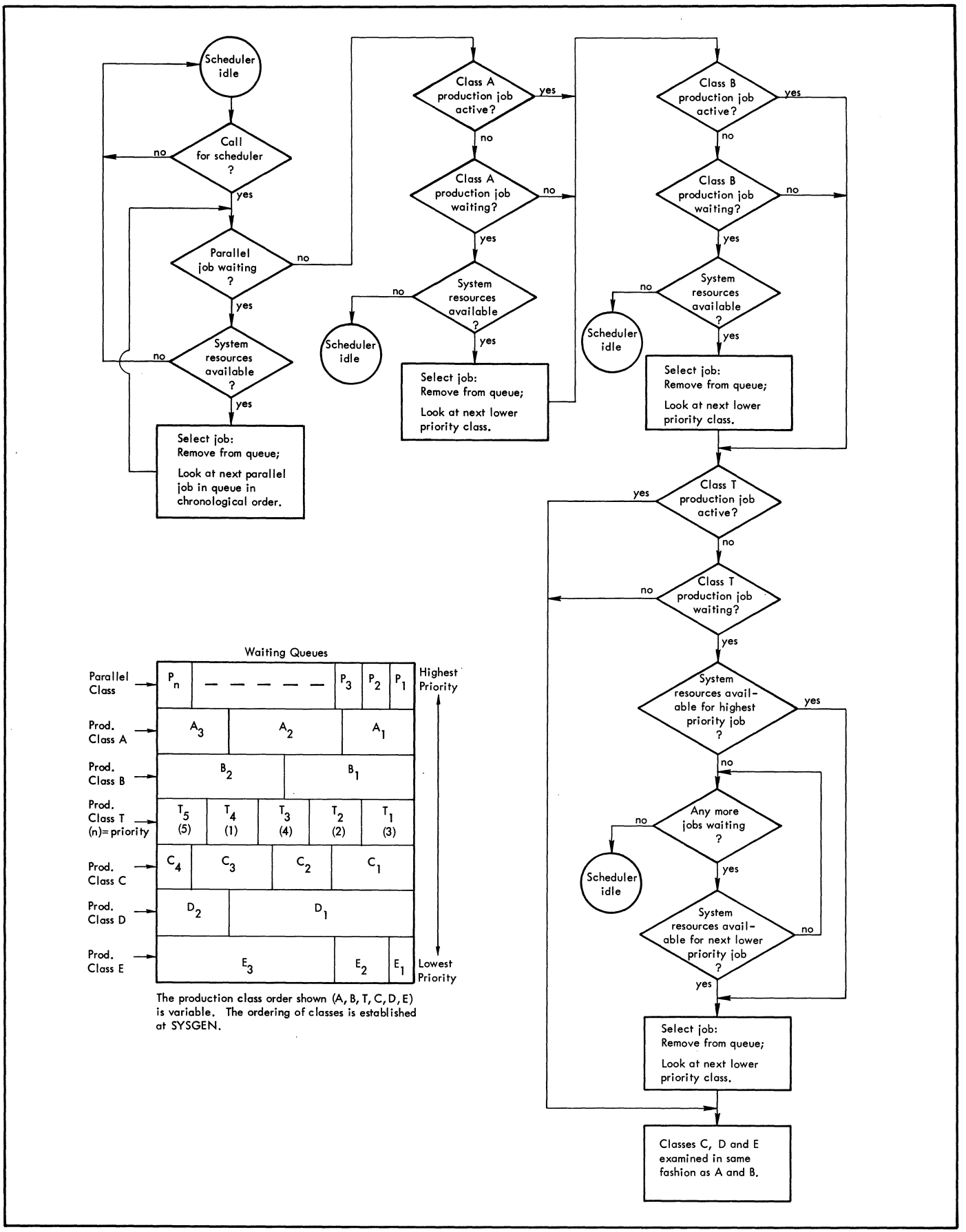


Figure 1-4. XOS Job Scheduler – Scheduling Flow Example

2. SYSTEM FACILITIES

INTRODUCTION

XOS is an efficient and flexible multiprogramming system that maximizes data throughput of production jobs and minimizes turn-around time of high-priority jobs. The first function is accomplished by efficient sharing of system resources (CPU time, memory, secondary storage, and peripheral devices) among many jobs. The second function is made possible through the establishment of priority scheduling procedures. These installation-defined and hardware-implemented procedures assure that high priority jobs are processed first.

FACILITIES PROVIDED BY MONITOR

SYSTEM SERVICES

System services enable the user's program to request a variety of system functions. When a service call is encountered during the assembly or compilation of a program, the processor responds by retrieving a symbolic calling sequence from the XOS procedure library, modifying it according to the parameters specified in the procedure call, and inserting the modified instructions into the user's object program. At execution time, the calling sequence calls the appropriate system routine, which in turn performs the desired functions.

FILE MANAGEMENT SERVICES

The XOS File Management System (FMS) is comprised of a collection of system programs responsible for the movement of data between memory and external storage for user programs and system tasks. These programs provide the facilities to locate data, manage buffers and external storage, read data, and write data.

FMS provides a comprehensive set of services to coordinate the transfer of information between user programs and data files:

- FMS handles all types of physical files consistent with the I/O devices on XOS systems. These include unit-record devices, magnetic tapes, disk packs, and RADs.

- For magnetic tapes and disk packs FMS handles all combinations of single or multiple volume files or multi-file volumes.

- FMS handles both standard and nonstandard labels on magnetic tape. The standard tape label is ANS compatible. For nonstandard labeled files (user labels), the entire volume is treated as data.

- In order to achieve maximum flexibility, FMS supports a variety of file organizations and record formats. File organizations include sequential, indexed sequential, direct, and partitioned. Record formats include fixed, variable, and undefined lengths. Fixed and variable formats on tape are ANS compatible.

- FMS provides file-sharing and file protection functions. Shared files may be read by several tasks or processes concurrently. However, in order to write on a shared file, the user must obtain exclusive use of the file. A shared file may be protected by the file owner against unwarranted access. This protection is achieved by means of a password specified at the file's creation and by a list of users who are authorized to read or write the file.

- FMS permits tape file concatenation. This facility enables the user to logically connect several data files into a single consecutive file. FMS will automatically process from the end of one file to the start of the next file without any intervention from the user.

XOS provides facilities for six different methods of file processing, referred to as access methods. These access methods are divided into two groups according to the general techniques involved in their use.

The assisted access methods operate at the logical record level and are characterized by a high degree of system service and control: record blocking/deblocking, error checking, volume switching, etc. They are

- Assisted sequential access method (ASAM), intended for the creation and sequential processing of files on any type of media.

- Assisted indexed access method (AIAM), intended for the creation and direct-access processing of indexed files.

- Assisted partitioned access method (APAM), intended for the creation and processing of files that are segmented into partitions.

The basic access methods operate at the physical record (block) level and are characterized by a high degree of user control and relatively little system intervention. They are

- Virtual sequential access method (VSAM), intended for the creation and sequential processing of files, at the block level, on any type of media.

- Virtual direct access method (VDAM), intended for the creation and direct-access processing, at the block level, of files on direct-access storage media.
- Basic direct access method (BDAM), intended for access to a private or unlabeled direct-access volume by relative sector addressing.

COMMUNICATION MANAGEMENT SERVICES

The Communications Managements Services (CMS) is that portion of XOS that controls all remote communications. It includes TAM, the telesymbionts, and the communications interrupt processors.

TELECOMMUNICATIONS ACCESS METHOD (TAM)

TAM provides the applications programmer with the means for complete control of a communications network, through use of a set of specialized system procedures (macros). The objective of TAM is to relieve the user of the burden of handling actual data transmission and to allow him to work independently of the transmitting device or activity.

REMOTE BATCH PROCESSING & TELESYMBIONTS

Remote Batch Processing is an integral part of XOS and is made possible by the telesymbionts. The telesymbionts are system routines that read programs, data, and control messages from, and send program results to, remote terminals. Telesymbionts are a logical extension of the central site symbionts (i.e., they move data between slow remote peripherals and fast central site RAD and disk pack devices). The RAD and disk pack files may then be accessed by the control and processing elements of the system, taking full advantage of the high transfer rates of the RAD and disk pack devices. Jobs entered through the telesymbionts are treated like those submitted through the central site symbionts. Accounting information for remote batch jobs is reported to the user on his remote printer and is entered into the system's accounting log.

COMMUNICATIONS INTERRUPT PROCESSOR

The Communications Interrupt Processor (in the I/O supervisor) handles the actual hardware interface between TAM and the remote terminal as directed by TAM.

FACILITIES PROVIDED BY PROCESSORS

The standard XOS processors are divided into language processors, service processors, and utility processors.

The language processors are those elements of the system that accept a user's symbolic source-program code and translate it into a standard object language form.

The service processors are those elments of the system that assist the user in

- Preparing object-language program elements for loading and execution.
- Debugging, or trouble-shooting, of executable program.
- Sorting/merging of data file.

Utility processors operate on user files, supplementing the file management facilities of the monitor.

LANGUAGE PROCESSORS

The following language processors are offered to assist computer users with varying background in preparing tasks on Sigma 6/7/9 computers

- XOS ANS COBOL
- XDS Extended FORTRAN IV
- XDS Meta-Symbol

XOS ANS COBOL

XOS ANS COBOL is based on the American National Standards Institute (ANSI) definition of the language. The compiler's design permits rapid compilation of source code as well as generation of efficient object code for minimum program execution time.

The following features summarize the advantages of XOS ANS COBOL as an important tool for the programmer:

- Maximum use of high-speed secondary storage (RAD/disk pack file).
- Overlay organization for minimal core memory usage during compilation.
- Table handling feature permits tables up to three dimensions.
- Extensive diagnostic information and cross reference listings.
- High compilation and execution speeds.
- Extended language features.

XOS ANS COBOL greatly increases programmer productivity. Less time is required to produce a given program or

series of programs than is needed when various different programming languages are used. Time is saved not only in the development and debugging of the original programs but also in modifications and maintenance.

Because of the language structure, systems and logic changes can be made easily and rapidly in programs prepared in XOS ANS COBOL. The required changes fall into clearly defined sections of the source code and do not necessarily cause the entire program to be revised. XOS ANS COBOL accepts compressed (encoded) source input and can generate compressed source output. This feature reduces source-program storage requirements and maintenance effort.

XDS EXTENDED FORTRAN IV

XDS Extended FORTRAN IV is a comprehensive algebraic language that provides the user with a powerful and easily learned method of stating computational processes. This system conforms to the standard version of FORTRAN IV defined and maintained by ANSI. It has been extended to provide more complete capabilities and fewer syntactic restrictions.

XDS Extended FORTRAN IV is compatible with other FORTRAN systems and contains (as subsets) most other FORTRAN languages.

The XDS Extended FORTRAN IV processor makes optimum use of storage during compilation and for generated machine instructions. Compiler tables are dynamically assigned and readjusted during compilation to provide maximum use of available space. Intermediate output from the compiler is retained in storage as long as space is available, thereby eliminating unnecessary I/O operations. These and other advanced compiler techniques provide exceptionally rapid compilation of source language programs.

XDS Extended FORTRAN IV includes many features not found in many other FORTRAN compilers:

- Control of specialized I/O and interface equipment by using in-line assembly language instructions.
- Compiler generates reentrant object code.
- Conditional compilation allows a single copy of a program's source statements to be maintained for both debugging and production purposes

XDS Extended FORTRAN IV accepts source input and generates source output program files in a compressed (encoded) format. This significantly reduces source program storage and greatly eases the effort required to update source program files.

XDS META-SYMBOL

The Meta-Symbol assembler is a multipass, high-level language assembler. The most important advantage offered by Meta-Symbol is its powerful procedural (PROC) capability. Procedures (bodies of code similar to subroutines) have their effect during the assembly of a program rather than during its execution. Particularly powerful is the ability of Meta-Symbol procedures to generate instructions conditionally based upon their calling statements. For example, a single procedure may add a series of numbers when called in one manner and average a series of numbers when called in another. Procedures correspond somewhat to macros in the sense that, when called, machine instructions are generated in-line based upon the arguments that are provided. Procedures of general value can be placed into the system library and be automatically retrieved at assembly time.

Other Meta-Symbol features include the following:

- Full use of lists and subscripted elements in procedure references.
- Argument fields can contain both arithmetic and Boolean expressions, using constant or variable quantities.
- Command procedures permit generation of many units of code for a given procedure call line.
- Procedures can be nested and one PROC may call another.
- Symbol tables and concordance listings.
- Comprehensive error messages during assembly.

The ability to accept and generate compressed (encoded) source similar to that described for COBOL and FORTRAN is also provided with Meta-Symbol. This reduces source program storage and eases source program maintenance procedures.

SERVICE PROCESSORS

LINK EDITOR

The Link Editor (Link) is an XOS processor that transforms program modules in object-language format, as produced by compilers, assemblers, and other language processors, into a loadable and executable form (load module). Optionally, it will transform object modules into library modules, a specialized form for convenient storage of commonly used program elements, such as generalized subroutines. Link provides for the following kinds of flexibility in the production of complete, functioning programs:

- The combination of several separately compiled or assembled object modules, each constituting a subdivision of a functional program, into a single load module.
- The inclusion of pre-prepared library subroutines in a load module, along with one or more object modules.

- Overlay structuring of the resultant programs, allowing conservation of memory space.

- Modification of load-module code during link editing, allowing fewer recompilations or reassemblies.

DEBUG PROCESSOR

Debug aids assist in locating program errors. They may be activated by either the XOS control command language or by the program itself using XOS system services.

To activate debug services through the XOS control command language, the DEBUG processor is invoked by appropriate commands placed into the control job stream. Orders corresponding to the commands are incorporated, by the DEBUG processor, into the executable load module in the form of calls to the appropriate XOS system services.

To activate debug services from the program, calls to the required system services are placed by the user at the appropriate locations in the program.

XOS SORT

XOS SORT, in standard form, is a generalized parameter sort program that utilizes either magnetic tape or disk for intermediate files. Its input and output files are device independent; sort input and output can, alternatively, be dynamically supplied and received by a concurrently executing COBOL or Meta-Symbol program. In overall structure, it is a polyphase sequential sort, employing backward reading of intermediate files, thus providing optimal performance as either a tape or disk sort.

SORT can be employed as an overlay-structured subprogram of a COBOL or Meta-Symbol program. In addition to the standard-form sort, more specialized sorts can easily be generated by each installation, incorporating application-specific parameter defaults and user-developed input, output, and duplicate-record handling routines.

UTILITY PROCESSORS

To assist the user in the overall maintenance of his files, XOS provides six file/device oriented utility processors that supplement the facilities of the File Management System. A seventh utility allows convenient generation of test files via a COBOL-like language. (Detailed information is provided in the XOS Utilities Reference Manual, 90 17 69.)

PREP – VOLUME PREPARATION

PREP prepares an XOS standard volume for subsequent processing by writing a standard tape volume label on magnetic tape or a standard disk volume label and volume catalog on disk packs.

During magnetic tape preparation, PREP does not check for existance of old labels. It writes a new volume label followed by a tape mark, rewinds the tape, reads the just-written label, and displays this label on the printer.

Disk preparation consists of writing a standard volume label and the volume catalog. The size of the catalog is specified by the user and may contain 64, 128, or 256 file labels. The catalog is intialized with empty file labels and two special file labels :FDY and :PIL. File :FDY represents the catalog itself and :PIL represents the space on the volume.

PREP checks disk packs for existing labels and displays the old labels. PREP requests operator verification before overwriting a disk label and catalog. For tape, this verification is not performed to prevent overwriting.

FMGE – FILE MAINTENANCE

FMGE provides the user with eight major services: COPY, SAVE-RESTORE, COMPRESS, DISPLAY, INCLUDE, EXTRACT, LIST, LISTCTG, and DELETE. These services may be performed on file segments, individual files, or groups of files from a device. FMGE operations always include at least one magnetic device; at the time of file creation or save, it may use a card reader or printer. FMGE services are described as follows:

- COPY – Permits copying a file from one device to another (only magnetic devices). The structure of the copy is the same as that of the original file. The user may, at his option, delete the original at the end of the copying process freeing the space occupied by the original file.

- SAVE-RESTORE – With SAVE, the user may save a tape or disk file on tape or cards. Either single files or groups of files may be saved. The entire contents of a disk file may be saved by specifying the files FDY or PIL (discussed under PREP, above) are to be saved. PIL implies the entire disk will be saved. In specifying FDY, the user may also specify an expiration date. Then, only the unexpired files are saved.

- RESTORE – Rebuilds files, in original format, that are saved by the SAVE function. Saved files are directly processable only by RESTORE. They are intermediate files destined to be processed by RESTORE.

- COMPRESS – Allows the user to create on magnetic tape or disk a standard format compressed file for symbolic cards or card images.

- DISPLAY – Displays on the line printer or card punch the contents of a compressed file.

- INCLUDE – Creates a member of a partitioned file from a sequential file. The user may specify both a keyword and synonyms by which the new file is to be known.

- EXTRACT – Creates a sequential file from a partitioned file member. The user has the option of deleting the partitioned file segment at the end of the operation.

- LIST – Lists files in hexadecimal form.

- LISTCTG – Enables the user to list the contents of a catalog on a printer. From the catalog, the following can be obtained:

  - o Available space on the disk.
  - o Number of files he may create.
  - o Number of secondary blocks available in the secondary part of the catalog.

  The following items are displayed for each file:

  - o Identification (name, number, absolute generation number, version number, creation and expiration dates).
  - o Size (in quanta and blocks, 8K bytes per quanta).
  - o File organization.
  - o Format.
  - o Block length
  - o Record length.
  - o Key length and position of key in the record.
  - o Number and address of the last block.
  - o Number of overflow blocks.

- DELETE – Enables a user to erase his files regardless of the expiration date.

REORG – FILE REORGANIZATION

REORG comprises two processors: REORGI, to reorganize indexed sequential files and REORGP, to reorganize partitioned files.

In the frequent updating of an indexed sequential file, the user normally creates overflow blocks and blocks containing deleted information both of which increases file access time. The REORGI processor builds a new file from the old by incorporating the overflow blocks and not copying the deleted blocks. The reorganization is generally done in two phases. In the first phase, a sequential file is made from the old indexed sequential file. Deleted records are not copied and overflow blocks are inserted. The sequential file is then used to build a new indexed sequential file.

The processor REORGP is used to reorganize partitioned files. Reorganization consists of

1. Discarding partitions marked as deleted in the partitioned file's directory.
2. Resorting the directory in ascending order.
3. Reordering the remaining partitions based on the order of the newly sorted directory.

Reorganization eliminates wasted file space and reduces file access time.

GENER – MEDIA CONVERSION PROGRAM GENERATION

The Media Conversion Program Generator (GENER) provides routines that simplify the movement of data between any combination of card, tape, disk, and RAD devices. The data conversion is a two-step process (i.e., conversion-program generation and actual execution of the program generated).

In the program-generation step, the user supplies control commands describing the formats of the input and output files, the types of devices to be used, the data translation tables to be used (if any), and indicates the presence of any user-supplied routines to be used for the processing of non-standard labels. GENER then uses these parameters to generate object modules ready to be processed by the Link Editor. GENER also prepares a file describing the tree structure to be used by the Link Editor in selecting conversion program modules from the GENER library of object modules. The Link Editor is then called to form a loadable program from the modules.

To execute the generated conversion program, the user must define the input and output files via !ASSIGN commands and must provide a !RUN command. The user may execute his conversion program immediately, or he may catalog the !ASSIGN and !RUN commands for a later execution of the program.

DEFG – FILE GENERATION GROUP DEFINITION

The DEFG processor allows the user to define and maintain a set of files as a relative generation group. This permits the system to perform certain of the functions of job setup and volume retention normally done manually; functions can be quite burdensome for certain applications run on a cyclical basis: daily, weekly, and monthly runs of an accounts receivable program, for instance.

If a file generation group is established, the system can "keep track" of the volume serial numbers associated with specific generations of a given file and optionally, can decide which of the volumes retained as backup to a current file may be reused. This capability of XOS frees the user of the need to supply, for each run of a program, the volume serial numbers to be assigned to specific files, and relieves him of the responsibility of deciding when each of the volumes associated with an application becomes available for reuse.

Through use of DEFG, the user initially establishes the set of storage volumes to be used for a relative generation group, and subsequently can modify the set as the need arises.

GEF – TEST FILE GENERATOR

Preparing adequate test files for testing large, new, data-based programs can often be a difficult problem. The Test File Generator (GEF) helps ease this problem by assisting the user in generating test files. Through control commands, the user defines the files he wishes to build. His description will include the block and record length for the file. The record may be subdivided into fields and these fields may, themselves, be further subdivided. The user also describes the types of data that will eventually fill the fields. The fields may contain EBCDIC, binary, single and double precision floating point, and packed decimal data. The manner in which data fields are described follows very closely the form used in the PICTURE clause in COBOL.

Once the file format has been established, the user then supplies the data values to be used in actually building the file. The statements used in filling record fields with data values allow the use of iteration techniques. This enables the user to initialize many records through the repeated execution of a few GEF statements. As a check on the files being built, the user may request the records be listed on the line printer along with a listing of the GEF commands used to define the file.

SYSTEM GENERATION

The System Generation process (SYSGEN) produces a working XOS system that supports an installation's hardware configuration and its software requirements. The generation process consists of

- Adapting the monitor's program modules to the installation.
- Linking the modules into elements and overlays.
- Relocating language processors and other programs supplied with the system.
- Writing a loadable image of the new system on tape, to be known thereafter as the system tape.

The results of system generation are similar for most operating systems. However, the method of generating a system varies extensively. Many systems require SYSGEN to be one continuous process from start to finish. In XOS, a system may be generated in one continuous process or may be generated as a series of related but distinct steps. Also, generation may be performed under either a full sized existing XOS system or a smaller, skeletal system (in the case of a new installation). At many points in the SYSGEN process, it is possible to generate "save" tapes. These tapes are generated by the FMGE processor and enable the user to save the current status of the system generation. These tapes may be used as restart points if an installation's requirements should change. This saves the user from restarting from the beginning any time a failure occurs (even minor failures) during the generation process.

3. JOB CONTROL

INTRODUCTION

The XOS monitor receives job descriptions from control commands. They are a means of communication between user and system; they describe the sequence of the different steps comprising a job. Data may be placed after each step for use during execution of that step.

The collection of control commands and data forms a job, which is entered into the system by the input symbiont. Job initiation does not necessarily occur in order of presentation to the input symbiont. The system schedules jobs and job steps by job class, by user-assigned priorities (T class), and by required resources, as described by the control commands.

The XOS control command language consists of fourteen commands.

!JOB	!SWITCH
!RUN	!TITLE
!EXEC	!MESSAGE
!LIMIT	!COMMENT
!SLIMIT	!processor-call
!RESOURCE	!DATA
!ASSIGN	!EOD

General characterisitcs common to all commands are described in the following section. Subsequent sections discuss syntax and options for each command and present examples.

GENERAL SYNTAX

Syntax rules common to all XOS commands are presented herein. Commands have the following general form:

!command option field comment field

where

! is unique to the job-control commands; the system recognizes such commands by this character and requires it to be placed in character position 1.

command is an alphanumeric keyword identifying the command. It must begin in character position 2 and ends with a sequence of one or more blank characters with a maximum of 12 characters in length.

option field is a list of elements, separated by commas, which specifies the parameters of the command. No blank characters are permitted in this field. An element of the list may take one of the following forms:

1. A sublist having the same form as the option list and placed between parentheses.

2. An alphanumeric keyword recognizable to the system.

3. A value –

   - Decimal integer: the maximum size of which is determined by the requirements of the particular command

   - Symbol: a sequence of alphanumeric characters, the maximal length of which is specified by individual command requirements. Acceptable characters are the 26 letters, 10 digits, minus sign, and colon

   - Character string: a sequence of any characters of the set defined by a specific command (e.g., OPTION parameter of !RUN). When such a sequence appears in the option field, it must be the last element of the option list.

   - Option (i.e., a symbol, as defined above) limited to 18 characters.

4. A symbolic parameter (i.e., a symbol, as defined above) has a percent sign (%) as the first character and is limited to a total of nine characters.

The option field is terminated by a blank character or by the first semicolon, in the case of continuation.

comment field includes all characters following the option field.

SPECIAL SYNTAX NOTATION

Control command descriptions use a special syntax notation. An element shown as prefixed with a percent sign (%) may be made a symbolic parameter, allowing later substitution at the time of command set execution. (See "Cataloging" under !JOB Command).

CONTINUATION OF COMMANDS

The option field of a control command can extend over one or more records. The command to be continued must end with a semicolon (;). The subsequent record must begin in character position 1 with an exclamation mark (!) and may be followed by any number of blanks prior to the option field continuation. The continuation record may itself be continued, by the same rules.

PROCESSOR COMMANDS

The commands for calling the following XOS processors are explained in detail in the indicated chapter or manual:

Link Editing – This manual, Chapter 4.

Debug Aids – This manual, Chapter 5.

Utilities – XOS/UT Reference Manual, 90 17 69.

System Generation – XOS/SM Reference Manual, 90 17 66.

Compilers, Assemblers and other processors – Applicable processor manual.

The general syntax of a !processor-call command is described later in this chapter.

CONTROL COMMAND INTERPRETER (CCI)

The CCI processes commands read in by the input symbiont and creates a file that is used later for scheduling and control of the corresponding job. The file created by CCI contains the information in a format suitable for efficient scheduling. This information also becomes a convenient source of information for the process of mounting and dismounting volumes and releasing devices during job execution.

The CCI can also catalog sets of commands to be later called and activated as parallel jobs or to be called and effectively inserted into a job command set with an EXEC command. The syntax of each type of command is processed such that it is generally known if a string, decimal number, or delimiter is expected. Although it is an important part of the monitor, the CCI may be considered as a parallel job, and it is scheduled as such.

The Mini-CCI is required to initiate a parallel job. It uses the cataloged commands previously prepared by CCI when requested by the operator. Mini-CCI provides requests to the operator to allow substitution for previously defined symbolic parameters, if necessary. When the parallel job is ready (i.e., its required resources are available), it is scheduled with the highest job level priority. Mini-CCI may be considered as a parallel job and is scheduled as such.

!JOB COMMAND

The !JOB command must be the first command of every job. In the case of batch processing, it also defines the end of the preceding jobs except in the particular case when commands (!JOB included) are being used (read) as data (see !DATA command).

The !JOB command specifies the execution type expected by the user:

1. Syntactic analysis of commands and subsequent execution.
2. Syntactic analysis of commands without execution.
3. Syntactic analysis and cataloging of commands without execution.

The !JOB command must be present; if it is absent, the job is either considered to be part of the preceding job or ignored (in the case of the first job of batch processing).

Syntax

```
                                  ┌ job-id,account,user-id[,{D       }] ┐
!JOB[,class-id]                   │                        {priority}  │
                                  │                                    │
                                  └ name,account,(CATLG,{ADD}) ────────┘
                                                        {DEL}
                                                        {REP}
```

where

class-id is an alphabetic character identifying the class under which the job must be executed (ignored when using the CATLG or D option). The correspondence is as follows:

- A Class A production job.
- B Class B production job.
- C Class C production job.
- D Class D production job.
- E Class E production job.
- P Parallel job.
- T Class T production job (default option).
- $ Super job (predicate job link).

(See "Job Classes and Super Jobs".)

job-id is the job identity. This is the name by which the system identifies the job for execution (a maximum of 12 characters).

account is the user account number. This is an identifier consisting of a maximum of four alphanumeric characters, assigned to a user by the installation, which identifies the account under which the job is to be processed.

user-id is user identity. This is an identifier consisting of a maximum of 12 alphanumeric characters.

D is the DEBUG option. Requests syntactic analysis of subsequent commands without execution.

priority is the external priority for class T jobs only. This is a decimal integer from 0 to 15. The priority is increasing from 0 to 15. The default value is 15.

name is the name (maximum of eight characters) under which the command set is to be cataloged or under which such a set has already been cataloged.

ADD indicates the addition of a new set to the catalog of control command sets. The name (job-id) must not already have been allocated. ADD is the default option (see "Cataloging").

DEL indicates that a command set, already cataloged under the identifier of the current !JOB command, is to be deleted from the catalog of command sets.

REP indicates that a command set, already cataloged under the identifier of the current !JOB command, is to be replaced by the command set following the !JOB command.

JOB CLASSES AND SUPERJOB

All production class jobs are executed on a first-in, first-out basis within a given job class. Class T jobs may be prioritized with the priority option on the !JOB command, which can cause the first-in, first-out sequence to be overridden if priorities are different for individual class T jobs. Only one job per class is initiated at any given time. Overall, the job classes A, B, C, D, E, and T are initiated according to their relative priority which is assigned during system generation, and by the available resources of the system.

Class P jobs are initiated first-in, first-out and by available resources. They have the highest relative priority of all job classes, and if present in the job stream, are always initiated first.

Superjobs may be formed within the production job classes A, B, C, D, and E by the following method:

Given several jobs, each of which is dependent on the execution of a prior job or jobs, the first job should be given a normal class assignment (A, B, C, D, or E) and subsequent jobs to be linked should be given the predicate job assignment, dollar sign. The net effect of this scheme is that an abort of an upstream (predicate) job causes all downstream jobs to be aborted.

CATALOGING

XOS allows the cataloging of control commands into groups called command sets.

A set to be cataloged is syntactically analyzed and is then filed on the system disk in a specialized file.

The set can be retrieved for execution in two different ways:

1. Insertion into a set of commands. This execution is made with the aid of the !EXEC command. The command set to be executed can represent one or several job steps, or part of a job step.

2. Initiation of a parallel job from the operator control device. The cataloged command set must contain all the commands necessary to execute the same job in a production class. Such a job is then limited to a single job step as are all parallel jobs.

CONTROL COMMAND PARAMETERS

Certain elements appearing in the option field of a command may remain undefined until execution. These elements are indicated by a percent sign (%) prefix in the syntax. If the element is to be used as a symbolic parameter, an actual % must appear preceding the element in the command. If the % does not appear, the element is assumed to be an actual value.

The symbolic parameters are replaced by their effective values for execution from a list of equivalences given either by the !EXEC command, or by the operator in the case of a parallel job.

A parameter equivalence has the following format:

%parameter = actual element value.

If it is desired that the element default value be used instead of specifying an actual element, the expression is written

%parameter = % %

In any case, the format and the nature of the actual elements must be identical to what they would have been if they had been placed directly on the command card.

In this manual the elements which can be parameterized are indicated by the optional % prefix in the command syntax.

A single symbolic parameter can replace part of an option field or an entire set of sublist elements.

A single sublist element may be parameterized. The parameter symbolizes the entire sublist. For example, the VOL option of the !ASSIGN command has the syntax

(VOL,% serial-no ,...)

the serial-no sublist elements may be represented by only one symbolic parameter.

Incorrect usage:

(VOL,%A,V125)

Correct usage:

(VOL,%A)

Substitution for the above could be

%A = V124,V125

Example of parameterization of elements of different type:

The SIZ option of the !ASSIGN command has the syntax

(SIZ,%size,%incr)

Incorrect usage:

(SIZ,%C)

with implied substitution of the form

%C = 100,25

Correct usage:

(SIZ,%C,%D)

where the implied substitution would be of the form

%C = 100

%D = 25

Example 1: Class T Production Jobs

!JOB,T TRIAL1,XDS,DOE,10

This command indicates a job identified as TRIAL1 in class T; the job is to be executed under the account-id XDS, the user-id is DOE, and the priority level is 10 within the class T scheduling queue.

Example 2: Cataloging of a Job

!JOB TRIAL2,DSB,(CATLG)

The set of the commands which follow (up to the next JOB command), are cataloged under the name TRIAL2. This name must not already exist in the catalog or the job will be errored. The cataloged commands may be inserted into another job by means of an !EXEC card bearing the name TRIAL2, or may be executed as a single step parallel job by the operator (see XOS/OPS Reference Manual, 90 17 68).

Example 3: Replacing a Cataloged Job

!JOB CATID,ACCT,(CATLG,REP)
:
new command set
:

A previously cataloged command set, CATID, will be replaced by the new command set under the same identity.

Example 4: Deletion of a Cataloged Job

!JOB TRIAL3,V010,(CATLG,DEL)

The set cataloged under the name TRIAL3, will be deleted from the catalog.

Example 5: Superjob

!JOB,A FIRST,XDS,R:LAVER

commands/data

!JOB,$ SECOND,XDS,J:NEWCOMBE

commands/data

!JOB,$ LAST,XDS,K:ROSEWALL

commands/data

This set of commands shows the structure of a superjob, where job FIRST is the predicate for execution of jobs SECOND and LAST.

!RUN COMMAND

The !RUN command calls an executable load module into memory and initiates its execution as a job step.

Syntax

```
!RUN [%step][,(LMN,%module)][,(UNLESS;
!    ,%value,. . .)][,OPTION,%option,. . .]
```

where

step is a parameter of up to four alphanumeric characters which identifies job step for further reference by the !ASSIGN command OP suboption.

module is a parameter of up to 17 alphanumeric characters. It identifies the file containing an executable load module of the program to be executed. This file is assumed to be on the accounting volume (volume or pseudo volume for user account on which file information is maintained) of the job that uses it. If not, a !ASSIGN command with operational label LM must precede the !RUN command to specify the means of access to the file. If the option LMN is omitted, the system assumes that the user wishes to execute the last load module built during the current job by the Link Editor.

value is a decimal integer ranging from 2 to 31, designating a bit of the Job Switch Word (JSW). If (at least one of) the specified JSW bit(s) is set (i.e. is 1), the !RUN command is not executed.

option is a string of any EBCDIC characters (blanks not allowed) with a maximum size of 119 characters that is transmitted to a table, the first byte of which is a byte count of the number of characters actually in the string. A pointer to the string byte count word is given to the program module to be executed, at its initiation, through program register 2. The string may be any format with the exception of symbolic parameters (%) which must be standard (see "Control Command Parameters").

Example 1:

```
!JOB  TEST,XDS,SMITH
!RUN  (LMN,TEST31)
```

The job loads and executes the load module contained in the file TEST31 cataloged under account number XDS.

Example 2:

```
!JOB,B  MATRIX,421,PETER
!FORTRAN  LO, SI, GO
    FORTRAN source module
!LINK
!RUN
    data file
```

Example 2 will cause a FORTRAN program to be compiled (!FORTRAN), a temporary load module to be created by the link editor (!LINK) and finally the program loading and execution (!RUN). The data file will be read dynamically from the job input stream.

Example 3: Suppose that the following command set has been cataloged:

```
!JOB  FORT-GO,421,(CATLG,ADD)
!FORTRAN  LO, SI, GO
!LINK
!RUN
```

To execute this cataloged command set, the following commands may be used:

```
!JOB,B  MATRIX, 421, PETER
!EXEC  FORT-GO
    FORTRAN source module
    data file
```

The effect of this execution will be identical to that of the job described in example 2.

Example 4:

```
!JOB,T  EXECUTION, XDS, JOHN
!RUN  STP1,(LMN,ADVANCE)
    data file
!ASSIGN  LM,FIL,(STS,OLD),(UNT;
!    ,AC,103),(NAM,DELAY)
!RUN  STP2,(UNLESS,10,11,12)
    data file
```

The first step (STP1) will be the execution of the load module in a file named ADVANCE cataloged under the account XDS.

In the second step (STP2), the !ASSIGN command specifies that the load module to be executed is located in the file DELAY and is cataloged under the account 103. The second step will be executed only if bits 10, 11, and 12 of the JSW are 0.

!EXEC COMMAND

The !EXEC command executes a previously cataloged command set.

Syntax

```
!EXEC  %name[,(UNLESS, %value,...)] [,%list]
```

where

name is the identifier (maximum of eight characters) given to the cataloged command set during its cataloging (job-id).

value is a decimal integer ranging from 2 to 31 that designates a bit of Job Switch Word (JSW). The specified value(s) will be substituted in the UNLESS options of all the !RUN commands in the cataloged command set. (The values previously specified in the !RUN commands are not interrogated.) Thus, if at least one of the corresponding bits is set to 1 in the JSW, all the cataloged job steps will be skipped.

list is an equivalence list which, at execution, transmits actual values to the symbolic parameters declared during the cataloging. The list is a series of parameter sets, separated by commas, each set having the form

%symbolic-parameter = value

Example: Suppose the following job has been cataloged:

```
!JOB  DESIGN,XDS,(CATLG,ADD)
!SWITCH  (S,%KEY)
!ASSIGN  FIL1,FIL,(STS,OLD);
!    ,(UNT,MT,(VOL,%N));
!    ,(DSP,%T),(NAM,%NAME)
!RUN  (LMN,DESIGN1)
```

The command set following the job command is cataloged under the name DESIGN. The cataloged commands are invoked by a !EXEC command.

The following sample !EXEC command illustrates the types of parameter values that may be substituted for the symbolic parameters defined in the cataloged command set:

```
!EXEC  DESIGN,%KEY = 10,11,12;
!    ,%N = 12, %T = %%,;
!    %NAME = DATA
```

The commands effectively executed are:

```
!SWITCH  (S,10,11,12)
!ASSIGN  FIL1,FIL,(STS,OLD),;
!    (UNT,MT,(VOL,12)),;
!    (DSP,DMT),(NAM,DATA)
!RUN  (LMN,DESIGN1)
```

Note that the DSP parameter (%T) assumes the default, which which is DMT.

!LIMIT COMMAND

The !LIMIT command transmits two types of information to the system:

1. It specifies the limiting values for the number of printed pages and punched cards to be produced by the output symbionts during the execution of a job.

2. It specifies the maximum amounts of the shared resources (CPU time, memory space, temporary disk space) that a job will require.

If these estimates are exceeded, the job is aborted.

There can be only one !LIMIT command per job, and it must immediately follow the !JOB command. All the parameters of the !LIMIT command are optional, as is the !LIMIT command itself. The system assumes default values for all parameters that do not appear; default and maximum values are established for each job class during system generation.

Syntax

```
!LIMIT  [(TIME%duration)][,(SPDISC,%quanta)];
!    [,(CORE,%space)][,(CARDS,%cards)];
!    [,(PAGES,%pages)]
```

where

duration is the maximum amount of execution time (in minutes) allocated to the job. The system initializes a timer to the specified value for each job. This timer is decremented only when the job takes

control of the CPU; decrementing stops each time the job loses control.

quanta is the space allocated to the job to create temporary files on the system disk. One quantum equals 8192 bytes.

space is the memory space allocated to the job for its execution. This space is expressed in memory pages (1 page = 512 words).

cards is the maximum number of cards that may be punched for the job on the SCP device (symbiont card punch).

pages is the maximum number of pages that may be printed for the job on the OUT and/or SLP devices (symbiont line printers).

If the values specified for the CORE or SPDISC options are exceeded during execution, the job is aborted unless a new allocation of higher value has been made for the current job step by means of an !SLIMIT card.

Example: Assume the following !LIMIT command:

```
!LIMIT (PAGES,200),(CARDS,100),(SPDISC,50)
```

The above command shows that if the job under consideration attempts to output more than 200 pages on the line printer or more than 100 punched cards, it will be aborted.

The job is given 50 quanta of temporary disk space (409,600 bytes). If this value is exeeceded, the job will be aborted unless a higher value has been given to the current job step by an !SLIMIT card. Because the TIME and CORE options are not specified, their values will default to values specified during system generation. Exceeding the default values will have the same effect as exceeding values actually specified.

!SLIMIT COMMAND

The !SLIMIT command redefines, for the duration of the job step to which it applies, the limiting values of CPU memory and temporary disk space.

If the !SLIMIT command is to be used, it must be placed immediately prior to the command that activates the job step (!RUN or !processor call). The job scheduler checks to see if the requested resources are available; if not, the job is placed in a wait state until resources become available at which point the job becomes active.

At the end of the job step, limit values return to those specified in the !LIMIT command, or to the default values of the system, unless a new !SLIMIT command specifies other values for the next job step. If the amount of resources in use at the end of a job step is higher than the value specified in the !LIMIT command (e.g., temporary disk files are using more quanta than specified in the SPDISC option), the job will be aborted.

Syntax

```
!SLIMIT [(SPDISC,quanta)][,(CORE,space)]
```

where

quanta is the space allocated to the job step to create temporary files on the system disk. One quantum equals 8192 bytes.

space is the memory space allocated to the job step for its execution. This space is expressed in memory pages (1 page = 512 words).

The !SLIMIT command is optional as each of its parameters is optional.

If the !SLIMIT command is used, resource limits not specified on the !SLIMIT command default to the corresponding limits on the !LIMIT command; if omitted in the !LIMIT command, the system default values are used. If the values specified on the !SLIMIT command are lower than the corresponding value on the !LIMIT command (or the job default value), the SLIMIT value is ignored.

The job is aborted if any of the resource limits effective for a particular job step are exceeded.

Example: Assume the two following commands within a job:

```
!SLIMIT (SPDISC,50)

!RUN STP3,(LMN,ABC)
```

Suppose that for this job the temporary disk space limit value was set to 30 quanta in the !LIMIT command. This value will be increased to 50 quanta at job step STP3. At the end of STP3 the limit will be brought back to 30 quanta unless a new !SLIMIT command specifies a higher value for the next job step.

If at the end of STP3 the temporary disk space actually in use is higher than the next limit value to be used, the job will be aborted.

!RESOURCE COMMAND

The !RESOURCE command specifies the minimum amounts of common resources (I/O devices) required for initiation of a job. The system default value is used for any peripheral type not specified in the !RESOURCE command; this value indicates the maximum number of units that may be simultaneously active during the job process.

The values shown on the !RESOURCE command may be exceeded during execution. However, the scheduler places the job in a wait state if the request cannot be fulfilled immediately until such time as resources are available, at which point the job will be placed in the active state.

If the value shown on the !RESOURCE command (or the default value) is greater than currently necessary for execution, the system will use the lower value actually required.

Syntax

```
!RESOURCE (yy,%number)[,...]
```

where

yy is a mnemonic of two alphanumeric characters designating a type of peripheral as follows:

MT	9-track magnetic tape unit
7T	7-track magnetic tape unit
DM	removable disk (7246/7242)
CP	card punch
CR	card reader
LP	line printer

number is a decimal integer stating the minimum number of peripheral units of type yy that must be allocated to the job before initiation.

The !RESOURCE command must be placed immediately after the !LIMIT command or, if the latter is absent, after the !JOB command. The !RESOURCE command is optional.

If one of the values on the !RESOURCE command is exceeded, the job scheduler puts the job into a wait state. This is done before execution of the step that would cause the excess demand. The job is reinitiated at that job step by the job scheduler as soon as the resources on which the excess demand was made are available.

Example: Assume the following command:

```
!RESOURCE (MT,2)
```

The job scheduler initiates the job as soon as at least two 9-track magnetic tape units are available, unless the job or job step actually requires fewer than two tape units.

!ASSIGN COMMAND

The !ASSIGN command allows the user to assign a logical file, identified by an operational label, to a physical file as described in the command. The logical file is defined by a data control block (DCB) in the user's program; associated with a given DCB is a one- to four-character operational label that is unique within the program. (Logical files and DCB's are described in detail in Chapter 6, where the DCB/!ASSIGN relationship is also further discussed under "!ASSIGN Command Usage." Creation and modification of DCB's is described fully in Chapter 7.) A maximum of 20 !ASSIGN commands is allowed per job step.

The !ASSIGN command defines the physical resource, i.e., the type of I/O device and, where applicable, the specific storage media volume(s), to which the operational label is to be assigned, either for creation of a new physical file or access to an existing physical file. Therefore, the processing program can be independent of specific I/O devices, and if carefully designed can be independent of I/O-device type also.

Many other characteristics of a physical file (besides the corresponding physical resource) may also be defined in the !ASSIGN command, such as file name, size, mounting mode (for multivolume files), end-of-processing disposition, etc. The assignment of an operational label allows the establishment of an association, at the time the DCB is opened (see Chapter 7, "Introduction"), between the DCB and the defined physical file. This permits the system to utilize, at execution time, the externally (!ASSIGN command) established characteristics of the physical file. Optionally, the user can also specify DCB parameters – defining aspects of the logical file – in the !ASSIGN command, which modify or complete the DCB when it is opened, analogously allowing the program to utilize at execution time externally defined characteristics of the logical file.

The use of predefined operational labels, for which the system will make an implicit assignment, is described later in this section under "Predefined Operational Labels".

SYNTAX

The full syntax of the four basic types of !ASSIGN commands is shown on foldout Chart A-1 in Appendix A. The four basic types of !ASSIGN commands correspond to

- File-type (FIL) assignments: labeled permanent files, or temporary files, on magnetic file media. (FIL options and DCB parameters apply.)
- Device-type (DEV) assignments: files on nonmagnetic devices, or unlabeled files on magnetic tape volumes. (DEV options and DCB parameters apply.)
- Indirect-type (OPL) assignments: essentially an indirect form of a FIL or DEV assignment, referring to another !ASSIGN command. (DCB parameters apply only.)
- Dummy-type (DUM) assignments: a null or "dummy" physical file assignment. (No options or parameters apply.)

The meaning of the options applicable respectively to the FIL and DEV types of !ASSIGN command, as shown on fold-out Chart A-1, are described separately below. Immediately following are descriptions of the two fields common to all four !ASSIGN command types, and of the OPL and DUM types of assignment.

COMMON !ASSIGN OPERANDS

The meaning of the two operand fields common to all forms of the !ASSIGN command are as follows:

op-label This mandatory field, which must appear in the position shown, specifies the one- to four-character operational label to be assigned in order to establish the characteristics of a physical (or dummy) file.

{MTN / FRE} MTN (maintain) specifies that the file assignment for op-label is to be maintained, i.e., is to remain in effect, through subsequent job steps until replaced or freed by another !ASSIGN command specifying the same op-label. (The defined physical resource, if any, remains allocated until such time or until end-of job.)

FRE (free) specifies that the assignment for op-label is to be effective only for the job step with which the command is associated; the association is "freed" at the end of the job step, and the corresponding physical resource, if any, is released unless the FIL-option DSP, RET is specified.

If neither MTN or FRE is specified, FRE is assumed by default. (Exception: see "OPL-Type Assignment" below.)

OPL-TYPE ASSIGNMENT

The indirect type of assignment, distinguished by the keyword OPL, is simply a reference to another active !ASSIGN command. The op-label-1 field identifies the primary operational label (op-label field) specified in another !ASSIGN command that either appears in the same job step or is still active from a previous job step (with MTN option). The primary operational label specified in the referencing command thus takes the physical file assignment made in the referenced command. (DCB parameters of the reference command do not apply.)

The OPL type of !ASSIGN command essentially allows for the creation of a synonym of an operational label assigned with another command. This allows multiple op-label assignments to a given volume/device. Were each assignment made separately, a separate resource request would be implied by each assignment, i.e., several devices would be allocated for a single volume. This situation would be wasteful of the system's I/O resources.

In the syntax

1. !ASSIGN op-label,OPL,op-label-1

op-label effectively defines a synonym of op-label-1. The assignment of op-label is effective for one job step only; assignments made via OPL cannot be maintained. The original assignment of op-label-1 remains in effect if MTN was specified therein.

Optionally the user creates a synonym operational label and also frees the referenced operational label (at the end of the current job step) by the use of the syntax

2. !ASSIGN op-label,FRE,OPL,op-label-1

Note that in this form FRE effectively refers to op-label-1, since the op-label assignment is effective only for the current job step in any case.

SPECIAL CASE OF THE OPL-TYPE ASSIGNMENT

A special case of the OPL-type !ASSIGN command allows the user to free a previously assigned and maintained operational label at the end of the current job step. The syntax is

!ASSIGN op-label,FRE

This is the default case of form 2 above, where OPL is the type-keyword default and op-label-1 is assumed by the system to be identical to op-label.

DUM-TYPE ASSIGNMENT

The DUM type of !ASSIGN command assigns the operational label to a "dummy" or simulated physical file; no actual I/O device is defined for the operational label. An attempt to open the corresponding DCB for input, update, scratch, or backward operations results in an abnormal return (X1-class, abnormal code X'01'). The DCB can be opened normally for output, and output operations requested through the DCB are simulated but result in no actual I/O transmission.

MEANING OF THE FIL-TYPE OPTIONS

The meaning of the options applicable to the FIL type of assignment are given below.

STS introduces the status of the file being assigned:

NEW specifies that a new file is to be created. The system requires that the account number of the job be the same as the account number (if

any) associated with the volume on which the file is to be created, i.e., excepting public volumes. The system also verifies, at the time the DCB is opened, that no file of the same name already exists on the volume if direct-access media; if magnetic tape, the name verification is optional (see options available with M:OPEN).

OLD specifies that an existing file is to be read or rewritten. The file owner and other authorized users (see PRT option) can read the file, but only the file owner can rewrite it. If rewritten, only the file content is changed; the original logical and physical file characteristics as described in the file's labels are not modified.

MOD specifies that a file is to be modified (read, updated, and/or extended). The file owner and authorized users (see PRT option) can read, update, and/or extend the file, as authorized. For sequential files, only extensions are allowed. If the file does not exist at file opening time, the status evolves to NEW and processing is as described thereunder.

LNK specifies that the existing file currently being defined is to be linked (logically concatenated) to a previously defined file:

op-label-2 is the operational label of the first file of the linked sequence, by which label the entire set of linked files can be read (status OLD, input processing mode only). Any number of files can be so linked, but these files must be on magnetic tape and have the same physical characteristics. The order in which the !ASSIGN commands appear in the job step determines the order of linking and access.

UNT specifies the physical resource(s), i.e., device type and volume(s), on which the file exists or is to be created:

{DM / MT} defines a removable-volume device type; DM = 7242 disk, MT = 9-track magnetic tape.

VOL introduces the serial number(s) of the volume(s) to be mounted (a maximum of 12 per ASSIGN). Default = a public volume – if a permanent file is created thereon (NAM option specified), the volume automatically becomes private.

serial-no is a one- to six-character volume serial number.

SQN introduces a volume-list sequence number.

sequence-no is a positional index into the volume-serial-number list (see VOL above) that points to the number of the first volume to be processed in a multivolume sequence. For example: given a multivolume magnetic tape file comprising volumes A001, A002, and A003; if it were desirable to begin processing with A003 and bypass mounting of A001 and A002, the UNT option would be

(UNT,MT,(VOL,A001,A002,A003),(SQN,3))

This applies to sequential files only. The default value for SQN is 1.

AC introduces the account number corresponding to an account volume from which the file is to be accessed (in the mode or modes authorized by the account volume owner).

acct-number is a one- to four-character account number.

OP specifies that the file is on, or is to be created on, a device/volume that has already been defined and is still allocated to the job, the purpose being to prevent unnecessary mountings and dismountings.

op-label-3 is the operational label that identifies the !ASSIGN command in which the device/volume has previously been defined; this operational label need not be actively assigned in the job step in which it is referenced, i.e., it may be currently free.

step identifies the job step containing the !ASSIGN command identified by op-label-2; the job step identifier may be the !RUN command step parameter or the relative step number (first step = 1). If step is omitted, the current job step is assumed unless the referenced label is active, i.e., has been maintained from a prior job step.

The default for the entire UNT option for a permanent file (NAM option specified) is UNT,AC,acct-number where acct-number is the account number specified on the corresponding !JOB command; i.e., the user's own account volume is implied. The default for a temporary file (NAM option also omitted) is secondary storage – a portion of system disk space.

PAR specifies parallel volume mounting; i.e., all volumes are to be mounted concurrently on different physical devices. PAR is valid only for removable volume disk devices, for which it is the default option; PAR is mandatory for multivolume files to be processed by AIAM, APAM, and VDAM. Parallel mounting is not relevant to public volumes.

MNT specifies serial mounting of volumes; mounting and processing of the volumes will therefore be sequential. For magnetic tape files, MNT is the default, with one physical unit reserved for volume mounting:

number is the number of physical units to be reserved for mounting of some or all of a serial-mounted volume sequence.

DEF specifies deferred volume mounting. In this case, the system will not reserve additional physical units but will use the physical unit(s) assigned to another file by another ASSIGN command. The corresponding physical unit(s) must be available (i.e., not associated with an open or temporarily closed DCB) when the subject file is opened. Several deferred mountings may occur in sequence within a job step. The mounting type specified on the referenced ASSIGN command must be serial (MNT), and applies to all mountings deferred to it:

op-label-4 is the operational label of the referenced ASSIGN command; both ASSIGN commands must appear in the same job step. The volume assigned via op-label may not be used in a subsequent job step.

DSP introduces the option specifying desired volume disposition following closing of the DCB corresponding to op-label. The default for the DSP option is DSP, DMT:

DMT requests dismounting of volume(s). (Default option.)

KEP requests no dismounting of volume(s); they will remain mounted for possible subsequent use by other jobs or job steps but may be dismounted if the system needs the corresponding physical units.

RET requests that the volume(s) are not to be dismounted during the job unless a deferred mounting is requested on the same device(s). The corresponding device(s) will remain allocated to the job unless released by a subsequent deferred-mounting assignment.

SIZ introduces the size parameters of a file to be created on direct-access media (recognized for status-NEW files only):

RST specifies that the unused space remaining after file creation is to be returned to the system.

SEP specifies that the overflow area of a multi-volume indexed file is to be created on a volume separate from the volume(s) on which the base data blocks have been written. (Permanent, multi-volume, indexed-organization files only.)

size is the initial (C or P organization) or total (I or D organization) size of the file, expressed in quanta of 8192 bytes.

increment is the increment of space by which a sequential or partitioned file is to be extended when necessary (and possible), or the portion of the total space of an indexed file that is to be reserved for index and overflow blocks; both expressed in quanta of 8K bytes. (See Chapter 6, "Space Allocation.") This option is not recognized for direct-organization files.

NAM introduces the name (i.e., file identification) of the physical file to be created or retrieved; presence of this option implicitly defines the file as a permanent file:

name is the name of the file, consisting of 1 to 17 alphanumeric characters (see Symbol under "General Syntax" earlier in this chapter).

PRT introduces suboptions specifying the access protection, by means of account authorization control desired for a file to be created; and, implicitly, the nature of file sharability, if any (see "Volume/File Sharability" in Chapter 6):

NCT specifies no control of file access, i.e., that no account authorization control is to be performed. Use of NCT excludes use of any of the following PRT suboptions.

R introduces an explicit or implicit list of accounts authorized to read the file.

account is the account number of an account so authorized.

ALL implies all accounts are so authorized. ALL is the default for the R suboption.

NO implies no accounts are so authorized.

W introduces an explicit or implicit list of accounts authroized to write in (modify or extend) the file.

account is the account number of an account so authorized.

ALL implies all accounts are so authorized.

NO implies no accounts are so authorized. NO is the default for the W suboption.

PAS specifies that a password composed of any eight characters supplied by the creating program is to be attached to the file; if so attached, the password will be required of any authorized program (including the file owner's) attempting subsequent access. The password is supplied, either for file creation or for subsequent access, during opening of the file via abnormal condition processing. Failure to select the X1 abnormal condition class, or to supply the correct password, results in a program abort. (See "Password Protection" in Chapter 6, and "Processing of Abnormal and Error Conditions" in Chapter 7.)

The PRT option is recognized only for file creation, i.e., file-status NEW. The default for the entire PRT option is PRT,(R,ALL),(W,NO); that is, if PRT is omitted all other users have read access and no other users have write access. Note that unless the NCT suboption is specified, some degree of regulated access is implied.

RET introduces a value specifying the file's nominal retention period, i.e., a projection of the file's useful life span; this option is recognized for file creation (file-status NEW) only:

period specifies the retention period, expressed in number of days following creation. The default period is 0 days.

MEANING OF THE DEV-TYPE OPTIONS

The meaning of the options applicable to the DEV type of assignment are given below.

Option 1: A removable magnetic medium, specified by volume number, where

{MT / 7T / DM} defines a removable-volume device type; MT = 9-track magnetic tape, 7T = 7-track magnetic tape, DM = 7242 disk (BDAM only).

VOL introduces the serial number(s) of the volume(s) to be mounted.

serial-no is a one- to six-character volume serial number.

{PAR / MNT,number / DEF, op-label-4} specify parallel, serial, or deferred volume mounting, as for the FIL-type assignment.

Option 2: Any medium, specified by class of device, where

{CR / CP / LP / MT / 7T / DM} defines a class of input or output device; CR=card reader, CP=card punch, LP=line printer, etc. The operator will be asked to supply a logical device address when the device is needed.

Option 3: Any medium, specified by logical device address, where

ADR introduces the logical address of a specific I/O device.

logical-address is the logical address of the desired device, consisting of four characters of the form yyee; yy is designator of the device type and can be MT, 7T, DM, CR, CP, or LP; ee is normally a numeric designator of a specific device within that type (installation determined).

Option 4: A symbiont-controlled nonmagnetic medium, by symbiont name, where

IN specifies the symbiont card reader associated with the job control input stream. By use of this option the user program can obtain access to data files included in the job control deck. Refer to the !RUN, !DATA, and !EOD control command descriptions for information on defining such data files. Only one DCB associated with an IN assignment can be active at any one time within a job step.

OUT specifies the symbiont line printer associated with the listing log (job control listing output) for the job. Assignment of a file to OUT causes the user-produced records to be printed in the same print file as is the system-produced information, on a first-in, first-out basis. Only one DCB associated with an OUT assignment can be active at any one time within a job step.

SLP specifies a symbiont line printer other than OUT. Several DCBs each associated with an SLP assignment can be active in the same job step; for each such assignment the system creates a separate print file, each independent of one another and of job-control output.

SCP specifies a symbiont card punch. Several DCBs each associated with an SCP assignment can be active in the same job step; as with SLP the system creates a separate file for each such assignment.

NKP specifies that printing (SLP) or punching (SCP) is to begin after the job step has completed its output and closed the file (effected either by M:CLOSE...,RLS or by job step termination). If this suboption is omitted, actual printing or punching takes place as soon as the symbiont is able to acquire an LP or CP type device, as applicable. Not applicable to the IN or OUT symbionts.

STA introduces the identifying number of a remote terminal where printing (SLP) is to be performed.

terminal-id a two-digit identifier designating a specific remote terminal.

DCB PARAMETERS

The FIL, DEV, and OPL types of assignment allow the specification of a subset of the M:DCB/M:SETDCB procedure parameters; these are to be imposed on the assigned DCB at open time. See the descriptions of the M:DCB procedure, under the several access methods in Chapter 7, for the meanings and applicability of the parameters.

USAGE EXAMPLES

The following examples illustrate the sets of options applicable to specific types of files.

TEMPORARY DISK FILE

A temporary file to be placed in the temporary file area of secondary storage must be created; no device need be specified.

```
!ASSIGN op-label[,{MTN}],FIL[,(SIZ,size;
                 {FRE}
!    [,increment])][,DCB,parameters]
```

TEMPORARY REMOVABLE VOLUME FILE

Temporary file creation is requested on magnetic tape or removable disk. The file must be monovolume and the medium used must be a public volume.

```
!ASSIGN op-label[,{MTN}],FIL,(UNT,type);
                 {FRE}
!    [,(SIZ,size[,increment])][,(DEF,op-label-4)];
!    [,DCB,(parameters)]
```

PERMANENT FILE ON THE USER'S ACCOUNT VOLUME

This is a permanent file since the NAM option appears. In the absence of any specified media type (UNT option), it is located on the account volume corresponding to the account number under which the job is submitted.

For File Creation:

```
!ASSIGN op-label[,{MTN}],FIL,(NAM,name);
                 {FRE}
!    [,(STS,{NEW})][,(PRT,suboptions)];
            {MOD}
!    [,(RET,period)][,(SIZ[,(RST)],size;
!    [,incr)]][,DCB,parameters]
```

Note that the PRT, RET, and SIZ are recognized only when the file status is NEW.

For File Reference:

```
!ASSIGN op-label[,{MTN}],FIL,(NAM,name);
                 {FRE}
!    ,(STS,{OLD})[,DCB,parameters]
           {MOD}
```

PERMANENT FILE ON AN ACCOUNT VOLUME OTHER THAN THE USER'S

This is a permanent file since the NAM option appears. The UNT option is necessary in order to specify the account number of the desired account volume.

```
!ASSIGN op-label[,{MTN}],FIL,(NAM,name);
                 {FRE}
!    ,(STS,{OLD}),(UNT,AC,number);
           {MOD}
!    [,(DSP,suboptions)][,DCB,parameters]
```

Note that it is not possible to create a file on any volume, account or private, other than the user's own (i.e., one in his own account). It is possible to modify if so authorized.

PERMANENT FILE ON EXPLICITLY IDENTIFIED REMOVABLE VOLUME

This is a permanent file since the NAM option appears. The UNT option is necessary in order to specify the media type. In creation, if the serial number(s) of the volume(s) are not specified, the system uses public volumes which then become private volumes.

For File Creation:

```
!ASSIGN op-label[,{MTN}],FIL,(NAM,name);
                 {FRE}
!    ,(UNT,type[,(VOL,serial-no,...)]);
!    [,(STS,{NEW})],[({PAR          })];
            {MOD}    {MNT,number   }
                     {DEF,op-label-4}
```

```
!   [,(DSP,suboptions)] [,(SIZ,suboptions,)];

!   [,(PRT,suboptions,] [,(RET,period,)];

!   [,DCB,parameters]
```

For File Reference:

```
!ASSIGN op-label [,{MTN|FRE}],FIL,(NAM,name);

!   ,(STS,{MOD|OLD}),(UNT, type,(VOL;

!   ,serial-no,...)[,(SQN,sequence-no)]);

!   [,(LNK,op-label-2)][,({PAR|MNT,number|DEF,op-label-4})];

!   [,(DSP, suboptions)] [,DCB, parameters]
```

PERMANENT FILE ON REMOVABLE DEVICE DESCRIBED IN A PRECEDING !ASSIGN COMMAND

This is a permanent file since the NAM option is used. The UNT option is necessary in order to specify the operational label of the !ASSIGN command on which the device was described.

For File Creation:

```
!ASSIGN op-label [,{MTN|FRE}],FIL,(NAM,name);

!   ,(UNT,OP, op-label-3[,step]);

!   [,(STS,{NEW|MOD})] [,(DSP, suboptions)];

!   [,DCB, parameters]
```

For File Reference:

```
!ASSIGN op-label [,{MTN|FRE}],FIL,(NAM,name)

!   ,(UNT,OP, op-label-3[,step]);

!   ,(STS,{OLD|MOD})[,(LNK, op-label-2)];

!   [,(DSP,suboptions)][,DCB,parameters]
```

ACCESS TO MAGNETIC REMOVABLE-MEDIA DEVICES

```
!ASSIGN op-label [,{MTN|FRE}],DEV,{MT|7T|DM};

!   ,(VOL,serial-no,...);

!   [,({PAR|MNT,number|DEF,op-label-4})];

!   [,DCB,parameters]
```

DIRECT ACCESS TO NONMAGNETIC-MEDIA DEVICES

```
!ASSIGN op-label [,{MTN|FRE}],DEV,{CR|CP|LP};

!   [,DCB,parameters]
```

ACCESS TO PERIPHERALS REFERENCED BY ADDRESS

```
!ASSIGN op-label [,{MTN|FRE}],DEV,(ADR;

!   ,logical-address)[,DCB,parameters]
```

SYMBIONT ACCESS TO NONMAGNETIC-MEDIA DEVICES

```
!ASSIGN op-label [,{MTN}],DEV;
                  {FRE}

!       ,{IN                                          }
         {OUT                                         }
         {{SLP}[,NKP][,(STA,terminal-id)]}            };
         {{SCP}                          }

!   [,DCB,parameters]
```

CREATING SYNONYM OPERATIONAL LABELS

```
!ASSIGN op-label[,FRE],OPL[,op-label-1]

!   [,DCB,parameters]
```

DUMMY DEVICE SPECIFICATION

```
!ASSIGN op-label[,{MTN}],DUM
                  {FRE}
```

PREDEFINED OPERATIONAL LABELS

The system recognizes certain predefined operational labels. They must be defined during system generation and are referenced by the standard XOS processors. Therefore, the user does not have to include the corresponding !ASSIGN commands in his command set; but he may do so to redefine the implicit assignment made by the system for one or several steps of a job.

Table 3-1 shows the list of standard predefined operational labels and the corresponding implied !ASSIGN commands. Each installation may add to this list, and/or may change the assignments shown, to suit its individual needs.

Table 3-1. Predefined Operational Labels and Corresponding !ASSIGN Commands

Label	Corresponding Assign Command
SI	!ASSIGN SI,MTN,DEV,IN
LO	!ASSIGN LO,MTN,DEV,OUT
GO	!ASSIGN GO,MTN,FIL,(SIZ,3,1)
LM	!ASSIGN LM,MTN,FIL,(SIZ,3,1)
TREE	!ASSIGN TREE,FIL,(SIZ,3,1)

!SWITCH COMMAND

The !SWITCH command allows setting and resetting of each of the 32 bits of the JSW (Job Switch Word) associated with a job.

Bits 2 to 31 of the JSW may also be interrogated and set by the system procedures:

M:TSS, M:RSS, and M:SSS

(See Chapter 8.) The JSW bits are referenced by the UNLESS option of the !RUN and !EXEC command and by the :IF command (debug control).

Syntax

```
!SWITCH [(S,%v,...)][,(R,%v,...)]
```

where

S specifies that the bits of the JSW specified by the following values are to be set to 1 by the system prior to job step activation.

v is a decimal integer between 0 and 31 specifying a bit of the JSW

R specifies that that the bits of the JSW specified by the following values must be set to 0 by the system prior to the job step activation.

At job initiation all bits of the JSW are set to 0. Throughout job execution, setting and resetting of the JSW bits is controlled only by the user, either through the !SWITCH command or by program control. Switch settings remain in effect across job step boundaries and also across superjob boundaries. Bits 0 and 1 have fixed system functions. If a job step aborts and bit 0 is set, job execution continues with the next step. Otherwise the entire job is aborted. If a job step aborts and bit 1 is set, a postmortem dump is automatically taken. Bits 0 and 1 can only be set by the !SWITCH command.

!TITLE COMMAND

The !TITLE command specifies printing of a page heading at the beginning of each logical output page on the OUT file.

Syntax

!TITLE[,CONT] title

where

CONT specifies that page numbering is not to be reinitialized to 1 for the new title (i.e., for a change in title).

title is a sequence of any characters (including blank) which may not be continued to a subsequent command.

During a job the title changes each time that a new !TITLE command is encountered, but if several !TITLE commands appear in the same job step, only the last one is used. In addition to the title, the date and the logical page number are printed on the heading of each logical output page on the OUT file.

!MESSAGE COMMAND

The !MESSAGE command allows the sending of a message to the operator (on the operator control device) during execution of the job step in which it appears.

Syntax

!MESSAGE[,WAIT] message

where

WAIT specifies that job execution be interrupted after the message is sent to the operator. The operator can reactivate the job by using the INTERRUPT command.

message is any string of characters (including blanks), which may not be continued to a subsequent command.

One or more !MESSAGE commands are permitted per job step. However, a maximum of eight are actually output to the operator's control device per job step.

!COMMENT COMMAND

The !COMMENT command permits insertion of any kind of commentary into a job's control command deck. This commentary appears in the job's OUT file.

Syntax

!COMMENT commentary

where

commentary is the text of the command to be written on the job's OUT file. The commentary field may not be continued to another command.

Example:

!COMMENT THIS IS THE COMMENT

Any number of !COMMENT commands may be used in a job or job step.

!PROCESSOR-CALL COMMAND

The !processor-call command allows the invocation of programs cataloged under the system account number (compilers, assemblers, or possibly programs unique to the installation that are frequently used and adhere to the rules governing processors).

Syntax

!processor [option,...]

where

processor is the 3- to 12-character name under which the program file has been cataloged in the accounting volume (:SYS) of the system.

option is a string of characters that is pointed to (in register 2) at activation of the processor (see !RUN command OPTION).

Processor-call commands are equivalent to !RUN commands of the following form:

!RUN (LMN, processor)[,OPTION,option,...]

!DATA COMMAND

Within a job control command set the user may include data records that are accessible during execution through the IN device. Normally, the user cannot use data records beginning with !; the system would confuse it with a command marking the end of the data-record file. To avoid this restriction, the user may begin the data-record file with the !DATA command; he then must end it with a !EOD command.

Syntax

!DATA

The reading of a job file is performed automatically by the IN input symbiont; only EBCDIC encoded data is accepted.

A job step including the UNLESS option (see !RUN and !EXEC commands) cannot have input-symbiont data files.

!EOD COMMAND

The !EOD command allows the user to separate his data files with "end-of-file" indications. This is necessary because only one data file can be accessed at a time.

Syntax

!EOD

When the !EOD command is encountered, the system transmits an end-of-file indication to the user; the user must then close the current DCB to be able to access a subsequent data file. If he does not do this and attempts a new read operation, the job step is aborted.

The !EOD command may not be input as part of a data file.

The !EOD command must be used in conjunction with a !DATA command or !DATA will cause all subsequent control commands to be read, including !JOB commands, up to the next !EOD command.

Absence of both the !DATA and !EOD commands will indicate that the first encountered command (! in column 1) will signify "end-of-file" for the user.

4. LINK EDITING

INTRODUCTION

The Link Editor (Link) is a service processor that prepares language processor output for execution. In addition to the direct language processor output, Link incorporates other information (e.g., library subroutines) as necessary, according to the user's requests, and combines the whole to form a loadable and executable program module.

Compilers, assemblers, and other language processors generate object modules. Each object module is essentially a translation of a source module, which is a logical unit of code written in a source language such as COBOL, FORTRAN, or Meta-Symbol. This unit of code may be a functionally complete program (excepting possibly library subroutines), or may be a subroutine or subprogram that must be joined together with other program elements to become a functionally complete program. Essentially, the link editing process allows for this joining of separately compiled or assembled program elements into a functional program.

Object modules are the primary input to Link. They consist of a collection of records in Xerox Data Systems Sigma standard object language. Object modules are themselves not loadable or executable. They contain unresolved expressions[†] in many cases. They are generally relocatable in nature, i.e., they have no assigned origin in memory nor do they reference fixed virtual addresses. Object modules may contain symbolic references to locations in other modules that were undefined during the compilation or assembly; these are called external references. They may also contain symbolic definitions of locations within themselves so that other modules may reference. These are called external definitions. It is one of the possible tasks of Link Editor to establish a linkage between modules via their external references and definitions and to resolve incomplete object-language expressions containing such references. The resultant output of a Link operation is called a load module.

Load modules are, by definition, independent program modules, whose entire set of expressions have been evaluated and resolved; they are capable of being loaded into memory and executed by the system loader. The origin of an executable load module may consist of one or more (unedited) object modules and by one or more specially-edited modules that, although not executable, have been partially processed by Link. These modules have had their expressions partially resolved and their external definitions and references placed in a relocation dictionary (described below). The latter of these two origin types are called library modules.

Library modules are, as the name implies, normally referenced many times by object modules and other library modules; they are utilitarian in nature. They may be considered as subroutines or subprograms to a main program. The structure of a library module is basically the same as a load module, (i.e., they both originate from object modules). By itself, a library module is not normally functional. It usually operates only upon the request of the load module to which it has been linked.

Essentially, Link allows the user to maintain modularity in his programs. He may elect to have many object and library modules linked into one load module. He may call library modules explicitly by referencing the library module name. Or he may elect to have the Link Editor selectively link library modules by means of external references (in the object modules) to external definitions in the library modules. The first method is explicit reference (to other object modules or to library modules). The second method, implicit reference to library modules, is an unsatisfied reference search in which the editor actually searches a file of library modules for matching definitions.

Both object modules and library modules must be on magnetic-storage media before Link usage (see Table 4-1). There are two types of files to be considered. The first is the sequential file which may exist on either magnetic tape or disk. The sequential file consists of a continuous set of records that, as a whole comprises one or more object modules. The second type is the partitioned file. The partitioned file consists of a set of partitions, or subsections, each separately identifiable by a name called a key, and of a dictionary containing the keys. Partitioned files are unique to disk devices. They are described in detail in Chapter 6.

Object modules may exist on sequential tape or disk files as single modules per file or as multiple modules per file. Whether the object modules are single or multiple per file, Link interprets a sequential object module file as being one continuous object module. Object modules may also be on partitioned disk files in the form of one object module per partition.

Library modules must always be partitions of a file, one module per partition, although one library module may have been constructed from several object modules. The user actually has no choice in that Link automatically creates a library, and updates it, as a partitioned file.

Figure 4-1 shows the relationship of source, object, library, and load modules in the link editing process.

[†] An unresolved object language expression represented as a source language expression might be:

```
REF     A
LW,1    A
```

Where the value for A has not yet been established since it is part of another module.

Table 4-1. Link Edit Inputs and Outputs

Inputs	Default Operational Label	Allowed Operational Labels	Access Method	File Characteristics
Object Module (Sequential file)	GO	Any except LM,GEN†	VSAM	ORG=C, must be a disk file (temporary or permanent).
Object Module (Partitioned file)	GO	GO or GEN	APAM	ORG=P, GO is used with either temporary or permanent disk file; GEN is used with permanent disk file only.
Library Module	None	Any except LM†	ASAM	ORG=P, must be permanent disk file created by Link Editor as a library.
Control Commands :OPTION, :TREE, :REDEF, :MODIFY	SI	SI	ASAM	ORG=C, must be a card reader symbiont file.
Predefined tree-structure specification	TREE	TREE	ASAM	ORG=C, must be a disk file.
Outputs	**Default Operational Label**	**Allowed Operational Labels**	**Access Method**	**File Characteristics**
Executable Load Module	LM	LM	VSAM	ORG=C, must be a disk file.
Library Module	LM	LM	APAM	ORG=P, must be a disk file.
Diagnostics and Load Map	LO	None	ASAM	ORG=C, line printer symbiont file.

†:TREE card must be used to identify op-label.

SYSTEM INTERFACE

Load modules are always output by Link as sequential magnetic tape or disk files. Loading and execution of these modules is requested via the IRUN command (see Chapter 3). However, the user does have a means of controlling the selective loading and execution of specified portions of his program. He can direct Link to structure his program into several sections, known as segments. One segment, the controlling portion of the program, resides in memory throughout execution, and is called the root segment. At any given point in time, the other segments may either be resident in memory with the root of the program or maybe nonresident (stored on temporary secondary storage) for subsequent loading and execution. Loading and execution control of nonresident segments can be provided automatically by Link or can be achieved directly by user-programmed procedures.

A segment of a load module is defined as a portion of the module that is loadable as a separate unit and is executable or can be referenced by other segments of the load module. Program segments may be constructed from one or more object modules and from one or more library modules. The primary advantages of program segmentation are space conservation and program modularity.

Space conservation is achieved by indicating to Link that certain segments are not needed simultaneously during execution and may serially occupy the same (approximate) memory space. This process of sharing memory space is called overlay structuring, where segments sharing space are stored on disk and are loaded one at a time, as needed, into a common area of memory, each overlaying the preceding segment.

Program modularity is achieved by the user's capability of saving his separate object modules as permanent files. He may then elect to recreate one or more of these object modules for the purpose of program modification, and to use the new object module(s) plus the remaining unchanged modules

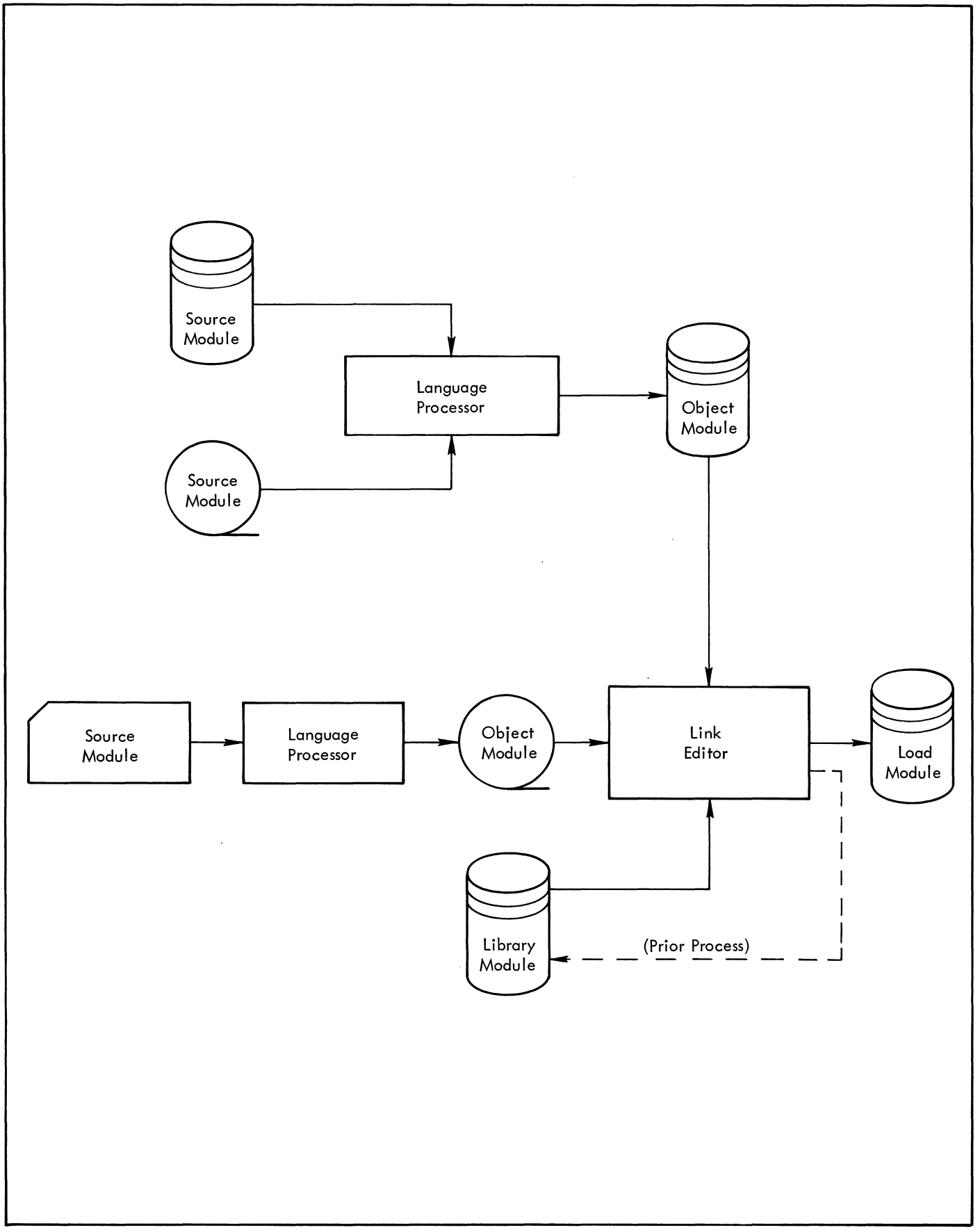


Figure 4-1. Source, Object, Library, and Load Module Relationships

to form a new load module. This process saves the user time and effort by eliminating the necessity of complete program reconstruction when only a small portion of the program actually needs modification.

An alternative provided by Link for effecting changes to a program is through its capability to directly modify load module expressions, insert new expressions, and redefine the module's externally defined symbols during the editing process. (The user should exercise care when modifying his executable code.) Another feature of Link is that, by default, it forms load modules in a (virtually) absolute form. That is, no relocation dictionary is included with the module. The relocation dictionary serves the purpose of maintaining a list of all external references and definitions within a module, leaving any expressions involving such references unresolved. An absolute module has already had all expressions resolved by Link and has no need of the space-consuming relocation dictionary. Consequently, a considerable amount of disk space is saved for the storage of load modules. Library modules are by default created with a relocation dictionary and must be so created.

The user does have the option of creating load modules in a relocatable form (i.e., with dictionary), which allows processing by the SYSREL processor, which is generally associated with system generation (see XOS/SM Reference Manual, 90 17 66).

Figure 4-2 illustrates the complete sequence of load module creation and execution.

LINK EDITOR COMMANDS

The Link processor itself is invoked by the job control command !LINK. All specific Link Editor processes are invoked and controlled by Link Editor commands.

The Link Commands are:

:OPTION	Indicates to Link which of its optional facilities are being requested by the user.
:TREE	Indicates the desired structure of the load module to be created.
:MODIFY	Specifies the instructions and/or data that are to be directly modified in the load module to be created.
:INSERT	Specifies the instructions and/or data that are to be inserted into the load module to be created.
:REDEF	Indicates the external definition symbols that are to be changed to new symbols by Link.

Each of the above Link commands may contain user-selected options which determine the specific course Link Editor will take in creating a load or library module. Those options are defined below, along with their default values as applicable.

!LINK COMMAND

The !LINK control command calls the Link Editor. It is generally preceded by one or more !ASSIGN control commands defining inputs and outputs. It is optionally followed by Link Editor commands, which specify the functions to be performed, in terms of

1. Modules to be link-edited.
2. Procedures for changing the content and definitions of the load module.
3. Structure of the resulting program.

Syntax

```
!LINK    comments
```

Note that the !LINK command itself has no options field.

General Link-command syntax follows the rules defined in Chapter 3, "Job Control" (section on General Syntax). The individual Link commands, their structure, and usage are defined below.

:OPTION COMMAND

The syntax of the :OPTION command is as follows; the command-continuation points (marked by a semicolon) indicated below are purely arbitrary, as is the ordering of the options. Both are strictly a matter of the user's convenience. Note that since all of the parenthesized elements of the option field shown below are optional, the square brackets indicating optionality are omitted at that level for the sake of clarity.

```
:OPTION (BIAS,address),(LIB,lib-module name);
:   ,(CREATE),(UNSAT,op-label[,...]), {MAP / NOMAP}
:   ,({SYSLIB / NOSYSLIB}), ({ABS / REL});
:   ,(SL,value),(START,external symbol),(TSS,value)
```

The default form of the :OPTION command, i.e., the options effectively assumed by the system if the command is omitted, is:

```
:OPTION (NOMAP),(SYSLIB),(ABS),(SL,7),(TSS,20)
```

Some of the options shown above are mutually exclusive, and some imply others by default, as is shown in Table 4-2.

Definitions of the :OPTION-command options and indications of usage are given below.

(BIAS,address) where address of this option is a hexadecimal value indicating the virtual address

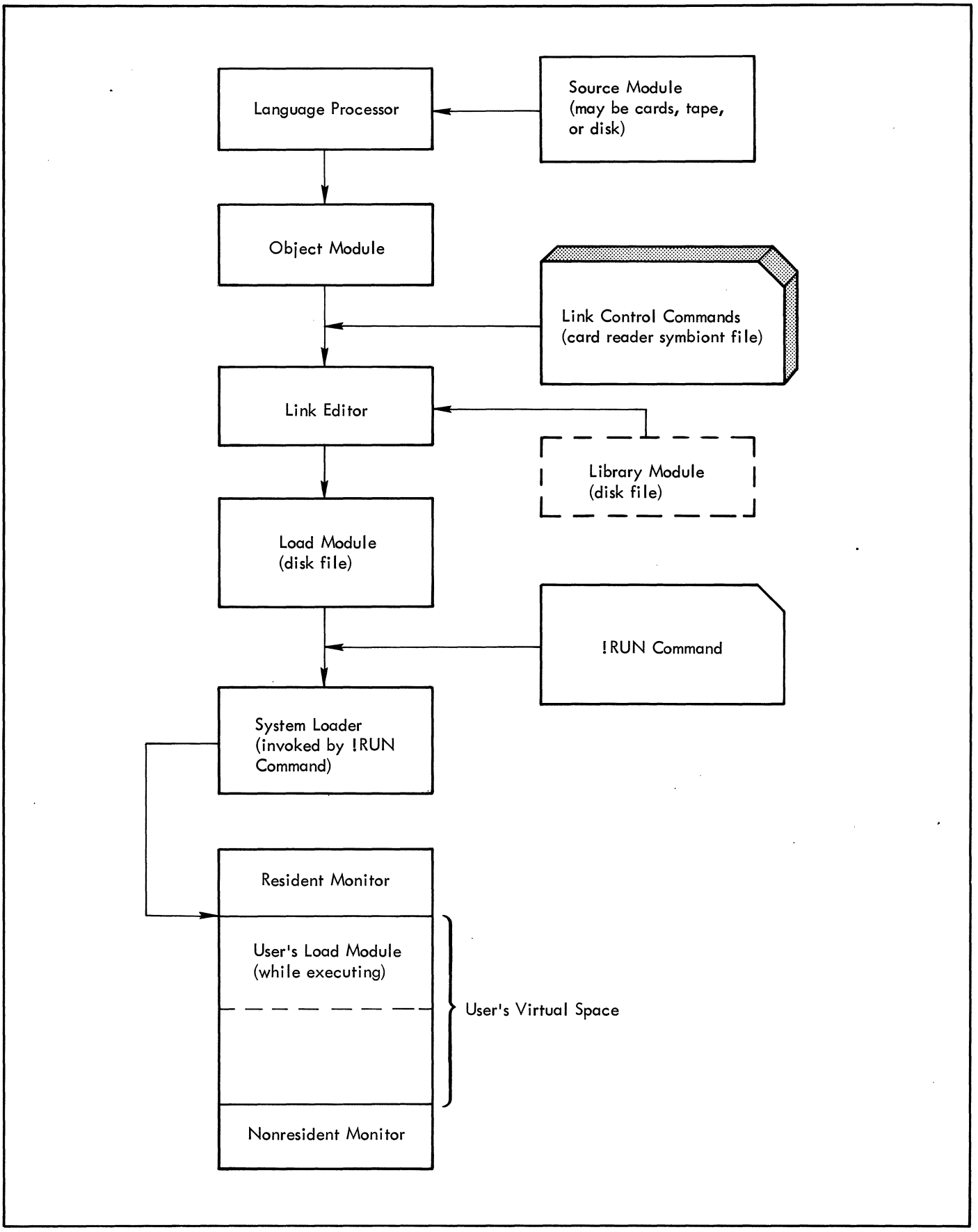


Figure 4-2. Generalized Load-Module Creation and Execution Sequence

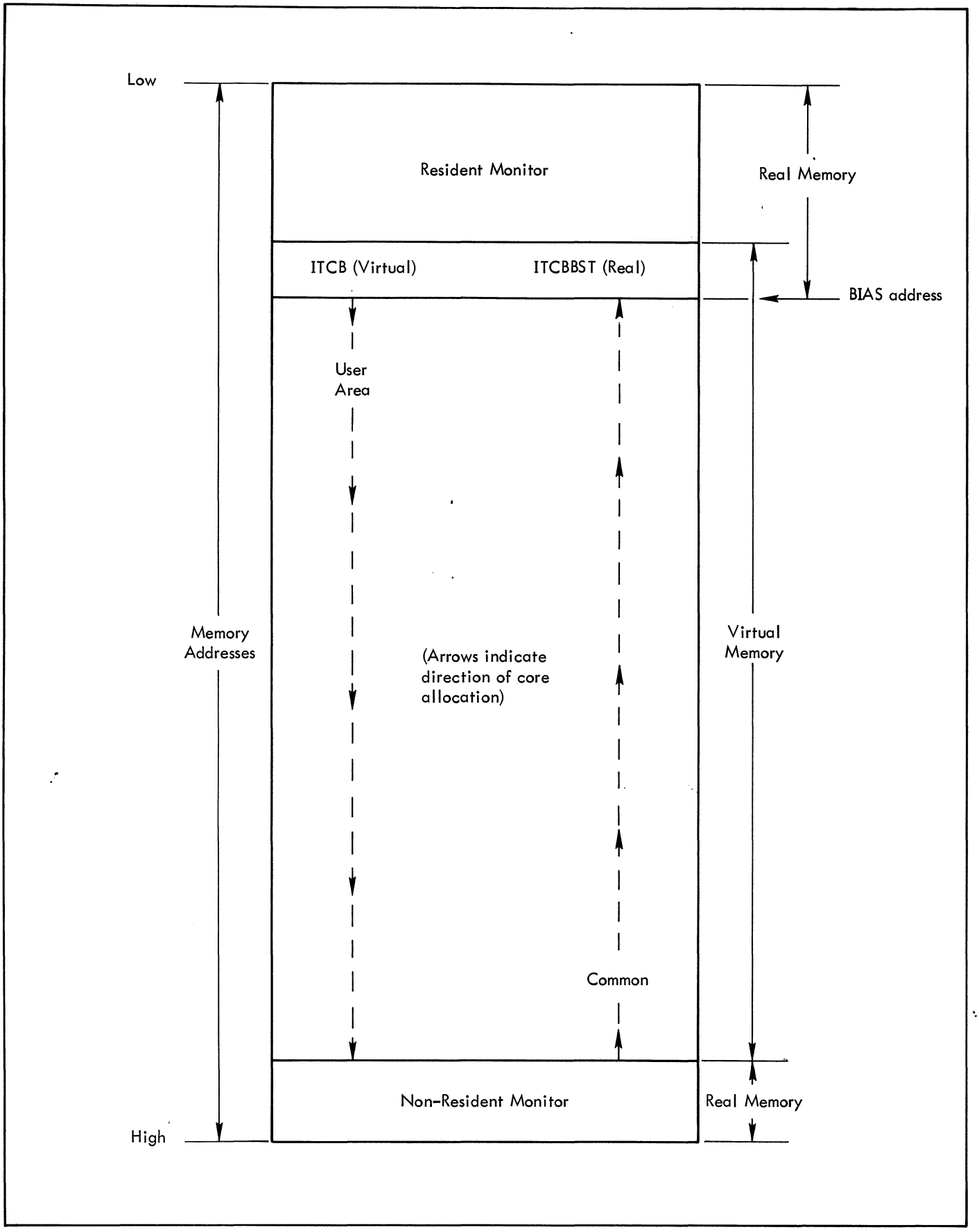


Figure 4-3. Core Memory Layout

5. They may have a COMMON area to allow communication between modules. However, the protection code of COMMON (and of the entire module) must be 0. A library module cannot generally contain labeled COMMON; blank COMMON must be used. If labeled COMMON is to be used, it must by itself constitute the entire module.

6. There must be only one segment per library module.

Each library module is a partition of a partitioned (library) file (see Chapter 6). Each time a library module is formed via a Link operation, the following occurs:

> The Link Editor uses the specified library module name as the principal partition key for that module. All external definitions contained therein are used as synonyms of the module name. All of these module identifiers are placed in the library (i.e., partition) directtory when a module is added to a library file.

If the library file status is NEW (not yet existent), the CREATE option must be used in addition to the LIB option to indicate that the library file is to be created. If the library file exists (status MOD), a search is made in the library's directory to find a library module with the same module name (partition key) as the one being entered. If a matching library module name exists, the existing library module name is deleted as well as all of the module name synonyms. The new library module is added to the library as a new partition, i.e., it does not physically replace the old module. Its name is added to the library as a new partition, i.e., it does not physically replace the old module. Its name is added to the library's directory as well as all of the module's external definitions. The space taken by the module being replaced remains used but inactive. The method for regaining the lost space is to periodically use the REORGP processor (see XDS Utilities Reference Manual). If a matching library module name is not found in the library dictionary, the new module is simply added to the library file as described above.

(CREATE) is effective only in conjunction with the LIB option and indicates that the library file identified by the LM operational label assignment is new. The absence of this option indicates that the library file identified by the !ASSIGN command specifying op-label LM is to be updated (must be STS, MOD), by either addition of new modules or replacement of old ones (see Example 1: Library Generation and Example 2: Library Updating).

Example 1: Library Generation

Given three object modules A, B, and C with the following external definitions:

Module A definitions are A1, A2

Module B definitions are B1, B5, B10

Module C definitions are CC, CX

Let us assume no library file exists and that one is going to be created using the above object modules as input. The illustration below shows the steps necessary to achieve the complete library of the modules.

Let us assume that object modules used in this example were created via assemblies of Meta-Symbol source modules into the following permanent sequential files on the user's account volume.

Module A = FILEA

Module B = FILEB

Module C = FILEC

Also assume that the name of the library file to be created is LIBFILE.

The following is representative of the control commands and options necessary to achieve the sample results:

```
!ASSIGN  LM,FIL,(NAM,LIBFILE),(STS,NEW)
!ASSIGN  GO,FIL,(STS,OLD),(NAM,FILEA)
!LINK
:OPTION  (LIB,A),(CREATE)
```

Step 1

```
!ASSIGN  LM,MTN,FIL,(STS,MOD),(NAM,LIBFILE)
!ASSIGN  GO,FIL,(STS,OLD),(NAM,FILEB)
!LINK
:OPTION  (LIB,B)
```

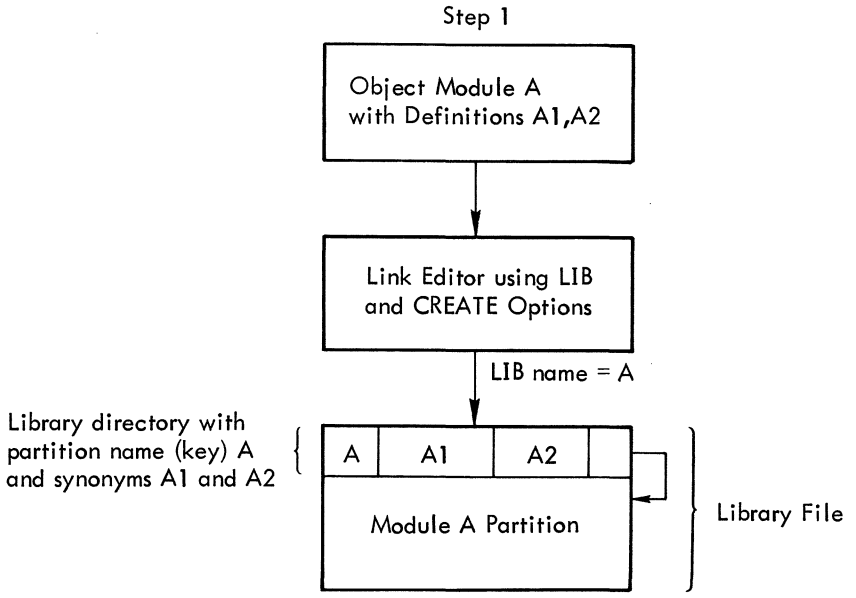
Step 2

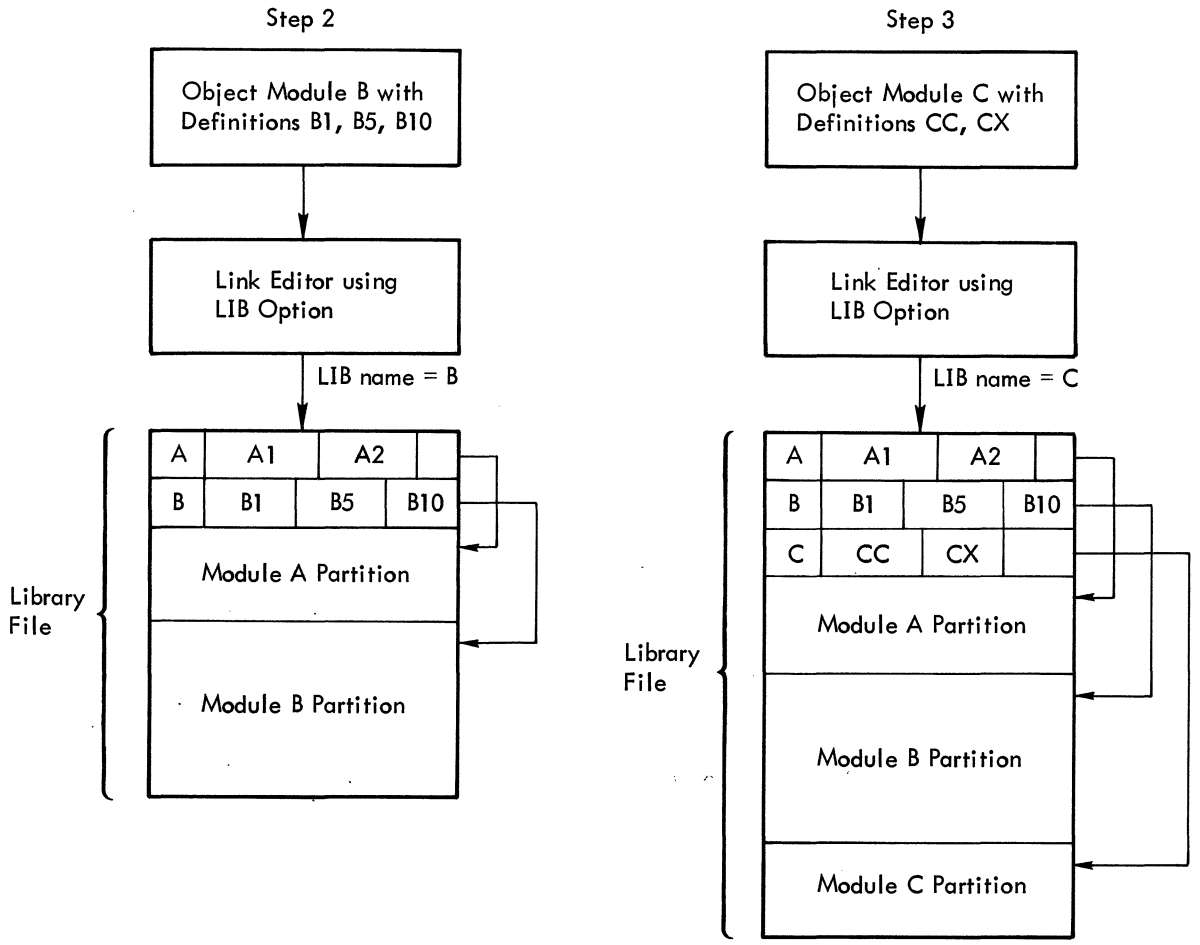
```
!ASSIGN  GO,FIL,(STS,OLD),(NAM,FILE)
!LINK
:OPTION  (LIB,C)
```

Step 3

Note that in Step 2, op-label LM is maintained (MTN option) for Step 3.

The above :OPTION commands do not use all applicable options and only serve to demonstrate the minimum requirements to build a library file.





Note: The above diagrams are not physically representative of the process of library building or of the actual library structure. Its intent is only to be schematically instructive.

Example 2: Library Updating

Given the library file created in the previous example, an update will be made to the library. One of the existing partitions will be replaced by a library module, B, formed from several object modules, in partitioned form, named B1, B2, and B3. The illustration below schematically illustrates the process.

The following commands illustrate the entire cycle of updating a library file (from the previous example) including assembly of source modules to a temporary partitioned GO file (created by Meta-Symbol).

```
!METASYM  SI,GO(B1,B2,B3)
      :
      :
  Source Modules for B1,B2, and B3
      :
      :
```

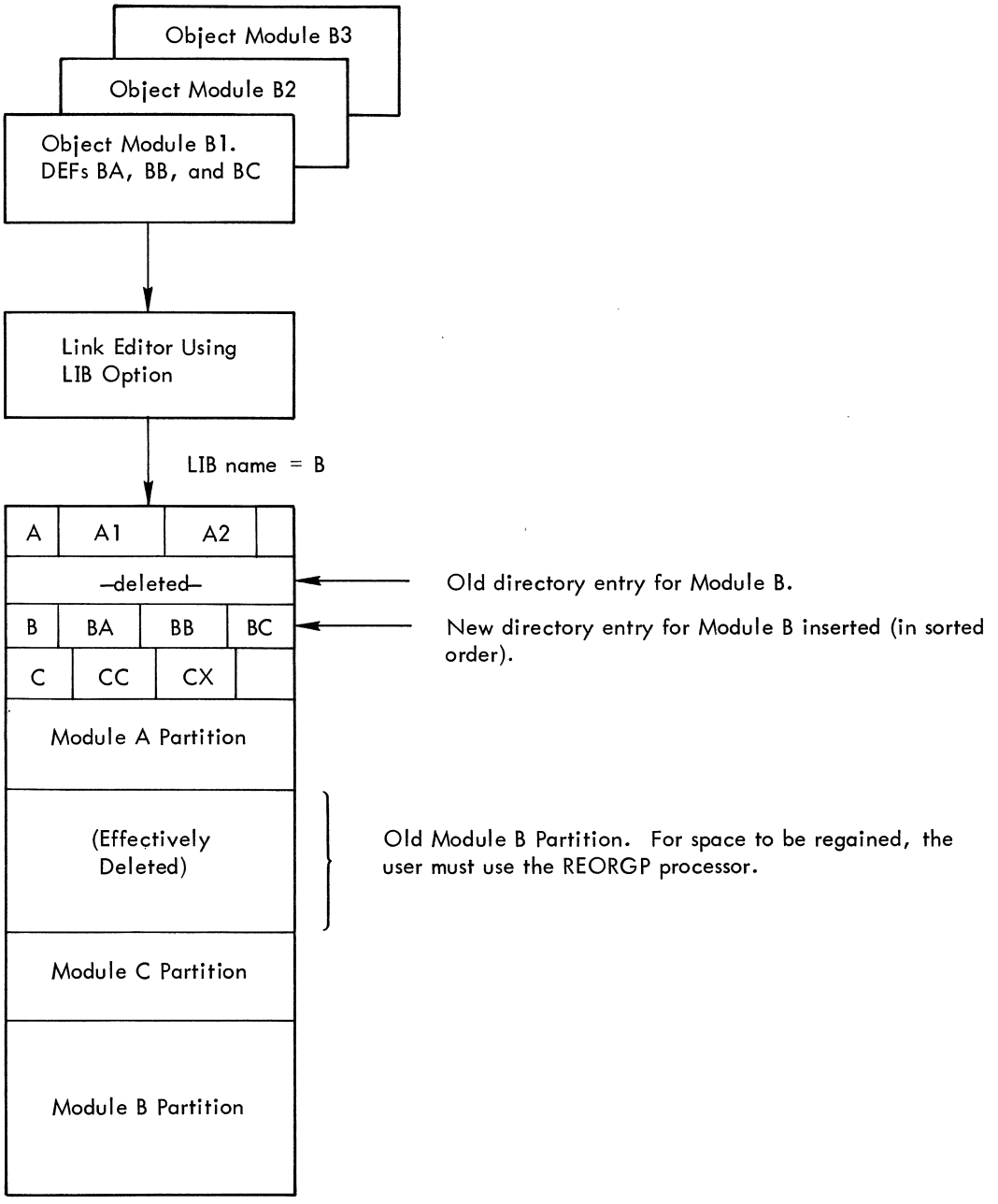
Step 1

```
    :
!ASSIGN    LM,FIL,(STS,MOD),(NAM,LIBFILE)
!LINK
:OPTION    (LIB,B)
:TREE      GO.B1-GO.B2-GO.B3
```

Step 2

Step 1 consists of the assembly (Meta-Symbol) of three source modules, each of the corresponding object modules to be placed in separate partitions on the temporary GO file. (GO is a predefined op-label, invoking an implicit assignment.)

Step 2 is the Link operation which forms the three partitions from the GO file into a single library module of three contiguous segments which, as a whole, will replace an existing library module named B as well as its associated external definitions (synonyms). The function of the :TREE command is described later in this chapter. Suffice it to say here that it serves to identify Link's input and structures the corresponding output.



(UNSAT,op-label ,op-label,...) where op-label is the operational label assigned to a library file to be searched during the Link operation to satisfy (primary) external references of object modules being linked. For each operational label used in the UNSAT option, there must be a corresponding !ASSIGN command that references a library file (op-labels are selected for search on left-to-right basis). A maximum of five libraries can be searched during one Link operation, including the system library if authorized (see SYSLIB/ NOSYSLIB option).

If the search of libraries specified in the UNSAT option does not exhaust (satisfy) the modules external references, the search is continued in the system library (by the name of SYSLIB, in account :SYS, read through op-label SYSL). The same search is made if the UNSAT option is not used.

Any primary references not satisfied by the library search will generate a severity level of 7 (see SL option and MAP), and although a module will still be generated, it will be impractical to execute.

({MAP / NOMAP}) Usage of the MAP option causes the Link Editor to produce a map of the loaded program. The load map provides the following information:

1. List of external definitions, external references, and program sections.

2. Hexadecimal addresses or values of external definitions and program sections.

3. Memory locations of segments and of areas with different protection types.

The default option is NOMAP.

The format of the map is described in a later subsection.

({SYSLIB / NOSYSLIB}) SYSLIB, which is the default, allows a search of the system library (SYSLIB) to satisfy any unsatisfied reference remaining after the search of user-specified libraries (see UNSAT option). The system library may be accessed by filename SYSLIB in the :SYS account, via the fixed operational label SYSL (in the corresponding !ASSIGN command).

The NOSYSLIB option prevents a search of the system library during Link's attempt to satisfy the external references of a module being link-edited. Any references remaining unsatisfied are errors of severity level 7 (see SL option).

(NOTCB) specifies that a Task Control Block (TCB) is not to be created by the Link Editor. The TCB table is normally created in the following order:

1. Program status doubleword at exit of the TRAP routine.

2. Status of the 16 registers.

3. Doubleword pointer of the user's temporary stack.

It is stored at the beginning of the first page of the program, or before the "0" protection of the user's root.

The minimum table is always generated for a load module because it is required at execution time. NOTCB is implied by the LIB option for a library module.

({ABS / REL}) ABS, the default, specifies that no relocation dictionary is to be created, and implies that the load point, or bias, of the load module is the system's default initial program load point (set by SYSGEN). The ABS option is ignored if LIB is also specified.

REL specifies that a relocation dictionary is to be included so that the load module is (virtually) relocatable (under special circumstances). This dictionary is required for a library module: the LIB option implies REL.

(SL,value) establishes the severity level of errors in object modules that the Link Editor will tolerate. If the severity of an error encountered in an input module is greater than the specified value, Link processing will be aborted. If the severity of errors encountered is less than the specified value, Link will attempt to continue processing. The specified value must be between X'0' and X'F', inclusive. The default value is X'7'.

Error severity levels are posted by compilers, assemblers, and other language processors as well as by Link.

At the end of object module generation, the compiler, assembler, etc., posts the severity level of any source errors. The Link Editor compares these values with that indicated by the user (or the default value). Table 4-3 shows the meaning associated with various points in the severity level range.

Link itself will detect (and indicate) possible or definite errors arising in the link editing process;

Table 4-3. Error Severity Levels

Severity Level	Meaning of the Code	Example
0	No abnormality or error.	No detected errors.
1	Abnormality.	
2	Possible error.	Minor assembly errors.
3		
4	Probable error.	Double definition.
5		
6		
7	Error detected: local effect.	Unsatisfied primary reference.
8		
9		
A		
B	Error detected: no local effect.	Invalid DO loop.
C		
D		
E	Fatal error.	
F		Syntax error on :OPTION card.

it will also compare these with the specified or default severity level. These errors are:

- An error in the :OPTION command, an unknown command, or a missing continuation card – severity level is F.
- An error in the :MODIFY command or the :REDEF command, or remaining unresolved primary references – the severity level is 7.
- Multiple external definitions – severity level is 4.
- An error in the :TREE command – the link edit operation is aborted.

(START, external symbol ± value) specifies the start address, i.e., the location of the first executable instruction. The optional value is in hexadecimal. The external symbol is an external definition that

- Indicates an address in the root of the program.
- Does not exceed 11 characters.

By default, the start address is determined by the compiler or assembler in a module of the root segment. If several object modules have a start address (e.g., several Meta-Symbol assemblies, each with the assembly instruction END address) the start address of the first object module of the root segment is used.

If the starting point must be in a library module to be included in the root segment, the START option must be used to reference an external definition in the module.

(TSS,value) specifies the number of words (hexadecimal) in the user's temporary storage stack. The default attribute is X'20'.

Note: Option TSS is not compatible with LIB.

COMMAND ORDERING

The :OPTION command, if used, must appear before any other Link command.

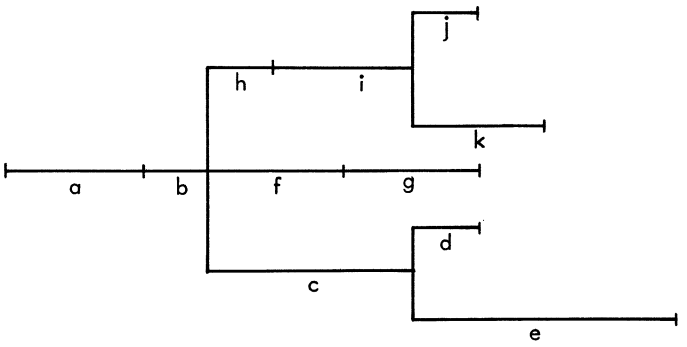
:TREE COMMAND (SEGMENTATION)

This command describes the structure of the program to be produced. A tree structure allows the user to minimize the program's core requirements. If there is no :TREE command, the GO file is the only one to be edited and the program will contain a root segment only.

Definitions: Several critical definitions follow.

- Segment: a named unit of code and/or data that can be separately loaded into memory at execution time, composed of one or more object modules.
- Root segment: the segment of a program that resides in memory throughout program execution.
- The name of the segment is obtained from the operational label of the object module (if only one) or of the first object module (if there are several that constitute the segment).
- A segment can also contain library modules.
- When the first module is a partition (whether an object or library module), the name of the segment is the partition name. Example: For GO.XYZ or L.LMED36, the name of the segment is XYZ or LMED36, respectively (for use in the M:SEGLOAD procedure and the :MODIFY and :INSERT commands).
- A path consists of those segments that may simultaneously occupy core storage.

Example:



where

a,b,...,k are object modules.

a+b,c,d,e,f+g,h+i,j,k constitute the segments named after the operational labels, A, C, D, E, F, H, J, K, respectively. A is the root segment.

The following terms are used to describe portions of a tree or overlay structure:

- Backward path of a segment: A is the backward path of C (see example above).
- Forward path of a segment: D and E are two forward paths of C.
- Overlay segment: E is the overlay segment of D.
- A+C+E, A+C+D, A+F, etc., are the paths of the tree.

Syntax

:TREE [,{REF[(value)] / SEG / FIL}] specification

where

REF indicates that the user relies on the system to load the segments that are not in core at the time of their execution. In this mode, any reference in a segment to an external definition located in another segment further from the root but on the same path is replaced by a branch to a subprogram. This subprogram generates an M:SEGLD that loads in memory the corresponding segment and all of its backward path as necessary. The value in hexadecimal, specifies the number of such references to be allowed for (default=X'10', i.e., 16 references).

SEG indicates that it is the user's responsibility to explicitly load each segment from disk storage to core storage (specified in advance by means of the M:SEGLD Meta-Symbol procedure). The tree structure thus produced is rigid because it is fixed at the time the program is written, contrary to the use of REF which makes structure changes easy.

The SEG and REF options are mutually exclusive and SEG is assumed by default.

FIL indicates that if the overlay tree specification is contained in a file, i.e., one generated by a language processor such as COBOL, :TREE,FIL and an ASSIGN command are used with the operational label TREE identifying the file describing the tree.

specification specifies the overlay tree structure, as described below. The specification field of the :TREE command is separated from the foregoing by one or more blanks.

TREE STRUCTURE SPECIFICATION

The delimiters used have the following meanings:

- - The operational labels separated by this delimiter correspond to segments loaded in contiguous virtual memory.
- , The two segments separated by this delimiter are to overlay one another in memory; they begin at the same virtual memory location.
- ( ) Parentheses indicate the limits of an overlay level.
- . The symbol that follows this delimiter identifies a partition of a file named by the operational label that precedes the symbol.

Usage Rules

- Two overlay segments cannot communicate with each other.
- When two overlay segments reference the same library module, this module is incorporated in both segments.
- When two or more segments located on the same path reference the same library module, this module is incorporated in the segment furthest from the root.
- The use of COMMON is possible with library modules. This technique allows communicating the same data to several modules. The storage protection code of a COMMON module must be 0.

Contrary to blank COMMON, labeled COMMON presents the following problems:

1. If they are initialized and are of different length, they are handled correctly by the Link Editor. But if the initialization concerns areas of the same length, the values of the last module edited prevails.
2. A library module cannot contain labeled COMMON as a subdivision; blank COMMON must be used.
3. Labeled COMMON can be stored in a library only if it constitutes by itself a library module.

Operational Label Assignments: The usage of several differing operational labels on the :TREE card is restricted, by file characteristics, as follows:

GO Must be used for temporary partitioned object module files; may be used for permanent partitioned object module files.

GEN Must be used for permanent partitioned object module files unless GO is used.

xxxx Any valid operational label (including GO) except LM or GEN may be used for sequential object module files, either temporary or permanent.

yyyy Any valid operational label except LM may be used for library module files.

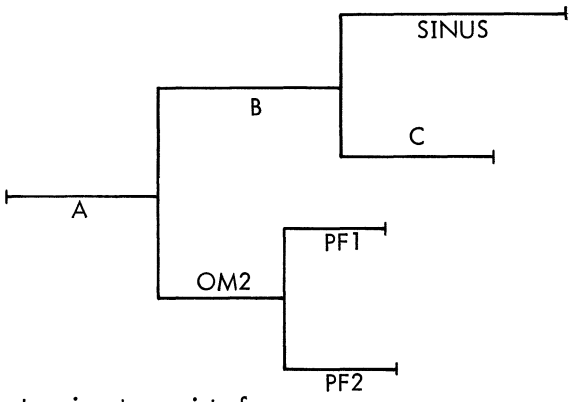
Example 1:

The overlay tree described in the foregoing is represented by:

```
:TREE A-(C-(D,E),F,H-(J,K))
```

Example 2:

The following tree must be link edited:



The tree inputs consist of

- Three permanent sequential object modules a, b, and c, with operational labels A, B, and C.
- Three object module partitions called OM1, OM2, OM3 in the GO temporary partitioned file.
- Two object module partitions called PF1 and PF2 in a permanent partitioned file referenced by operational label GEN.
- The library subprogram SINUS contained in a library file reference by operational label BIB.

The required :TREE command is

```
:TREE  A-(GO.OM2-(GEN.PF1,GEN.PF2);
:        ,B-(C,BIB.SINUS))
```

The names of the segments are as indicated:

A,OM2,PF1,PF2,B,C,SINUS.

The object modules on the GO file named OM1 and OM3 were not declared and are not edited.

Library subprograms other than SINUS that might be called to implicitly satisfy external references(UNSAT option) are incorporated in the corresponding segment.

COMMAND ORDERING

The :TREE command, if used, must follow the :OPTION command, if any, and precede any other Link command.

:MODIFY COMMAND

The :MODIFY command allows the user to modify Link input during its editing, i.e., without reassembling. These modifications remain in the load module indefinitely. The command MODIFY and the LIB option of the :OPTION command are mutually exclusive. :MODIFY is only valid for load modules; and the operational label LM must, in addition, be assigned to a disk file.

Syntax

:MODIFY[,segment-name] location,value[,value,...]

where

segment-name indicates the segment to be modified, according to the definitions given under :TREE. It must not appear if the program consists of only a root segment.

location specifies the location of the first word to be modified; generally expressed as an external definition plus or minus an optional hexadecimal offset value. It can also be expressed as a plus sign followed by a hexadecimal value directly designating the virtual address at execution time of the first word to be modified.

value specifies the value(s) to be placed in the indicated location(s). The values are unsigned hexadecimal values, optionally followed by a plus or minus sign and an external definition. For an appropriate address resolution, the address resolution function (BA, HA, WA, or DA) must precede the external definition name enclosed in parentheses. By default, word resolution is assumed.

If more than one value follows the location specification, ascending sequential locations are modified accordingly.

Example 1:

:MODIFY LOC+A1,0001234E+BA(ALPHA)

The value X'1234E' is added to the value of the external definition ALPHA, treated as a byte address, and the result is stored at the address LOC+A1, e.g., LOC+161_{10}.

Example 2:

:MODIFY,SEGA +328A,32380000+DEF,68300004+BETA

The two words of segment SEGA located at virtual addresses X'328A' and X'328B' are replaced by the specified values at execution time.

COMMAND ORDERING

:MODIFY commands, if used, must follow the :OPTION and :TREE commands, if any, may be intermixed with :INSERT commands in any order and must precede the :REDEF command, if any.

:INSERT COMMAND

The :INSERT command allows the user to insert new information into the Link input during link editing. The insertions are effective indefinitely. The insertion process produces an out-of-line 'patch area' that becomes a permanent part of the load module.

Syntax

:INSERT[,segment-name] location,value[,value,...]

where

segment-name indicates the segment to be modified, according to the definitions given under :TREE. It must not appear if the program consists of only a root segment.

location specifies the insertion point: the insertion(s) will effectively precede the instruction word specified. The location is generally specified as an external definition plus or minus an optional hexadecimal offset value. It can also be expressed as a plus sign followed by a hexadecimal value directly designating the virtual address of the insertion point.

value specifies the value(s) to be placed in the indicated location(s). The values are unsigned hexadecimal values, optionally followed by a plus or minus sign and an external definition. For an appropriate address resolution, the address resolution function (BA, HA, WA, or DA) must precede the external definition name enclosed in parentheses. By default, word resolution is assumed.

If more than one value follows the location specification, effectively ascending sequential insertions are made accordingly. In any case, the instruction currently occupying the specified location is replaced by a branch instruction that transfers control to an out-of-line patch area created by Link. This patch area will contain the specified insertion(s) followed by the replaced instruction, in turn followed by another branch to the specified location+1.

Example:

Given the :INSERT command

```
:INSERT   +3483,69303484,32100006
```

and the following program code (in symbolic form) before insertion

```
          :
          :
3482      LW,1   7
3483      LI,D   0
          :
          :
```

the same program area following insertion would appear as

```
          :
          :
3482      LW,1   7
3483      B      PATCH
          :
          :
```

Note that the instruction that occupied the specified (virtual) location, 3483, is replaced by a branch to an area where the insertions are located, e.g., PATCH. This patch area would appear as follows:

```
PATCH     BCS,3    3483
          LW,1     6
          LI,D     0
          B        3484
```

The instruction that originally occupied location 3483 (and was replaced by the branch instruction) now follows the inserted instructions. The patch area ends with a branch back to the next sequential location in the program (3484).

COMMAND ORDERING

:INSERT commands, if used, must follow the :OPTION and :TREE commands, if any, may be intermixed with :MODIFY commands in any order and must precede the :REDEF command, if any.

:REDEF COMMAND

The :REDEF command allows the user to replace external definition or reference symbols appearing in either object or library modules with new symbols during link editing, while forming either a load or library module. If the :REDEF command is used, :MODIFY and/or :INSERT commands and the START option must use the new symbol when referring to an altered definition.

The redefinition capability is particularly important in the following cases:

1. For resolving the problem of two different library subprograms with identical external definitions (definition symbols must be unique within a library file).

2. For replacement of external-definition symbols of more than 11 characters (unacceptable in library modules) with new, shorter symbols.

Syntax

```
:REDEF   (oldsymbol,newsymbol)[,(oldsymbol,;
:                newsymbol),...]
```

where

oldsymbol is the external definition or reference symbol to be replaced.

newsymbol is the replacement symbol.

Example:

```
:REDEF   (COS,COSINE),(LONGNAMEOFDEF,;
:                SHORTNAME)
```

All external definitions or references occurring in any of the modules being edited that correspond to COS or LONGNAMEOFDEF are replaced by COSINE or SHORTNAME, respectively.

LOAD MAP FORMAT

The load map produced by Link when the MAP option is specified (see MAP/NOMAP option under :OPTION

command, above) consists of multiple entries of the following form:

SEGMENT NAME: sss...s

aaaa xxxxxxxx y bbb...bp

where

sss...s is the current segment name or ROOT (in the case of no :TREE command)

aaaa is a code indicating the type of element, as follows:

DEF	external definition
UDEF	unused definition
LDEF	library satisfied definition
DDEF	doubly defined external definition
PREF	unsatisfied primary reference
SREF	nonsatisfied secondary reference
CSEC	control section
DSEC	dummy section

Note that PREF implies that a search was made and that this reference could not be satisfied. SREF, by definition, implies that there was no search, and therefore that the corresponding definition may or may not exist.

xxxxxxxx y is the hexadecimal address of the element, where y represents a byte displacement (y = 0, 1, 2, or 3).

If y is not present, xxxxxxxx represents the value rather than the address of bbb...b.

bbb...b is the symbolic name of the element.

If bbb...b = SEGLOp or = SEGHIp

then bbb...b represents the lower and upper boundaries of segments of code with different protection codes. SEGLOp is the first word of the segment lower limit and SEGHIp is the last word of the segment upper limit. The protection type, indicated by p, may be 0, 1, or 2. (Protection areas always begin on page boundaries.)

LINK EDITING EXAMPLES

Example 1:

```
!JOB  TRAVAIL,1234,JONES
!METASYM  SI,LO,GO
      :
      :
   symbolic cards
      :
      :
!LINK
!RUN
```

Example 1 illustrates the implicit assignments accepted by XOS. The predefined operational labels SI, LO, and GO, used for the first job step, imply the card reader input symbiont file, IN, the printer output symbiont file, OUT, and a temporary file (normally disk), op-label GO, created by Meta-Symbol. Note that no !ASSIGN commands are needed if the implied assignments are desired.

The Link Editor, by default (i.e., no options), edits the GO file and creates a temporary output file with the operational label LM. Finally, with no load module file name having been specified in the !RUN command, the module found under the label LM is loaded and executed.

Example 2:

```
!JOB  EXERCISE1,119,KELLY,2
!METASYM  SI,LO,GO(PART1,PART2,PART3)
      :
      :
   3 symbolic decks
      :
      :
!ASSIGN  LM,MTN,FIL,(STS,NEW),(NAM,ARBRE)
!LINK
:TREE  GO.PART1-(GO.PART2,GO.PART3)
!RUN  (LMN,ARBRE)
```

Example 2 shows three successive assemblies in the course of one job step. Note that it is not necessary to use three separate !METASYM cards if the names of the several object modules produced by the assembly are specified as partitions of a partitioned file, e.g., ...GO(PART1,PART2,PART3). The Link output is specified as a simple tree structure to be created on the permanent file named ARBRE, since the implicit temporary file assignment for LM is overridden by an explicit assignment.

Example 3:

```
!JOB  PROBLEM,115,VALDEZ,2
!METASYM  SI,LO,GO
     .
     .
     .
    symbolic deck
     .
     .
     .
!ASSIGN  OM1,FIL,(STS,OLD),(NAM,AB),(UNT,AC,116)
!ASSIGN  OM2,FIL,(STS,OLD),(NAM,ROOT),;
!        (UNT,AC,121)
!ASSIGN  BIBL,FIL,(STS,OLD),(NAM,PRIVATE)
!ASSIGN  LM,MTN,FIL,(NAM,PROGRAM)
!LINK
:OPTION  (SLS,7)
:TREE  OM2-(OM1-GO,BIBL.TRI)
:REDEF  (ANTICONSTITUTIONAL,IRREGULAR)
!RUN  (LMN,PROGRAM)
```

The result of the link editing in Example 3 is a tree-structured load module stored on a permanent file, PROGRAM, under account 115.

The step following the link editing is the running of PROGRAM, consisting of a root and two overlay segments. The library routine TRI forms part of one of the two segments.

The modules for editing are:

- ROOT cataloged under account 121 with a password.
- AB a permanent file in account 116.
- The temporary file assigned to GO in account 115, which was built during the Meta-Symbol job step preceding the Link step.
- TRI a library module from the user's library PRIVATE (account 115).

Example 4:

```
!JOB  RESULT,119,GOLDBERG,2
!EXEC  COMPILGO
!DATA
     .
     .
     .
    symbolic deck
     .
     .
     .
!EOD
:OPTION  (UNSAT,CLIB)
:TREE,FIL
!EOD
     .
     .
     .
    data cards
     .
     .
     .
!EOD
```

The job shown in Example 4 involves three steps:

- A COBOL compilation resulting in three object modules. The COBOL compiler segments the generated program according to the user's request (SEG option). The overlay tree specification is contained in a file generated by COBOL (indicated by the FIL option of :TREE), which for example might contain:

  :TREE GO.00-(GO.01,GO.02)

- Link editing: Linking the object modules and satisfying external references from a COBOL library named COBLIB.
- Execution (the data will be read dynamically onto a symbiont file).

The cataloged command set COMPILGO contains the following control commands:

```
!COBOL  LS,LO,XREF,DEBUG,GO,SI,SEG
!ASSIGN CLIB,FIL,(NAM,COBLIB),(STS,OLD);
!       (UNT,AC,:SYS)
!LINK
!RUN
```

Example 5.

```
!JOB  EXAMPLE,32,WONG,F
!LIMIT  (TIME,20),(PAGE,1000)
!FORTRAN  SI,LO,GO
     .
     .
     .
    symbolic deck
     .
     .
     .
!ASSIGN  EI,FIL,(STS,OLD),(NAM,BINARY1),;
!        (UNT,MT,(VOL,1234))
!ASSIGN  EO,MTN,FIL
!FMGE  COPY
!ASSIGN  ABC,FIL,(STS,OLD),(NAM,BINARY5),
!ASSIGN  LIBR,FIL,(STS,OLD),(NAM,MYLIBRARY)
!LINK
:OPTION  (UNSAT,LIBR),(SL,7)
:TREE  GO-ABC-EO
!RUN
```

The job in Example 5 consists of compiling a FORTRAN source deck and linking the object module produced (on a temporary file) to the other already existing object modules—one residing on magnetic tape and the other on permanent disk storage. A !FMGE COPY step is employed to copy the tape file BINARY1 to a temporary disk file (op-label EO).

External references are to be satisfied by the library file MYLIBRARY (op-label LIBR).

The load module produced on a temporary file (implicitly labeled LM) created by the Link Editor is executed.

DIAGNOSTIC MESSAGES

The diagnostic messages produced by Link on the listing log are listed below.

ABNORMAL FILE (RECORD=0)

ABNORMAL I/O

BIAS TOO LARGE

CONTROL CARD MISSING. BINARY AFTER OPTIONS PROCESSED

CONTROL CARD MISSING. UNEXPECTED BINARY IGNORED

END OF BI BEFORE LAST CARD OF ABOVE MODULE

ERROR: ABS AND LIB

ERRORED TREE

ILLEGAL ARG

ILLEGAL CARD IN ABOVE BI MODULE

ILLEGAL DSECT * NAME:

ILLEGAL LMED IDENTIF.
ILLEGAL LMED * COMMON
ILLEGAL LMED * OVERLAY
ILLEGAL LMED * PROTECTION
ILLEGAL LMED * RELOCATION
(LMED = library module)

ILLEGAL OBJECT LANGUAGE

ILLEGAL OM IDENTIFIED

ILLEGAL SEQUENCE OF CARDS

ILLEGAL SIZE OF FIELD
(One of the parameters has too many characters.)

ILLEGAL SYNTAX:
(followed by the portion of the command between the end of the last correct option and the current character at the time the error was detected – for :OPTION, :MODIFY, and :REDEF commands only.)

ILLEGAL SYNTAX AFTER:
(followed by the name of the last module processed – for :TREE command only.)

ILLEGAL SYNTAX * REF(20) SUPPLIED

ILLEGAL VALUE

ILLEGAL VALUE IN:
(followed by the name of the option – for :OPTION command values of BIAS, SL, or TSS.)

IMPROPER BOUND

LIB OPTION ABS IGNORED

LIB OPTION * MODIFY IGNORED

NO MODIFY WITH LM ASSIGNED TAPE

NO TYPE 3 PROTECTION

OVERFLOW PASS2

REPEAT ERROR. END OF CARDS SUPPLIED

REPEAT. UNK. CARDS

SEV. LEV.

SEV. LEV. EXCEEDED IN OM

STACK OVERFLOW

TOO MANY REFS

TOO MANY UNSATS

:TREE CARD ERRORED

UNEXPECTED END

UNEXPECTED EOD

UNEXPECTED OM END

UNKNOWN CARD

UNKNOWN LOAD IN ABOVE CARD

UNKNOWN OPTION

W CANNOT VALUE START ADDRESS

W MODIFY: IGNORED REL

5. DEBUG AIDS

INTRODUCTION

XOS furnishes the user with a collection of debugging aids grouped into a system service called Debug. Its principal components are

- A set of system procedures for use at assembly time to incorporate specific calls for Debug services in the user's program.
- A system processor, also called Debug (invoked by a !DEBUG control command), for use at run time to incorporate specific calls for debug services in the user's program.
- A set of modules within the monitor that services calls generated by the aforementioned components

Two methods, not mutually exclusive, are available for the user to request debug services.

1. With direct modification of his source program: the user may include, within his program source deck, references to the various XOS system procedures that automatically generate calls for the desired debug services. These procedures should be treated as executable but program-transparent statements to be located in the user program at the exact places where their execution is desired.

2. Without direct modification of his source program: the user may invoke the Debug processor, using a !DEBUG command rather than the !RUN command, to execute his program. The Debug processor reads a set of Debug processor commands, which define the specific debug services desired. It also reads the user program as an executable load module (LMEX) using the op-label LM. The Debug processor then modifies the user program to include calls for the debug aids specified by the Debug processor commands and copies the modified program onto a temporary file. It then transfers control to the modified user program using the system service M:LDTRC (see Chapter 8).

DEBUG PROCESSOR USAGE

Debug processor command usage follows the same rules as for system control commands. The !RUN command of a normal execution is replaced by

- Assigning the LM op-label to the load module corresponding to the user program.
- A call to the Debug processor.
- Debug processor commands that specify calls for debug services (e.g., :SNAP,:SNAPC,:PMD,:PMDI) or modifications for the user code (:MODIFY,:INSERT).

The Debug processor reads the Debug processor commands as data in the job input file, assigned to op-label SI. These commands must follow the !DEBUG command and must be terminated by a !EOD command if the user program reads subsequent data from the same file (SI). If the user program is normally initiated by a !processor-call command with options, these options should be included on the !DEBUG command, in the same form as for the !processor call command.

The Debug processor call influences the job resource requirement. It creates a temporary file that may require the user to increase the SPDISC value in his !LIMIT command.

User programs executed via !DEBUG occupy more memory space than those executed via !RUN. The space difference is about one-half to a full page.

During the job step involving invocation of the Debug processor, op-label LM corresponds to the initial user program, op-label *LMN corresponds to the Debug processor, and op-label **LM identifies the temporary modified version of the user program. Diagnostics output by the File Management System are in accordance with these op-label conventions.

Generally, all values used in Debug processor commands are in hexadecimal. The only exceptions are the :COUNT command and certain condition-specification (SWITCH option) parameters.

SPECIAL DEBUG SYNTAX

The syntax of a Debug processor command is identical to that of a system control command (see Chapter 3) except as follows:

1. The first character is a colon (:). If Debug commands are used, the Debug processor must first be invoked with a !DEBUG command. The Debug processor then reads the Debug processor commands through the op-label SI.

2. If an overlay segment is to be modified, the name of a command can be followed immediately by a comma and a segment name.

The syntax of the Debug procedures is identical to that of the other system procedures (see Chapter 8) or input/output procedures except for an extended syntax used to specify a DCB address in the M:SNAP and M:SNAPC procedures.

Throughout this chapter, the restrictions on explicit value syntax (e.g., hexadecimal address, decimal :COUNT and SWITCH specifications) apply only to the Debug processor commands. In the Meta-Symbol procedures, an integer is always an integer, regardless of its type (hexadecimal, decimal, octal, etc.). To simplify the syntax notations, the address syntax is not detailed each time. Generally, the address form is

+hexadecimal constant

or

external[±relative hexadecimal constant]

where

hexadecimal constant specifies an absolute address. This address format necessitates foreknowledge of loading. This is possible with the map furnished by the Link Editor.

external is a symbol declared external at assembly time by the assembler directive DEF.

relative hexadecimal constant gives a displacement with respect to an externally defined symbol.

DEBUG COMMANDS AND PROCEDURES

Warning: In the Debug commands that specify an initialization address, i.e., where the command is to be executed, the specified location must contain an executable instruction which is neither altered nor replaced during program execution. The user is specifically cautioned against specifying the address of an instruction that refers to the current location counter such as a BAL (Branch and Link) instruction. This is necessary because, at memory loading, the contents of this address are replaced by a branch to the debug service. The replaced instruction is executed after the debug service is performed, but at a different memory location. This applies to all Debug processor commands except :DCB, :PMD, and :PMDI.

:DCB COMMAND

This command allows the user to specify a user DCB to be used for the output of requested debug information.

Syntax

:DCB address

where

address is the address of the DCB to be used for output generated by other DEBUG commands.

This command is optional but, if present, it must immediately follow the !DEBUG command. If omitted, the analysis information is printed automatically on the listing log (OUT file) associated with the job. The DCB must be opened by the user program before any possible execution of the debug processor commands. Furthermore, it must be specified for use with the ASAM access method and no record movement (LOC).

There is no corresponding system procedure. However, the Debug procedures M:SNAP and M:SNAPC permit an optional DCB address parameter.

:PMD AND :PMDI COMMANDS

These postmortem dump commands allow the user to specify that portions of his program area be dumped (in hex) at the end of its execution.

The :PMD command is conditional in its operation. It causes the output of the specified dumps only when the user's program is aborted or terminated with the execution of an M:ERR system procedure.

:PMDI is an unconditional command. It causes the output of the specified dumps regardless of the program's condition upon termination (i.e., normal, abort, or exit with M:ERR).

Syntax

$$:\begin{Bmatrix}\text{PMD}\\ \text{PMDI}\end{Bmatrix}[,\text{seg}]\ \begin{Bmatrix}[(\text{start address, end address})],\ldots\\ [(\text{pp})]\end{Bmatrix}$$

where

seg specifies the name of an overlay segment (valid only if program has tree structure).

start address and end address give the beginning and ending word address of an area to be printed. A single command may include multiple area specifications. If no area is specified on the command, the complete memory area occupied by the program is printed.

pp is a number which can be 00 (all access), 01 (no write), or 10 (read only). It indicates that the area corresponding to that protection type is to be printed.

Multiple :PMD, :PMDI commands, and redundant or overlapping area specifications are honored except for the default (no area specified) case. Area and pp specifications may not both appear in the same command.

There are no corresponding PMD or PMDI procedures.

:SNAP COMMAND AND M:SNAP PROCEDURE

The :SNAP command causes the printing (dumping) of one or more memory areas just before the execution of the instruction at the address indicated on the :SNAP command.

The M:SNAP procedure reference functions like an executable statement whose execution results in printing (dumping) one or more memory area.

Command Syntax

```
:SNAP[,seg] address,comment[,(start-address,;
:    end-address),...]
```

Procedure Syntax

```
[labels] M:SNAP[,dcb-adr] 'comment'
        [,(start-address,end-address),...]
```

where

seg specifies the name of an overlay segment. (Valid only if the program has a tree structure.)

address specifies the initialization address. The Debug service is performed just before the execution of the instruction at this address. The initialization address must be in the designated segment. The branch (of the tree) in memory when the snap occurs must include whatever externals are used to define the area(s) to be printed. The area printed can be outside of the segment.

comment is a mandatory string of one to eight alphanumeric characters which is to be printed just before the image of the specified memory area(s). When using M:SNAP, this must be entered as a character string constant (i.e., with ' ' qualifiers).

start and end address give the beginning and ending addresses of an area to be snapped.

dcb-adr specifies the address of the DCB through which the information is to be output. If specified, the user must open the DCB prior to the first M:SNAP or M:SNAPC using it. If the DCB address option is not used, the data output automatically occurs on the OUT device (listing log).

When a link edit is performed with the REF option in the :TREE command, use of the M:SNAP procedure to print an area in another segment can cause unpredictable results. See Chapter 4, Link Editing.

:SNAPC COMMAND AND M:SNAPC PROCEDURE

The :SNAPC command and the M:SNAPC procedure have the same action as the :SNAP command and the M:SNAP procedure except that their operation is conditional.

Command Syntax

```
:SNAPC[,seg] flag,address,comment;
:    [,(start-address,end-address),...]
```

Procedure Syntax

```
[labels] M:SNAPC[,dcb-adr] flag,'comment'
        [,(start-address,end-address),...]
```

where flag is the name of a switch that is tested for set or reset (on or off) status.

Except for flag, all parameters have the same meaning as for the :SNAP command and the M:SNAP procedure.

For the Debug processor commands, flag is a string of one to eight alphanumeric characters. The monitor does not associate this flag with symbols in the user program, hence there is no possible confusion with those symbols. It creates its own switch tables (transparent to the user) in which the switches are initialized to the set state. For M:SNAPC, flag is the address of a data word in the user program.

The request is honored only if the specified flag is set. A flag can be set or reset by using other commands or system procedures of the Debug system.

:IF COMMAND AND M:IF PROCEDURE

The :IF command and the M:IF procedure request the Debug service to test a specified condition and to set or reset the associated flag according to whether the tested condition is true or false.

Command Syntax

```
:IF[,seg] flag,address,(condition)
```

Procedure Syntax

[labels] M:IF flag,(condition)

where

seg specifies the overlay segment, if any.

flag is the name of the switch that is to be set or reset (for subsequent testing by :SNAPC or M:SNAPC).

address specifies the initialization address. The test is made just before the execution of the instruction found at this address.

condition is either an arithmetic relation to be tested or a specification of bits in the job switch word (JSW) to be tested. The syntax is

$$\left\{ \begin{array}{l} [*]\ i_1,x_1\ [,b_1]\ ,r,[*]\ i_2,x_2[,b_2] \\ \\ \text{SWITCH},v_1\ [,v_2,\ldots,v_n] \end{array} \right\}$$

where

i_1 and i_2 specify the addresses of the fields to be compared. The addresses may be indirect, and either absolute or relative and may have addends.

x_1 and x_2 specifies the user's index registers to be used to modify the i_1 and i_2 addresses (hexadecimal) respectively. If there is to be no indexing, zero must be specified for either or both registers as appropriate. The legal range of values of x_1 and x_2 for effective indexing is 1 through 7 only.

b_1 and b_2 indicate the number of bytes in the fields to be compared. The values of b_1 and b_2 can be different (e.g., a 2-byte field can be arithmetically compared with an 8-byte field). If one or the other of the values is not specified, 4 is taken by default. The acceptable values are

1 byte

2 halfword

4 word

8 doubleword

r specifies the type of comparison. The acceptable relations are

GT greater than

LT less than

EQ equal to

GE greater than or equal to

LE less than or equal to

NE not equal to

v is a decimal integer where $2 \leq v \leq 31$. If at least one of the corresponding binary bits in the Job Switch Word is set, the condition is true.

:AND COMMAND AND M:AND PROCEDURE

The :AND command and the M:AND procedure request the Debug service to test a specified condition if the specified flag is already set. If the test condition is true, the flag remains set; otherwise, the flag is reset.

Note: In an AND, if one of the conditions is known initially to be false (e.g., flag initially reset), the evaluation of the second condition is superfluous; the result of the AND is known (to be false).

Command Syntax

:AND [,seg] flag,address,(condition)

Procedure Syntax

[labels] M:AND flag,(condition)

The parameters have the same meaning as defined for the :IF command and the M:IF procedure.

:OR COMMAND AND M:OR PROCEDURE

The :OR command and the M:OR procedure request the Debug service to test a specified condition if the corresponding flag is reset (off). If the test condition is true, the flag is set; otherwise, the flag remains reset. If the flag is initially set, it remains set.

Note: In an inclusive OR, if one of the conditions is known initially to be true (e.g., flag initially set), the evaluation of the second condition is superfluous; the result of the OR is known (to be true).

Command Syntax

:OR[,seg] flag,address,(condition)

Procedure Syntax

[labels] M:OR flag,(condition)

The parameters have the same meaning as for the :IF command and the M:IF procedure.

:COUNT COMMAND AND M:COUNT PROCEDURE

The :COUNT command and the M:COUNT procedure allow the setting or resetting of a flag depending upon the number of times the specified command or procedure has been executed.

Command Syntax

:COUNT[,seg] flag,address,start,end,step

Procedure Syntax

[labels] M:COUNT flag,start,end,step

where

seg and flag have the same meaning as in the :IF command and the M:IF procedure.

address designates the initialization address. The test is made just before the execution of the instruction found at this address.

start specifies the smallest decimal value of the internal counter for which the flag is to be set.

end specifies the greatest decimal value of the internal counter for which the flag can be set.

step specifies the decimal count intervals (within the inclusive range designated by start and end) at which the flag is to be set.

The Debug service constructs an internal counter initialized at zero and increments it by one each time the command is executed. The flag is set if the value of the counter less the starting value is a multiple of the specified step and also is within the specified range; otherwise, the flag is reset. Therefore, the flag is set if the count is between start and end, inclusively, and the quotient count minus start divided by step is an integer.

The values of step and start must be smaller than the value of end. For :COUNT, the internal counter is incremented just before the execution of the instruction found at the designated address.

:MODIFY COMMAND

Beginning at the designated address, the :MODIFY command specifies the replacement of one or more consecutive memory words in any area of a program (instructions or data).

Syntax

:MODIFY[,seg] address,value[,value,...]

where

seg designates the overlay segment name (valid only when the program has a tree structure). It is optional when the segment is the root of the program. The modified words must be within the specified segment.

address designates the word address of the first word to be modified.

value is one of the following three forms:

hexadecimal constant

hexadecimal constant ± external

hexadecimal constant ± resolution (external)

where

hexadecimal constant is a hexadecimal number varying from one to eight digits. It defines the value to be inserted in the specified word. It is right-justified and filled with zeros when less than eight digits.

external represents an externally defined program symbol.

resolution may be used to specify the resolution to be used if an external value is an address. The external address value is added to the value word with the indicated resolution. The keywords for address resolution specification are

BA	Byte address
HA	Halfword address
WA	Word address
DA	Doubleword address

:INSERT COMMAND

The :INSERT command specifies the logical insertion of one or many consecutive memory words at a specific word address in any area of a program (normally used with instructions only).

Syntax

:INSERT[,seg] address,value[,value,...]

where

seg designates the overlay segment name (valid only when the program has a tree structure). It is optional when the segment is the root of the program. The insertion address must be in the specified segment.

address specifies the word address where the insertion is to be made. The instruction originally in that address is executed following the execution of the insertion; it is not executed in its initial location.

value is one of the following three forms:

hexadecimal constant

hexadecimal constant ± external

hexadecimal constant ± resolution (external)

where

hexadecimal constant is a hexadecimal number varying from one to eight digits. It defines a value to be inserted at the specified location. It is right-justified and filled with zeros when less than eight digits.

external represents an externally defined program symbol.

resolution may be used to specify the resolution to be used if an external value is an address. The external address value is added to the value word with the indicated resolution. The keywords for address resolution specification are:

BA	Byte address
HA	Halfword address
WA	Word address
DA	Doubleword address

The effect of INSERT is to

- Place the specified insert values (instructions) in contiguous locations in the user's common dynamic area.
- Replace the value (instruction) at the specified location with a branch to the insert code.
- Append the value (instruction) originally at the specified location to the end of the inserted code.
- Follow that with a branch back to the specified location plus one.

Following the execution of an insertion, the instruction originally at the insertion address is executed at a location elsewhere in memory. Hence, the insertion address must be carefully selected. The addresses of location-dependent instructions must not be specified. For example, it is not possible to perform an insertion on a BAL (Branch and Link) instruction when the called subprogram can return to addresses BAL+2, BAL+3, etc.

DEBUG SERVICE USAGE

This example is not designed to detail all the syntactic possibilities of the Debug commands and procedures. It is intended to illustrate how the commands of the Debug processor are used for the case of a program which does not have a tree structure.

```
!DEBUG
:SNAP     ADR1,SNAP1,(+34A0,+35FF)
:COUNT    FLAG1,+2A12,22,102,5
:SNAPC    FLAG1,+28F5,COMPUTER,
          (DEB1-A,FIN1)
:IF       FLAG2,ADR2+D,(*CHAINE1,3,2,GT,
          CHAINE2,0)
:OR       FLAG2,ADR3-4,(COND1,5,4,LE,
          *COND2,5,4)
:AND      FLAG2,ADR4,(SWITCH,4,6,28)
:SNAPC    FLAG2,ADR5,LOGICAL,(DEB2,
          FIN2+12)
:PMDI     (DEB1-A,FIN1),(DEB2,FIN2+12)
:PMD      (00),(01)
!EOD
```

Since this example does not use the :DCB command, the data output automatically occurs on the OUT device (listing log).

The :SNAP card causes the output of the comment "SNAP1" followed by the image of the memory area between absolute addresses +34A0 and +35FF. This output is unconditional,

and occurs just before execution of the instruction referenced by the external label ADR1.

The :COUNT command creates an internal counter. This counter is incremented by 1 (starting at 0) each time program execution arrives at absolute address +2A12. The flag, FLAG1, is set if the counter value is 22 or some multiple of 5 plus 22 (i.e., 22 + N*5) in the interval 22 to 102, inclusive. (These are decimal values.)

The first :SNAPC command causes the output of the comment "COMPUTER" followed by the memory area contained between relative addresses DEB1-A (A is a HEX number) and FIN1. This output is performed just before executing the instruction located at absolute address +28F5. It is conditional and occurs only if the flag, FLAG1, is set. FLAG1 is set in its initial state; its state is changed via the :COUNT command.

The :IF command compares two memory fields: two bytes at the indirect address CHAINE1 indexed by the contents of register 3, and four bytes (default value) at address (CHAINE2) not indexed (value 0). This comparison is made just before the execution of the instruction located at relative address ADR2+D (D is a HEX number). If the first field is greater than the second, the flag FLAG2 is set, if not it is reset to 0.

The :OR command compares two memory fields: four bytes at address COND1 indexed by register 5 and four bytes at indirect address COND2 indexed by register 5 (the field size 4 may be omitted from the two fields, since it is the default size). The comparison is made just before executing the instruction located at relative address ADR3-4. If FLAG2 is set, it remains set; otherwise, it is only set if the first field is less than or equal to the second.

The :AND command tests the state of bits 4,6, and 28 in the Job Switch Word (see !SWITCH command, Chapter 3, and the M:TSS, M:RSS, M:SSS procedures, Chapter 8); the condition is true if at least one of these bits is set to 1. This test is made just before executing the instruction at relative address ADR4. If the flag FLAG2 is initially not set, it remains reset; otherwise, it remains set only if one of bits 4,6, or 28 of the JSW is set.

The second :SNAPC command outputs the comment "LOGICAL" followed by the area of memory contained between the relative addresses DEB2 and FIN2+12. This output occurs just before the execution of the instruction located at relative address ADR5. It is conditional and occurs only if FLAG2 is set.

The :PMDI unconditionally dumps two memory areas at the end of execution; the first is contained between the relative addresses DEB1-A and FIN1, and the second between relative addresses DEB2 and FIN2+12.

The :PMD command dumps the memory areas of protection type 00 and 01 at the end of execution if the job is either aborted or terminated by the execution of the M:ERR system service.

6. FILE ORGANIZATION AND MANAGEMENT

GENERAL CONCEPTS AND FACILITIES

This section presents an overview of XOS file management facilities and the basic concepts underlying these facilities.

TYPES OF INPUT/OUTPUT AND STORAGE DEVICES

The types of input/output devices supported XOS are:

- The low-speed, nonmagnetic, peripheral I/O devices:

  Card readers, 80-column

  Card punches, 80-column

  Line printers, 132-column

- The high-speed, magnetic storage devices:

  Magnetic tape units, 9- and 7-track

  Removable disk storage units

  Rapid Access Data (RAD) storage units (several models).

The latter two types of devices are referred to collectively as direct-access devices.

(Remote terminal devices, typically connected through communication lines, are managed by the Telecommunications Management System, and are described separately in Chapter 9.)

Data files input from or output to nonmagnetic devices must be assigned with a DEV (device) type of assignment. (See the !ASSIGN control command.) Although files on magnetic media may also, in certain cases, be assigned as DEV type, the usual assignment type for files on magnetic media is FIL, corresponding to standard labeled files.

Standard labeled volumes and files allow uniform file management facilities, i.e., permit automatic volume and file identification, automatic retrieval of named files on multifile volumes, and in general facilitate the file security features provided by XOS. Volume and file labeling conforming to ANSI standards are automatically provided by the system.

The differences between the several types of input/output devices, and the files assignable thereto, are described in detail in a subsequent section, "Categories of File Media".

LOGICAL VERSUS PHYSICAL FILES

The concept of a logical file as opposed to a physical file allows for a degree of flexibility and physical device independence at job execution time.

A logical file, in XOS usage, is a definition in the user's program of the logical characteristics of the file to be created or accessed. The definition includes such information as record and block structure, file organization (which relates the logical file to an access method), data-encoding mode, etc. This information is contained in a table in the user's program called a Data Control Block, or DCB. The Meta-Symbol user must create, via standard system services, one such DCB for each logical file that he wishes to process. Several distinct physical files, however, of identical logical characteristics, may be created or accessed serially, through one DCB.

All necessary DCBs are automatically created for the COBOL and FORTRAN user.

Lacking in the description of the file in the DCB are the physical attributes of the file: such items as physical device type, identification of the volume(s) on which the file resides or is to reside, the file name (if any), file storage space required (creation-time), etc. In short, all information pertaining to the physical and device-oriented aspects of the file is specified externally to, and kept separate from, the user's program.

We may define a physical file as a collection of related items of information existing on (or to be created on) a specific extent of file storage media, whether that media be, for example, cards or a portion of a disk storage device.

The kind of information needed to identify a specific physical file is typically given by the user in a !ASSIGN control command at job execution time. (The !ASSIGN command is described in Chapter 3; its usage is further described later in this chapter.) Thus, in programming terms, the point of definition of the physical file is the !ASSIGN command.

The DCB and the !ASSIGN command both specify one common item of information: an operational label. This label serves to identify a DCB and therefore a logical file, and constitutes the link between the logical and the physical file: specification of an operational label in a !ASSIGN command associates a logical file to the specific physical resource identified and described in the command.

The procedures for creating, completing, and modifying a DCB are described in Chapter 7.

ACCESS METHODS AND FILE ORGANIZATIONS

XOS provides facilities for six different methods of file processing – referred to as access methods – and four types of file organizations. The six access methods can be divided into two groups, according to the general techniques involved in their use.

- The assisted methods:

  Assisted Sequential

  Assisted Indexed

  Assisted Partitioned

- The basic methods:

  Virtual Sequential

  Virtual Direct

  Basic Direct

The assisted methods operate at the logical record level, and are characterized by a high degree of system-provided services and control: record blocking/deblocking, error checking, volume switching, etc. The basic methods operate at the physical record – or block – level, and are characterized by a high degree of user control and relatively little system service, allowing a corresponding degree of flexibility.

The primary correspondences between the several file organizations and access methods are as follows:

- Sequential organization – Assisted Sequential Access Method (ASAM)

  – Virtual Sequential Access Method (VSAM)

- Indexed organization – Assisted Indexed Access Method (AIAM)

- Partitioned organization – Assisted Partitioned Access Method (APAM)

- Direct organization – Virtual Direct Access Method (VDAM)

The sixth access method, Basic Direct, does not correspond to, nor is limited by, any defined file organization.

ASSISTED VERSUS BASIC METHODS

The unit of data transmitted between memory and an I/O device is referred to as a physical record or block. Depending upon the type of device involved, the length of the block transmitted may be device-determined or may be program-determined. For the nonmagnetic unit-record devices, the length of the block is fixed at the data capacity of the punched card or limited by the maximum length of a print line.

For these devices the logical record (as defined below) is always equivalent to the physical block.

In the case of magnetic-media devices, there is no inherent limit – for most practical purposes – on the length of an input or output block. (The XOS-defined maximum is 32,767 bytes.) An input record read from magnetic tape is limited, however, to a single tape block, as originally written.

The assisted access methods require that the user impose a uniform physical block length for each block of a file assigned to a magnetic medium. Essentially this is done so that a subunit of the block, called a logical record, can be defined. The size (and the type) of the logical record is also user specified. By means of the input/output procedures M:GET and M:PUT, which characterize the assisted methods, the user program works with logical records and the physical blocks. The system assumes responsibility for the management of the physical records; it will block logical records on output and will deblock logical records on input performing the physical block transfers as required. The physical I/O operations themselves are in general transparent to the user program.

If multiple I/O buffers are provided (controlled by the user), the system will anticipate the user's needs on input by performing physical reads in advance of the logical-record requests where access techniques permit; on output it is able to overlap the physical record transmissions with user program execution. Testing for proper completion of I/O operations, and switching from volume to volume of a multivolume file, are functions performed automatically by the assisted methods.

The basic access methods, characterized by the M:READ and M:WRITE input/output procedures, operate solely at the physical block level, and provide the user with direct control of the physical block transfers. Each M:READ or M:WRITE execution requests a physical data transfer. On direct-access media, the amount of data transferred by a single I/O operation is not restricted to the defined block length. No logical record structure is recognized by these methods, and no "buffering ahead" is performed. The user must test for proper completion of each I/O operation requested, using the M:CHECK procedure, and must request any required volume switching.

ASAM AND THE SEQUENTIAL FILE ORGANIZATION

The assisted sequential access method (ASAM) is intended for the creation and sequential processing of files on any type of media.

Sequential processing implies that one logical record after the other is accessed from the file in the order in which they were created (and recorded on the storage medium).

This type of processing and the corresponding file organization is appropriate for the files that can conveniently be created in the natural order of subsequent processing. This type of processing mandatory for files on media that is sequential by nature: card files, print files, and magnetic tape files. (Direct-access media can readily be accessed in a sequential manner.) The primary advantage of the sequential method and organization is device independence: one program can be so planned as to be applicable to a variety of device types. The system will automatically adjust block size, if necessary, when a file is assigned to a nonmagnetic device.

Update processing is defined as the modification of existing records in place. In ASAM, it is restricited to the replacement of records, without length change, on direct-access media only. Update processing must therefore be restricted to programs intended exclusively for use with disk/RAD files. Portions of an existing magnetic tape file may not be rewritten, but such a file is considered as always extendable. A more detailed description of the sequential file organization is presented under "File Organizations", later in this chapter.

AIAM AND THE INDEXED-SEQUENTIAL FILE ORGANIZATION

The assisted indexed access method (AIAM) is intended for the creation and direct-access processing of indexed-sequential files. The records of such files each contain an identifying key value, supplied by the user as part of the record during file creation. The position and length of the key field within the records are specified by the user (via DCB parameters). The system automatically creates and maintains a key index that allows the user to directly access any individual record by key value, regardless of its position relative to the previously accessed record. All or part of an indexed-sequential file may also be accessed sequentially (increasing key values) provided that the first such access is by key value.

An indexed file must be created in order of increasing key values, but higher-value records may subsequently be added, intermediate-value records may be inserted, and existing records may be modified, with or without length change in variable-length format.

AIAM and the indexed-sequential file organization are applicable only to direct-access storage media. The indexed-sequential organization is described in detail under "File Organizations", later in this chapter.

APAM AND THE PARTITIONED FILE ORGANIZATION

The assisted partitioned access method (APAM) is intended for the creation and processing of sequential-like files that are subdivided into individually accessible portions called partitions. The beginning of each partition is identified by one or more user-assigned names, referred to as partition keys. An individual partition is referred to by key in order to enable access to the first logical record contained in it. Thereafter, logical records are accessed from the file sequentially, i.e., in the order of their creation. A complete partitioned file, or any number of contiguous partitions within the file, can be accessed sequentially. If read access is to cease at the end of a given partition, the user is responsible for recognizing his own previously written end-of-partition signal. Hence partitions may be arranged in a nested, or hierarchical, structure.

Update processing, i.e., logical record replacement without length change, is permitted in APAM. Partitions may be added after initial file creation, existing partition keys may be given synonyms, and partition keys may be deleted.

The partitioned file organization is extremely useful for a library-like arrangement and storage of similar groups of information, analogous to the arrangement of books relating to the same subject on a library shelf. An example of partitioned file usage is for the storage of program or subprogram libraries.

APAM and the partitioned organization are applicable only to direct-access storage media. The partitioned organization is described in detail under "File Organizations", later in this chapter.

VSAM AND THE SEQUENTIAL FILE ORGANIZATION

The virtual sequential access method (VSAM) is intended for the creation and sequential processing of files, at the block level, on any type of media. (Not applicable to symbiont files.) The file organization produced is similar to that produced by ASAM, but no logical record structure is recognized or provided by the system. On direct-access media, however, the length of the physical records read or written via VSAM is not constrained by the defined block length, but reads or writes must always begin on a block boundary.

The sequential organization is described further under "File Organizations", later in this chapter.

VDAM AND THE DIRECT FILE ORGANIZATION

The virtual direct access method (VDAM) is intended for the creation and direct-access processing, at the block level, of files on direct-access storage media. Blocks of a direct-organization file may be accessed in any order, by specification of a block sequence number. (Block sequence numbering is not dependent on the order of creation of the blocks of the file.) Typically, the user will set up a relationship between a given block sequence number and the content, or range of content, of the block so identified. The length of physical record read or written is not

constrained by the defined block length, but in any case each I/O operation always begins on a block boundary. The direct file organization is further described under "File Organizations", later in this chapter.

BASIC DIRECT ACCESS METHOD (BDAM)

The basic direct access method (BDAM) is intended for access to a private or unlabeled direct-access volume by real physical sector addressing. No file structure or volume/file labeling is recognized. It is completely physical device oriented, and its use requires a thorough knowledge of the "hardware" characteristics of the disk unit being accessed. It is, however, the most efficient access method for direct-access media. (The system disk, i.e., any device assigned for system control and use, is protected from access via this method.) Under BDAM, the concept of file organization is completely user determined; none of the XOS-defined file organizations preclude use of BDAM however.

ALLOCATION OF SPACE ON DIRECT-ACCESS MEDIA

At file creation time, the user must inform the system of the maximum amount of space that a file on disk pack or on RAD will require. This is done by means of the SIZ option of the !ASSIGN command (or other form of assignment) that defines the physical attributes of the file to be created. When the file is opened for creation, the system will reserve space according to SIZ values specified and the file organization. This space reservation function is called space allocation. The specified amount of space may be allocated as one continuous area on the volume, or as a number of smaller, noncontiguous areas, as dictated by the size of the individual areas available at the time of allocation and the type of device. (On disk pack the system allocates storage in such a way as to minimize arm movement during processing.)

Space is allocated in units of 8,192 bytes (2,048 words); this unit is defined as a quantum. The SIZ option allows the specification, in quanta, of two parameters: the size of a primary amount of space to be allocated at open time, and the size of the supplementary space(s) to be added in case of overflow or extension. The precise sense of the SIZ parameters depends on the organization of the file to be created. This is described in detail under "Space Allocation", later in this chapter, where methods of calculating space requirements for indexed and partitioned files are also presented.

If the user omits the SIZ option on the !ASSIGN command for a file to be created, the system will assign installation-determined default values.

ACCOUNT VOLUME AND FILE RETRIEVAL

At the time an account number is authorized, the user (or group of users sharing a single account number) is assigned an account volume, which may be a disk pack or a defined portion of system-disk space. (See "Categories of File Media", below.) If volume information is not specified on the !ASSIGN control command used to assign a permanent file, the file will automatically be created on or retrieved from the user's account volume.

If a permanent file is to be created on a volume other than the user's account volume, the media type and possibly the volume serial number must be specified. If such a file, when created, is cataloged via the user's account volume, on subsequent access the system can automatically retrieve the file simply by file name, locating the file's volume of residence by reference to the user's account volume. The cataloging of files is fully described under "Cataloged Files", below.

FILE DELETION AND UTILITY PROCESSOR SERVICES

Once having created a number of disk, RAD, or magnetic tape files, the user will eventually want to delete some of these files as they become obsolete, either in favor of newer ones, or simply to free the occupied space in the case of disk or RAD files. Disk, RAD, and magnetic tape file space can be reused by creating a new file that overwrites the old one. Caution is needed in the case of multifile tape volumes, however, as all files subsequent to the file will be lost. Note that no new space allocation can be made for a disk/RAD file that is to be rewritten, i.e., a new file replacing an identically named old one will occupy the space allocated to the original file.

In the case of disk or RAD files, the DELETE function of the utility processor FMGE can be used to simply delete the file (regardless of expiration date). FMGE cannot delete a password protected file, however.

In addition to file deletion, file reorganization may periodically be required in the case of indexed-sequential and partitioned files. Due to repeated extension and/or updating (record or partition deletion, modification, or insertion), the physical arrangement of these files may become inefficient, thereby wasting space and causing increases in file processing time. The two utility processors REORGI and REORGP physically reorganize indexed-sequential and partitioned files respectively.

The FMGE and REORGI/REORGP processors are described in the XOS/UT Reference Manual, Publication 90 17 69.

FILE CHARACTERISTICS AND CLASSIFICATIONS

This section describes the logical structure and organization of the several kinds of files defined by the system as standard, i.e., those for which XOS provides processing facilities. It also defines the several classifications of these files in relationship to other elements of the file system

environment; that is, in relationship to storage media, account catalogs, labeling, etc. Some of the structures and organizations apply only to files on direct-access media (RAD and disk packs), some apply to these as well as to magnetic tape files, and some include nonmagnetic (or unit-record) files also. The range of applicability will be noted in the following descriptions.

LOGICAL STRUCTURE

The logical structure of a file includes both (1) the form of the logical records that constitute the file, and (2) the optional grouping of the records into blocks within the file. Both of these file characteristics, record format and blocking, are defined by the program that creates the given file.

These characteristics are defined in the user's Meta-Symbol program by means of the DCB-related system procedures discussed below under "Program/File Relationship", and/or the !ASSIGN command. The system, however, will supply default values in lieu of programmer-supplied specifications, and when reading a standard labeled file the system will automatically obtain the necessary information from a label associated with the file.

File organization is, in a broad sense, also an aspect of a file's logical structure, but it is separately described in a following subsection.

The specific compiler-language programming facilities for supplying file structure information are described in the appropriate language reference manual; the relevant system procedures for the Meta-Symbol programmer are described in Chapter 7.

PRELIMINARY DEFINITIONS

The following definitions are presented as a preliminary to the specific record and block format descriptions.

Data. We may intuitively take data to mean information to be processed. To facilitate the system's handling of the actual I/O transmissions for the user, additional information describing certain characteristics of the original data may be added by the system. This added data-describing information is transparent to programs using assisted access methods.

Logical Record. A logical record is defined as a collection of related data items that constitutes the basic unit of data for input/output processing.

Note: Hereafter the word "record" will be employed exclusively to mean logical record unless "physical record" is specified.

Block. A block is defined as a set of one or more contiguous records, plus control information (if any), that constitutes a unit of data for input/output transmission.

File. A file in the general case is defined as a collection of records.

RECORD FORMATS

There are three possible record formats; fixed length (F), variable length (V), and undefined (U). All of the data records within a file must have the same format. Format F and V records may be grouped into blocks on magnetic storage media at the user's discretion. A format U record is by definition equivalent to a block. The length of a record in any format may not exceed 32,767 bytes, regardless of storage device.

The record-level parameters of the DCB, described in detail in Chapter 7, are identified here for reference in the following text:

FRM – record format parameter (F, V, or U)

REL – record length parameter

KYL – key length parameter (for indexed-sequential and partitioned files)

KYP – key position parameter (for indexed-sequential files)

DLC – record-deletion control parameter

MXL – maximum I/O-transmission length parameter (for basic access methods only)

FIXED LENGTH (F) FORMAT

The fixed length (F) record format implies that the length of all records in the file is constant and is defined by the REL parameter of the DCB. Only F format records are applicable to punch-card files.

Usage Rules.

1. Applicable to all devices (magnetic and nonmagnetic media).

2. Applicable to sequential, indexed, and partitioned file organizations.

3. Recognized only by assisted access methods.

4. The entire record contains only user data (but see rule 6, below).

5. For files on direct-access media only, the first byte (byte 0) of the record may be specified as a record-deleted control byte via the DLC parameter of the DCB, during file creation only, when using an assisted access method.

6. The deletion control byte (if specified) may be utilized as a data byte, but if it is either created as or modified by the user to the value X'FF', the record is considered deleted by the assisted access methods.

7. If DLC is specified for an indexed-organization file, The record-key field must not overlap the deletion control byte (i.e., KYP $\neq$ 0), or a program abort will occur.

VARIABLE LENGTH (V) FORMAT

The variable length (V) record format is shown in Figure 6-1. This format implies that the length of all records in the file may not be constant, and that the length of the individual record is defined within the record. The maximum length of the record body, i.e., the user-supplied portion of the record is defined by the REL parameter of the DCB.

Usage Rules.

1. Applicable to magnetic storage media and to print files.

2. Applicable to sequential, indexed-sequential, and partitioned file organizations.

3. Recognized only by assisted access methods.

4. The record contains both system data (record header) and user data (record body): the user builds or otherwise supplies only the record body for output and receives only the record body on input; the system manages the 4-byte record header.

5. For direct-access files, the deletion control byte may be specified and used as stipulated for F-format, rules 5-7.

6. Zero-length records may be written and read.

UNDEFINED (U) FORMAT

The undefined (U) record format is shown in Figure 6-2. This format implies that the length of all records in the file may not be constant, and that record length and actual block length are equivalent. The maximum record length is defined by the REL parameter of the DCB. The REL value may be less than or equal to the BKL parameter value.

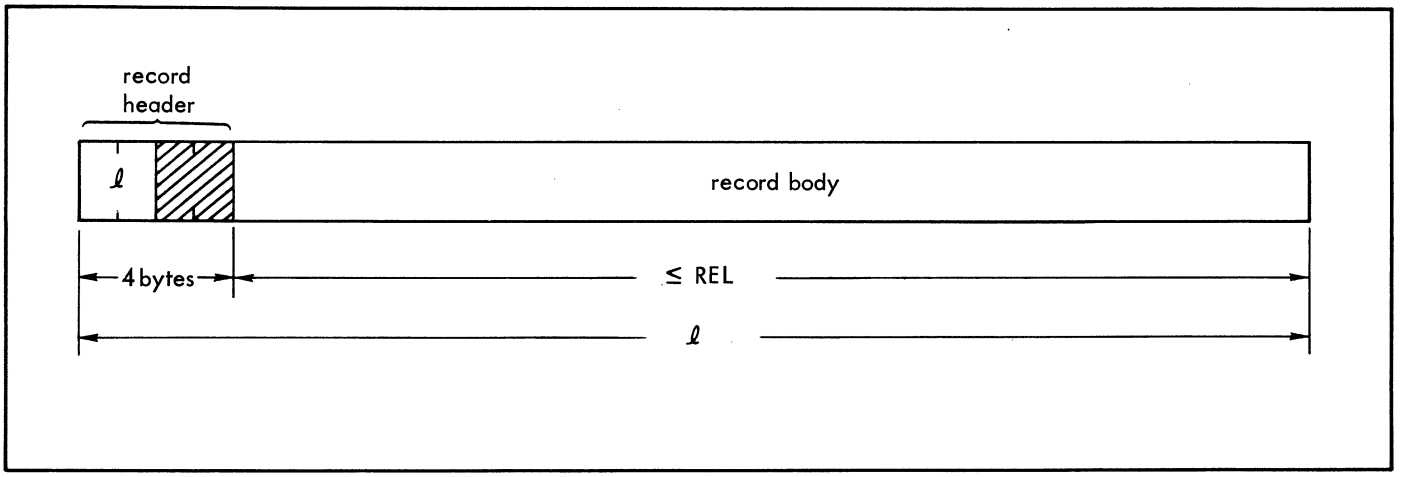


Figure 6-1. Variable-Length (V) Record Format

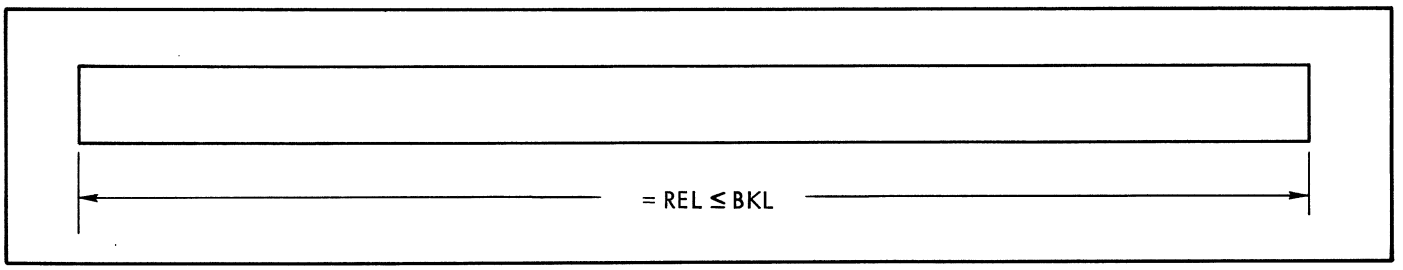


Figure 6-2. Undefined (U) Record Format

Usage Rules.

1. Applicable to magnetic storage media only.

2. Applicable for file creation to sequential file organization only.

3. Recognized by assisted sequential access method (ASAM) only.

4. The entire record contains only user data.

5. Record and block length are equivalent and on sequential-access media are equal to the physical record length.

BLOCK FORMATS

A block on magnetic media can contain one or more logical records; its length is program defined (BKL parameter of the DCB). A block on a nonmagnetic device, e.g., card punch, line printer, can contain only one record; its length is limited by the physical-record size intrinsic to the particular device. In the case of the card punch, the block length is fixed, and depends upon the data-encoding mode (80 for EBCDIC, 120 for binary). The corresponding program definition (BKL parameter) may not be less than 80 for card reader or card punch files.

On magnetic tape, the actual length of the blocks within a file may vary; the length of a tape block may be less than or equal to the BKL parameter value. ("Short blocks" are caused by data transmissions of less than BKL length.)

On direct-access media, a block is of fixed length, and always starts on a sector boundary. (Each sector can contain only one complete or partial block.) The block size for direct-access media is defined by the BKL parameter of the DCB.

The default values for the BKL parameter are:

- 1024 bytes for files on magnetic media.
- 133 bytes/characters for line printer files.
- 80 bytes/characters for card files.

The length of a block cannot exceed 32K-1 bytes in any case, including block header, record headers, and any other system-required information.

The possible block formats, which vary with differing combinations of record format, user block-header options, and storage media, are shown in Figures 6-3 through 6-8 and are further described below. (Note that block formats are given only for files on magnetic media; block control information is never affixed to card- or print-file records.)

In certain instances there may be unused space in a block on direct-access media. For example, if the number of records written is insufficient to fill the last block completely, and the file is closed at that point, the system will write the block even though the block contains only one record. Intentional residual space may be introduced by use of the M:TRUNC procedure, described in Chapter 7. One reason for using M:TRUNC in this manner is to leave space in an indexed-organization file so that additional records may be inserted into the file without creating overflow blocks within the file.

In general it is desirable to explicitly define the block size as an even multiple of record size (plus block and record headers, if any) to avoid introducing residual space into every block. This provides the most efficient utilization of both file space and I/O buffer space (see Chapter 7). For files on direct access media, it is also important to determine block size in relationship to the sector size of the device in question; see "Space Allocation" later in this chapter.

The block-level parameters of the DCB, described in detail in Chapter 7, are identified here for reference in the following text:

BKL – block length.

BHR – block header length, for F or V record formats.

NBC – indicates that block sequence numbering (on magnetic tape) is to be suppressed.

FIXED FORMAT MAGNETIC TAPE BLOCKS

Blocks on magnetic tape containing fixed (F) format records can contain either (1) records only, or (2) block header information and records. The user may eliminate the block header by specifying the NBC (No Block Count) parameter and defaulting the BHR parameter (or specifying BHR, 0). Note that the default value for BHR in F-format is 0 only if NBC is specified also. Figure 6-3 shows an F-format magnetic tape block with block header eliminated.

Figure 6-4 shows an F-format magnetic tape block with the 4-byte block header produced by accepting the defaults for the BHR and NBC parameters. The two bytes contain a binary block sequence number, relative to the first block (block 0) of the file.

The general form of the F-format block header is shown in Figure 6-5; the results of varying combinations of BHR- and NBC–option specifications are given in Table 6-1.

Note that the block length value (BKL) must allow for the additional length of the block header in order to achieve the desired blocking factor for a given record length.

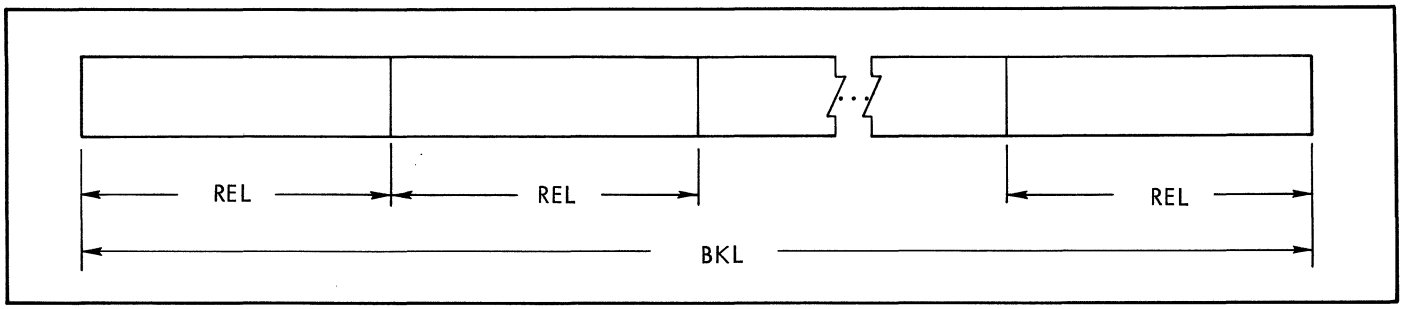


Figure 6-3. Structure of a Fixed Format Block on Magnetic Tape with BHR Defaulted and NBC Specified

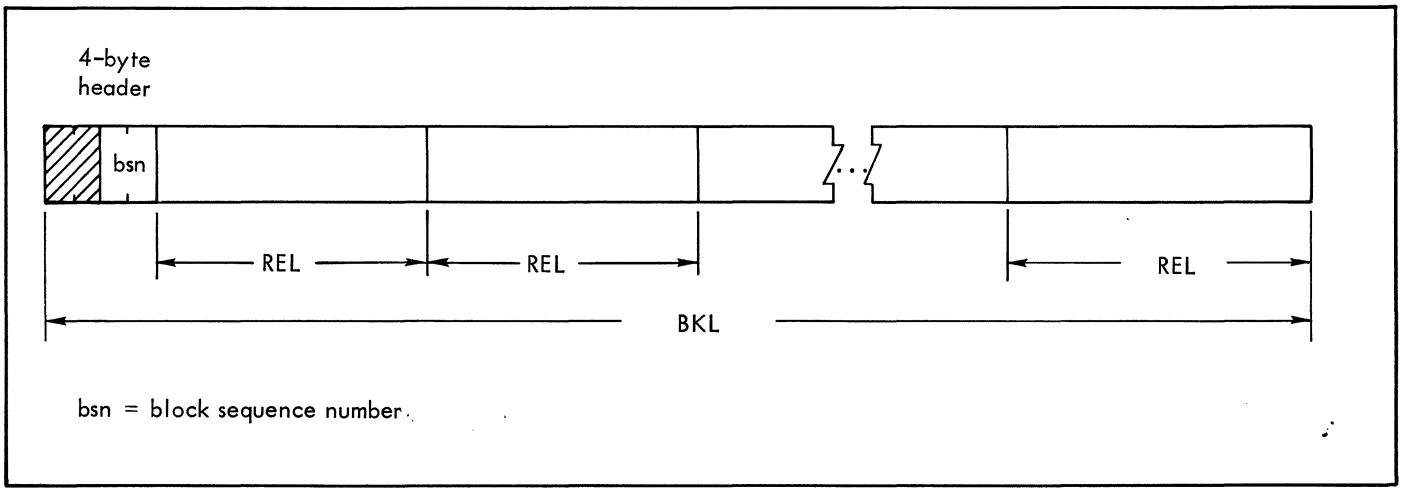


Figure 6-4. Structure of a Fixed Format Block on Magnetic Tape with Default 4-Byte Header

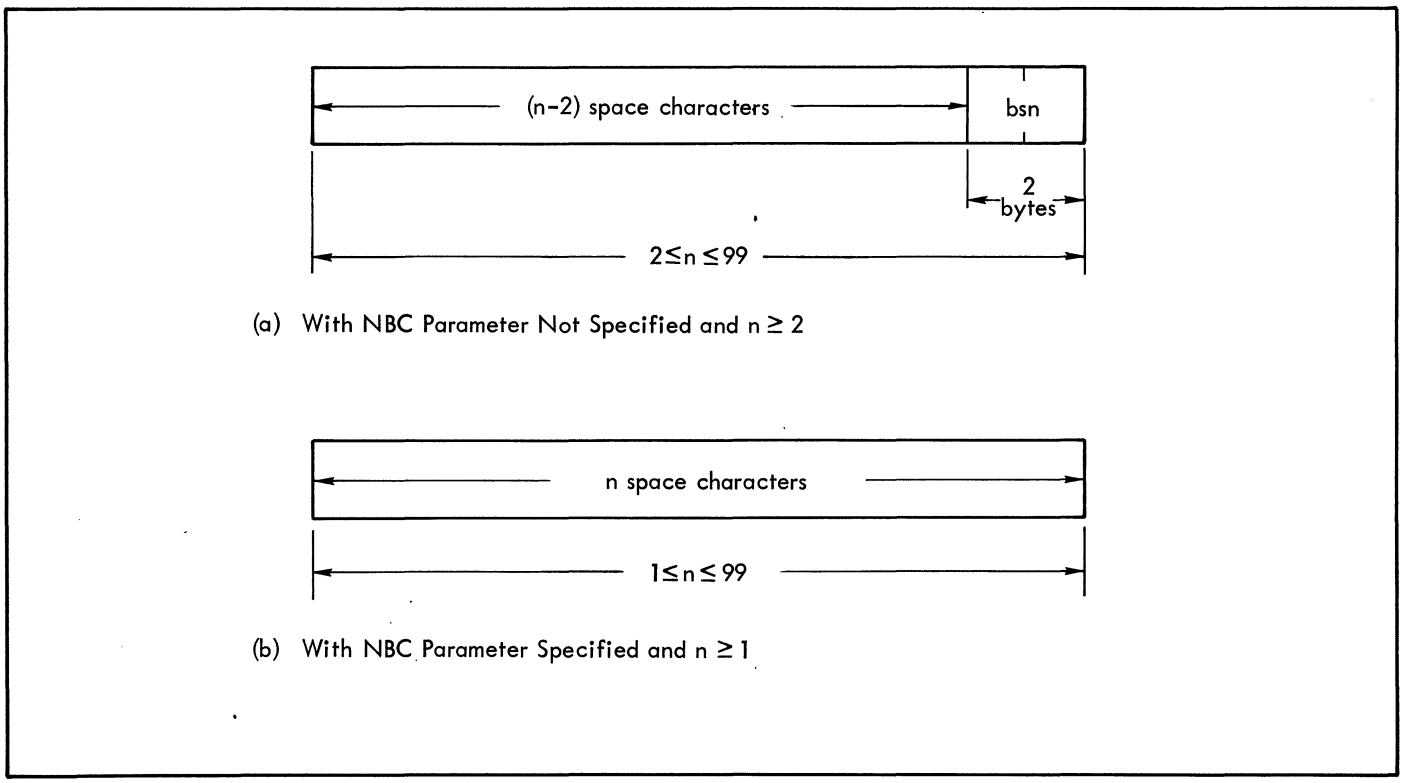


Figure 6-5. F-Format Block Header on Magnetic Tape, with BHR Value, n, Specified

Table 6-1. Effect of BHR and NBC Parameters for F-Format

Results for F-Format Magnetic Tape Blocks					
(BHR,n)	NBC not Specified			NBC Specified	
		Header Contents			
n	No. of Header Bytes	Leftmost Bytes	Rightmost 2 Bytes	No. of Header Bytes	Header Content
default	4	2 spaces	bsn	0	none
0	Same as default			Same as default	
1	Error (abort)	—	—	1	space
2≤n≤99	n	(n-2) spaces	bsn	n	n spaces
Legend: bsn - block sequence number					

VARIABLE FORMAT MAGNETIC TAPE BLOCKS

Blocks on magnetic tape containing variable (V) format record always contain block header information. The minimum size V-format block header consists of a 2-byte field containing the actual block length (in binary), including the length of the header itself. The minimum block header is obtained by defaulting the BHR parameter (or specifying BHR,0) and specifying the NBC parameter.

The default form of V-format block header, obtained by defaulting both the BHR and NBC parameters, is four bytes long, the left two containing the actual block length (including header length) and the right two containing the block sequence number, in binary, relative to block 0 of the file. This block format is shown in Figure 6-6. The general forms of the V-format block header are shown in Figure 6-7. The results of varying combinations of BHR- and NBC-parameter specifications are given in Table 6-2.

Note that the maximum length of a V-format block is defined by the BKL-option value; the block header length must be allowed for when specifying this value in order to achieve an optimum blocking strategy. Although the blocking factor of an individual V-format block cannot in general be predetermined, a mean can be estimated on the basis of average record length expected.

BLOCKS ON DIRECT-ACCESS MEDIA

The blocks of a file on direct-access media are of fixed length (defined by the BKL of the creating program) and contain a 4-byte block header, if created by an assisted access method. The leftmost two bytes contain the effective block length, i.e., the length of the useful data actually written in the block, plus 4 for header length. The right two bytes of the header are zero. When the file is created by a basic access method, the block header is absent.

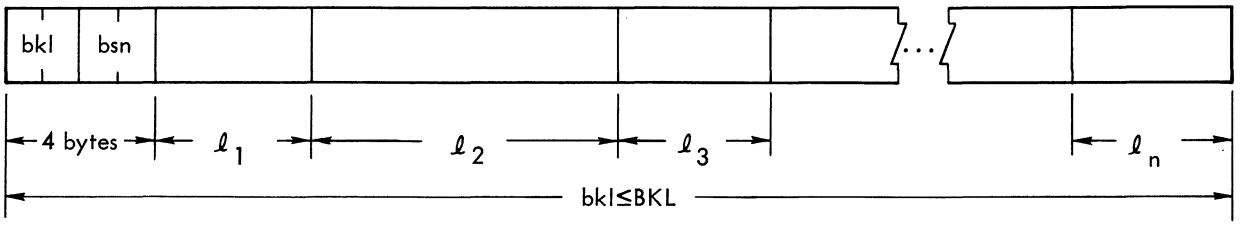
If the file is created in indexed-sequential organization, the last four bytes of each block will be reserved by the system for block linkage purposes, and must be accounted for in the BKL value in order to optimize blocking strategy.

The general format of an assisted-method created block (for F or V format records) on direct-access media is shown in Figure 6-8. Note that the only valid BHR parameter specifications are BHR,0 and BHR,4: these are both equivalent to the default; note also that the NBC option is not relevant. The user's block length (BKL) specification should allow for the 4-byte header, and also possibly for the 4-byte linkage field at the end of block, depending upon the access method and record format used for file creation.

When using the basic virtual access methods, VSAM and VDAM, the virtual physical record written or read may extend over a number of contiguous, i.e., virtually sequential, blocks of the file.

FILE ORGANIZATION

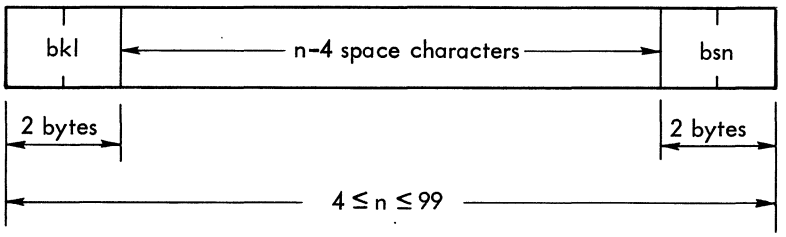
The organization of a file is determined by the user's choice of access method for file creation. From the user's viewpoint, file organization concerns the logical ordering of, and relationships between, the data blocks and system-supplied blocks containing control information (if any). The logical ordering of the blocks also reflects the physical organization of the information on the storage media, but in



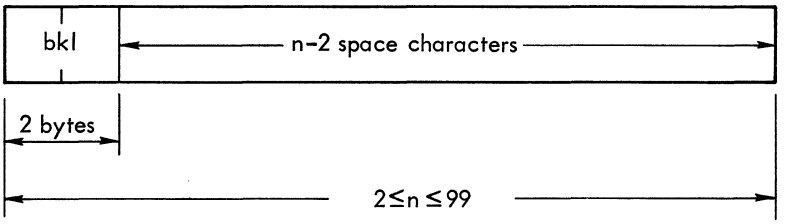
bsn = block sequence number

ℓ_i = record length ≤ REL

Figure 6-6. Structure of a Variable Format Block on Magnetic Tape with BHR Defaulted and NBC not Specified



(a) With the NBC Parameter Not Specified and n ≥ 4



(b) With the NBC Parameter Specified and n ≥ 2.

Figure 6-7. V-Format Block Header on Magnetic Tape with BHR Value, n, Specified

Table 6-2. Effect of BHR and NBC Parameters for V-Format

Results for V-Format Magnetic Tape Blocks							
(BHR,n)	NBC not Specified				NBC Specified		
		Header Content				Header Content	
n	No. of Header Bytes	Leftmost 2 Bytes	Middle Bytes	Rightmost 2 Bytes	No. of Header Bytes	Leftmost 2 Bytes	Rightmost Bytes
default	4	bkl	none	bsn	2	bkl	none
0	Same as default				Same as default		
1	Error (abort)				Error (abort)		
2	Error (abort)				2	bkl	none
3	Error (abort				3	bkl	1 space
$4 \leq n \leq 99$	n	bkl	(n-4) spaces	bsn	n	bkl	(n-2) spaces

Legend: bkl – block length (always ≤ BKL)
bsn – block sequence number

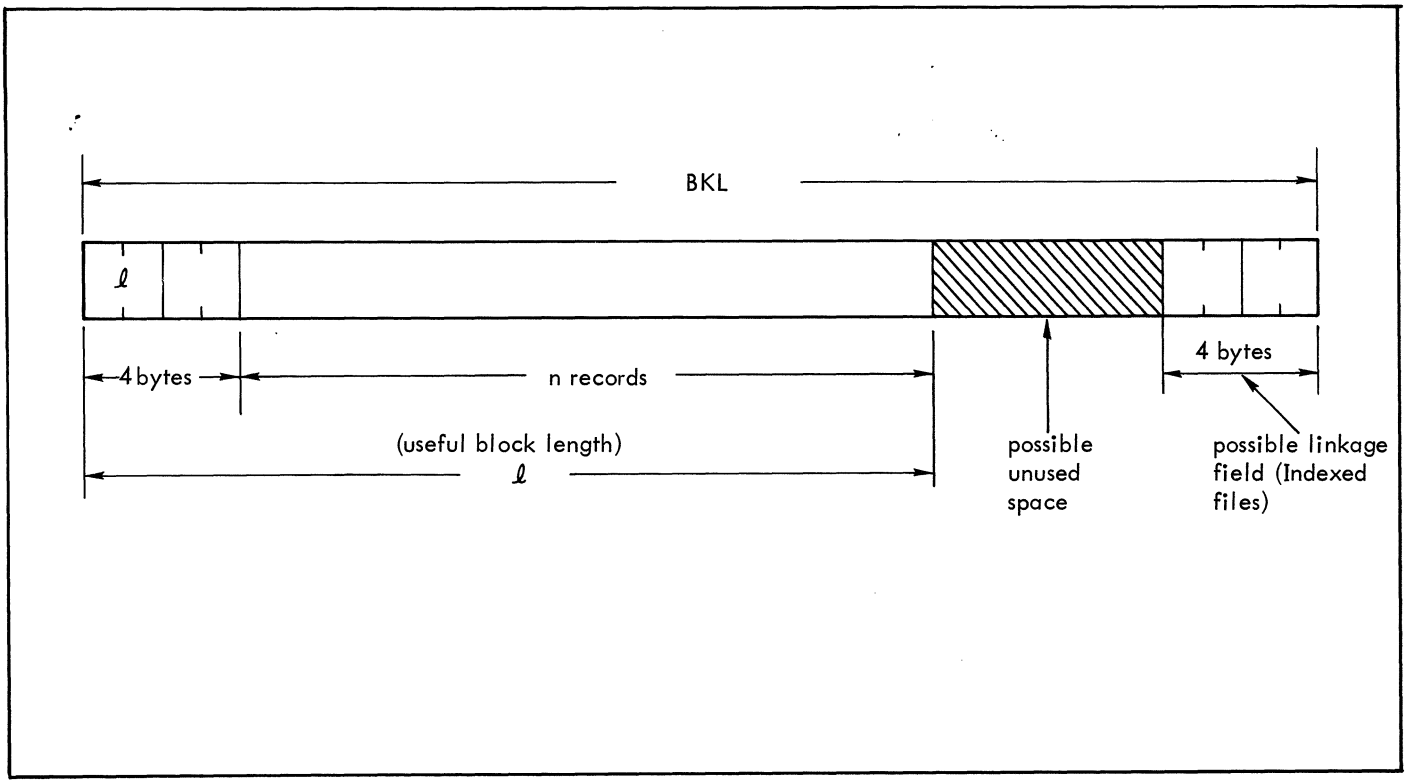


Figure 6-8. Structure of a Direct-Access Data Block (Assisted Methods)

the case of direct-access media the actual physical disposition of information is determined solely by the system and thus is transparent to the user. (The use of the basic direct access method, BDAM, implies no file structure or organization whatever, and can only be used with private or non-standard disk packs.)

The four possible file organizations, and the media to which they apply, are

- Sequential (C) – all media.
- Indexed-Sequential (I) – direct-access only.
- Partitioned (P) – direct-access only.
- Direct (D) – direct-access only.

Although, in general, each file organization corresponds to a particular access method for file creation, several access methods may apply for subsequent access to a file of given organization. For example, a partitioned file can be read by the assisted sequential, assisted partitioned, virtual sequential, and virtual direct access methods.

SEQUENTIAL (C) ORGANIZATION

The sequential file organization permits sequential access to the records or blocks of a file. It is created by either ASAM or VSAM, and is the only organization applicable to nonmagnetic device files as well as to files on magnetic media.

Depending upon file media restrictions, any of the three record formats, F, V, or U, are allowed with use of the assisted sequential access method. Although a sequential file can be written or read at the logical-record level by ASAM, it can be read (or written) only at the block level by VSAM.

Existing sequential files on magnetic tape are always extendable – at the cost of losing any subsequent files on the same volume. On direct-access media, they may be extended up to the limits of the possible space allocation; also individual records may be deleted, or modified if the record length is not changed.

INDEXED-SEQUENTIAL (I) ORGANIZATION

The indexed-sequential file organization permits either direct access to individual logical records identified by record key, or sequential access to records in ascending order of their keys, starting with a specified record. A record key is a data item within the record body, provided by the user, which serves to uniquely identify the record. The location of a record specified by key is determined (by the system) via an index mechanism that is constructed and maintained by the system as part of the file.

Indexed-sequential organization is applicable only to direct-access media. Either F- or V-format records are allowed. An indexed-sequential file is created using the assisted indexed access method (AIAM).

The indexed-sequential organization is shown schematically in Figure 6-9. The file is composed of data blocks, index blocks, and (possibly) overflow blocks. Upon creation, the file will consist of one or more data blocks, and at least one index block. The index may be multilevel, as illustrated in Figure 6-9 (1st and 2nd level index blocks). The number of index blocks and number of levels thereof is a function of block size, number of data blocks, and record-key length (as described below).

Prior to file creation, the user must request allocation of sufficient file space to allow for all of the data, index, and overflow blocks that may eventually be needed. The method for calculating this space requirement is described below under "Space Allocation". Also prior to creation, he must describe to the system both the beginning byte position, relative to byte 0 of the record body, and the length of the record key by means of the DCB parameters KYP and KYL respectively.

During file creation, the user must create the record keys and write the logical records in ascending record-key order (binary collating sequence); if a record is presented out of ascending key order it is not accepted and an abnormal condition occurs.

For each base data block written, a record-index entry is automatically created. It is composed of the record key corresponding to that of the last record in the data block and a pointer to the beginning of that block. The record-index entries are blocked as are user records, and the set of these blocks constitute the first-level index.

For each first-level index block written an index entry is created. It is composed of the record key corresponding to that of the last record-index entry in the first-level index block and a pointer to the beginning of that block. These index entries are blocked, similarly, into the second-level index.

Given enough data blocks, the above process applies recursively with third, fourth, . . . , 255th level indices produced. In general, at least one (partial) index block exists at any level when two or more blocks exist at the next lower level – including the "data block level".

Overflow data blocks are created if, during subsequent updating, either inserted or lengthened records cause original records to be "pushed down" beyond the boundary of a data block. The resulting overflow is automatically moved to an overflow block which is linked between the two data blocks as shown in Figure 6-9. Two or more overflow blocks can be linked between two data blocks in this manner. Note that overflow blocks do not appear explicitly in the index, and are undesirable from the viewpoint of access speed and storage space utilization. A utility processor, REORGI, is provided to effect a reorganization of overflowed indexed sequential files. (See the XDS Utilities Reference Manual.)

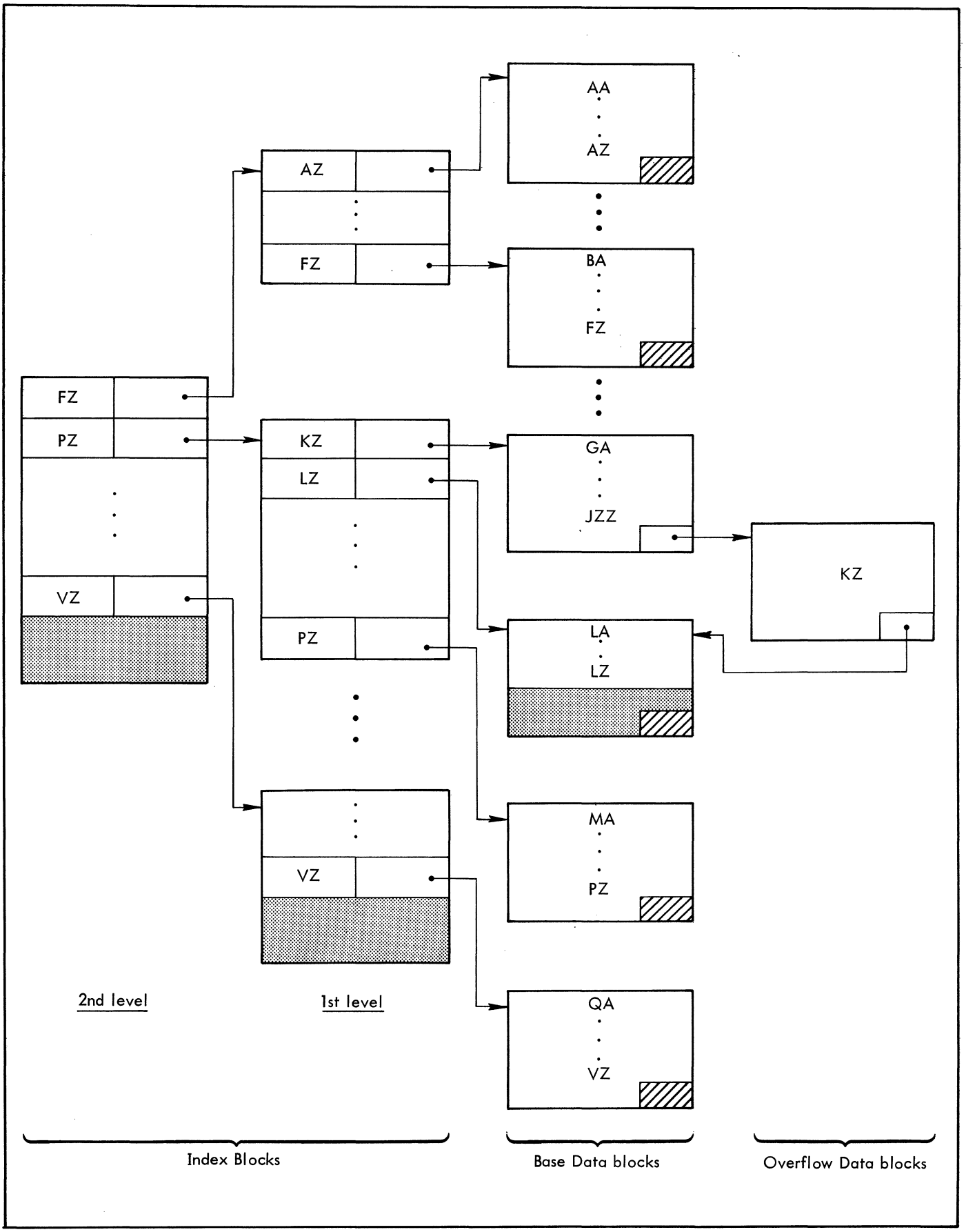


Figure 6-9. Indexed-Sequential Organization

At the end of the file creation process, the system automatically inserts a dummy record having the maximum possible key value (X'FF...F'). This permits subsequent insertion of records with keys greater than that of the last record originally written, effectively allowing file extension. The dummy last record cannot be accessed by the user program, however.

Care must be taken that the key field does not overlap the deletion control character (byte 0), if the latter is specified; a program abort on file opening will occur if KYP = 0 (default value 0) in this case.

If the ICY (index copy) option of the M:OPEN procedure is specified, the system will automatically copy the index portion of an existing file to a temporary file in secondary storage. This will generally result in faster direct-access processing time, especially if the secondary-storage media is appreciably faster than the media upon which the entire file resides, e.g., RAD vs. disk pack.

PARTITIONED (P) ORGANIZATION

The partitioned organization permits either sequential access to the records of a file, or direct positioning to the beginning of a named partition of a file for subsequent sequential access to the records thereof. This organization is essentially an arrangement of a sequential file into uniquely locatable subfiles.

A partitioned file is created with the assisted partitioned access method (APAM). It is applicable only to direct-access media. Either F or V record format may be utilized.

The partitioned organization is shown schematically in Figure 6-10. Note that the user's data blocks are preceded by, and possibly interspersed with, system-constructed partition key (name) and a pointer to the first logical record in the associated partition. The directory blocks are, however, transparent to the user's program as they are not accessible via assisted access methods. For example, if either APAM or ASAM is used to read a partitioned file, they will "skip over" the directory blocks. However, the user must, prior to file creation, request allocation of sufficient file space to allow for both data and directory blocks. The method for calculating and specifying this space requirement is described below under "Space Allocation".

To create a partitioned file, the user must begin by assigning a partition key (with the M:STOW procedure); that is, the file must contain at least one partition. During creation, the user may create as many partitions as desired. In addition to principal (i.e., first-assigned) partition keys, the user may assign synonym keys, i.e., aliases of a given partition name. The keys may be up to 255 bytes in length.

During subsequent processing of the file, synonym keys may be added, any key may be deleted, and new partitions created. In addition, existing records may be deleted, or be modified if the record length is not changed.

It is important to note that when reading a partitioned file (either with APAM or ASAM), the system does not detect an end-of-partition condition: the user may read to end-of-file, across partition boundaries, whether starting from a partition boundary or from beginning-of-file. A partition key locates the beginning of a partition, but not the end of the preceding one: therefore a partitioned file may be given a hierarchical, or "nested", structure by the appropriate ordering of subsumed partitions. (End-of-partition may, of course, be signaled by a user datum detected by the program, e.g., a zero-length record in V-format.)

The pointer portion of a directory entry contains the relative block number of the block in which the associated partition begins, and the byte displacement of that partition's first record. (Synonym entries contain, in addition, a synonym indicator.) The partition keys are sorted, when necessary, and maintained in ascending order of key value within the directory block chain.

DIRECT (D) ORGANIZATION

The direct organization permits direct access to blocks of a file by relative block number (in relation to the beginning of the file, block 0), via the VDAM access method only. It is an "unmanaged" organization relative to the C, I, and P organizations.

A direct-organization file is composed of blocks of BKL defined (or default 1024-byte) length, and transmission must begin on a block boundary. However, the length of the data actually transmitted is specified in the M:READ or M:WRITE procedure by the transfer-length (TRL) option (default = 1 block). The length of data transmitted may be less than the block length, or may extend over several contiguous blocks, but it is limited by the maximum-transfer-length (MXL) parameter of the DCB associated with the file.

No block header is created in D organization; no logical record structure within the block is recognized by VDAM.

Files created by VDAM are accessable by VSAM and also by ASAM using U record format.

TEMPORARY AND PERMANENT FILES

Files on magnetic media can be either temporary or permanent files. (The distinction is not relevant for nonmagnetic device files, for a number of reasons.) In principle, a permanent file is one that continues to exist in a retrievable form after the execution of the job that creates it; a temporary file does not. That is, a permanent

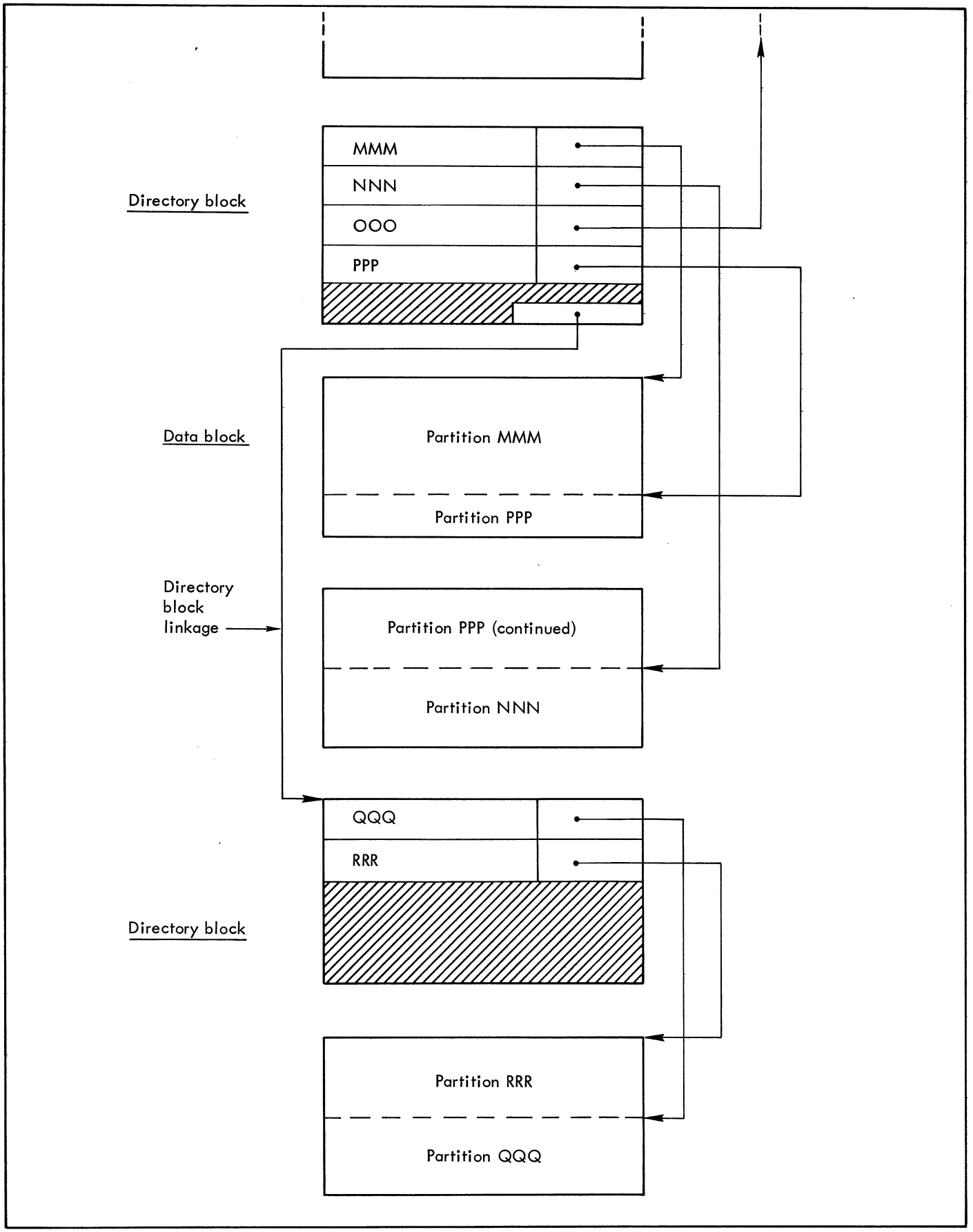


Figure 6-10. Partitioned Organization

file is reusable by another job. In terms of system usage, the following operational definitions apply:

- A permanent file is a named physical file.
- A temporary file is an unnamed physical file.

The permanent-file definition above implies, also, that a permanent file is a labeled file recognizable as such under XOS.

TEMPORARY FILE

A temporary file is generally used for intermediate, or "scratch", storage within a job or job step. It is assigned at creation time with a FIL-type assignment without the NAM option. The file may be created on a public tape or disk volume, but in the absence of any specification to the contrary, the file is automatically assigned to a portion of the system disk reserved for temporary files, known as temporary secondary storage. Thus the simple assignment

```
!ASSIGN op-label,FIL
```

is sufficient to define a temporary file of installation-defined default size that is valid for one job step only (the FRE option is assumed by default). To maintain the same file over several job steps, one only need write instead:

```
!ASSIGN op-label,MTN,FIL
```

The file space reserved for such a file is released automatically at the end of the job step or job, depending upon the usage of the MTN/FRE options.

If assigned to a public volume, a temporary file must be monovolume (contained within one volume). In any case, it may be noted that during execution of the job or job step in which it is defined, the file is known only by the operational label that associates it with a program DCB. Thus upon completion of the step or job, i.e., upon deactivation of that label, the physical file created is "lost" to the system.

PERMANENT FILE

A permanent file is assigned with both the FIL and NAM option specified in the assignment. It may be created on the user's account volume or on a private or public volume; a public volume will automatically become private under the user's account if the creating job-step terminates successfully.

A permanent file may be multivolume except when assigned to the user's account volume (see "File/Volume Relationships" below).

If an existing permanent file is to be accessed from one or more private volumes, the user must specify the volume serial number(s) in the referencing assignment unless the file has been cataloged; whereas the location of a file created on an account volume is inherently known to the system.

CATEGORIES OF FILE MEDIA

This section defines the two major categories, and several subcategories, of file media, from the viewpoint of XOS usage. The purpose of these categories is descriptive: certain sets of file management facilities, file characteristics and classifications, etc., apply to one media category or subcateogry and not to others. The two major categories are (1) nonmagnetic media and (2) magnetic media, the later category dividing into several operational subcategories.

NONMAGNETIC MEDIA

The nonmagnetic media (strictly speaking) supported by XOS are punch-card decks, both for input and output, and printer listings. The devices corresponding to these media are the card reader (input), card punch (output), and line printer (output). The characteristics of the files associated with these devices are device dependent, that is, determined by the inherent physical limitations of device, e.g., the printer's unit line length. Therefore, such files are assigned as device type (DEV) files, via the !ASSIGN control command.

Other devices included in this category are the telecommunications terminals, including remote teletypewriter terminals and remote batch terminals. The processing of all telecommunications devices, however, is treated separately in Chapter 9, as specialized facilities are provided therefore.

Nonmagnetic device files differ in several important operational aspects from files on magnetic media: they have low relative transmission speeds, are not dynamically reusable (i.e., switchable from input to output or vice-versa), and — most important from the viewpoint of system usage — standard labeling does not apply. In contrast, all standard magnetic storage volumes, described below, contain only labeled files.

It is important to note that when a magnetic tape file (or disk pack) is to be accessed as unlabeled, it is assigned with a device-type (DEV) assignment even though it is magnetic media. Thus, the distinction between the DEV and the FIL assignment is actually made on the basis of unlabeled vs. labeled files.

Nonmagnetic device files are also commonly spoken of as "unit record" or "peripheral device" files.

MAGNETIC STORAGE MEDIA

Magnetic storage media includes magnetic tape volumes, disk volumes (demountable disk packs), and system-disk space (secondary storage) which may consist of RAD or disk pack or both. The major operational subcategories are (1) magnetic tape, an intrinsically sequential-access media, and (2) direct-access media, which includes disk and RAD. A secondary distinction is that between demountable volumes and nondemountable volumes, or pseudo-volumes, defined below. ("Demountable" as used here is not synonymous with "physically removable"; see "System Disk", below.)

VOLUME CLASSIFICATIONS

There are four classes of standard volumes: public, private, account, and pseudo-volume. A standard volume is one with standard volume labeling, either as created under XOS or recognizable by XOS, and that contains only standard labeled files. (XOS magnetic tape label standards are ANS compatible.) Hereafter, the term volume, unqualified, will be employed exclusively to mean a demountable standard volume. A nonstandard volume must be assigned with a DEV (device) – rather than a FIL (file) – assignment. The user is restricted to the BDAM access method for the processing of nonstandard disk volumes.

VOLUME PREPARATION

A utility processor, PREP, is provided for standard-volume preparation. This preprocessing of a volume intended for standard usage is mandatory. PREP may also be used to delete all files on a tape or disk volume and/or change volume ownership. Use of the PREP processor is normally restricted to the computer-center operations staff. It is described in the XOS/UT Reference Manual, 90 17 69. Functionally, it writes and initializes the volume header, which includes the volume label, volume catalog, and the allocation table for disk volumes, and comprises the volume label group and tape mark(s) for tape volumes. Any volume can be prepared as either a public or private volume (see below).

PUBLIC VOLUME

A public volume is a tape or disk volume that does not belong to any user account: it does not carry an account identification in its volume label. It can be used by any account for the creation of temporary or permanent files. If a permanent file is created on a public volume, however, the volume automatically becomes a private volume under the file creator's account. The account number of the user is recorded in the volume label during creation of the permanent file.

The volume serial number of a public volume must not appear in the !ASSIGN assignment for a temporary file, i.e., one would specify FIL,(UNT,type) only. (A command syntax error occurs otherwise.) At program exeuction time, a public volume – either chosen by the system from among those already mounted or for which it issues a mounting request – will be allocated to the file.

If a permanent file is to be created, the volume serial number(s) may be specified but need not be. If not specified, a volume is selected on the same basis as for a temporary file.

Public volumes containing temporary files are released to the system, without being systematically dismounted, when the job or job step (depending on MTN/FRE option usage) has finished. They can then be assigned immediately to another job or job step.

PRIVATE VOLUME

A private volume is a tape or disk volume belonging to a given user account. The account identification appears in the volume label. Only the owner of a private volume may create files on it, although access can be granted to other accounts for the purpose of modification or extension of a specific file thereon. Only permanent files may be created on a private volume; the volume serial number(s) must be specified in the file assignment.

ACCOUNT VOLUME

An account volume is a direct-access volume that contains the account catalog for a given account. The account volume is known to the system, since its identification is associated with the account number in the system's super-catalog.

In addition to the account catalog (which is simply a special form of volume catalog), the account volume can also contain premanent files. In general, any file residing on an account catalog can be accessed simply by file identification and account number in the case of another user.

Each account may have only one account volume, and only monovolume files may be created on it.

PSEUDO-VOLUME

A pseudo-volume is by definition an account volume, but one that resides in a dedicated portion of secondary storage (which is in turn a portion of system disk). All usage rules applying to the account volume apply equally to a pseudo-volume, the only distinction between the two being that the latter is not physically demountable.

SYSTEM DISK AND SECONDARY STORAGE

The system disk is that set of RAD and/or disk units that is defined as being reserved for system use. (Any disk pack that constitutes any portion of system disk is logically non-demountable.) Although system disk is under the exclusive control of the system, a portion of it may be defined as secondary storage, which can contain space reserved for temporary files, space reserved for pseudo-volumes, and/or space reserved for symbiont files. The user has access to the temporary-file and pseudo-volume space (if these portions exist); his access to symbiont file space is indirect as its usage is generally transparent to the user.

The amount of system disk reserved for secondary storage, and the size of each of its subdivisions, is installation dependent.

SUPER, ACCOUNT, AND VOLUME CATALOGS

The supercatalog, account catalog, and volume catalogs together provide a "trail" for the system's use in locating user's files efficiently and economically. These catalogs have a number of other functions as well, but these are ancillary from the user's viewpoint. See Figure 6-11 for a schematic representation of this chain of catalogs.

SUPERCATALOG

The supercatalog is maintained on system disk; essentially it is a list of all authorized users of the system, by account number. Each entry in the supercatalog associates an account number with either (1) the volume serial number of an account volume, or (2) the location in secondary storage of an account pseudo-volume.

ACCOUNT CATALOG

The account catalog is essentially a table of entries representing the files that have been cataloged under the account. The account catalog resides solely upon the account volume belonging to a particular account. The account catalog provides information which the system may use to locate a given file specified by file name alone (or by file name and account number). Any file entered into an account catalog can be automatically so located. A file name can be entered into the account catalog under two conditions. When the user creates a file on the account volume the file name is automatically entered into the account catalog. When the user creates a file on a standard labeled volume and specifically requests for the file to be cataloged (via the !ASSIGN control command), the file name and volume serial number are entered into the account catalog. The account catalog is a special case of a disk volume catalog.

It should be noted that a limit upon the number of files that may be entered into the account catalog is established when the account volume is prepared. This limit can be set at either 64, 128, or 256 catalog entries as determined during PREP processing.

VOLUME CATALOG

For a given standard disk volume, the volume catalog is essentially a table of entries representing the files contained thereon. An entry consists of a file's labels along with information concerning file space within the volume. (See "Disk Pack Structures", under Physical File/Volume Structures, and also Appendix C, for further details.)

FILE/VOLUME RELATIONSHIPS

On private magnetic tapes and disk packs, i.e., removable volumes, several file/volume relationships are possible:

- Monovolume file – a file contained within one volume.
- Multivolume file – a file that extends over more than one volume.
- Multifile volume – a volume that contains two or more files (or portions thereof).
- Multifile multivolume – a set of files extending over two or more volumes, the set including at least one multivolume file.

These structures on magnetic tape are illustrated in Figure 6-12.

The system provides several volume mounting options for multivolume files, with either automatic or program-directed (M:CVOL) transition between volumes. For multifile volumes, extensive positioning and identification-checking facilities are provided; these are explained in the M:OPEN and M:CLOSE procedure descriptions in Chapter 7.

Serial mounting of multivolume files implies sequential access to volumes; it is the only mode possible for magnetic tape volumes. Several physical drive units may be reserved, however, permitting premounting of two or more volumes of the sequence. Volume sharing is not possible for disk volumes in serial mounting mode.

Parallel mounting is possible for a set of disk pack volumes; it implies, though does not necessitate, nonsequential access to the volumes (relative both to creation order and to volumes already accessed). Parallel mounting also allows for volume sharing and/or file sharing of multivolumes (if the volumes are sharable); see "File Protection and File Sharing"

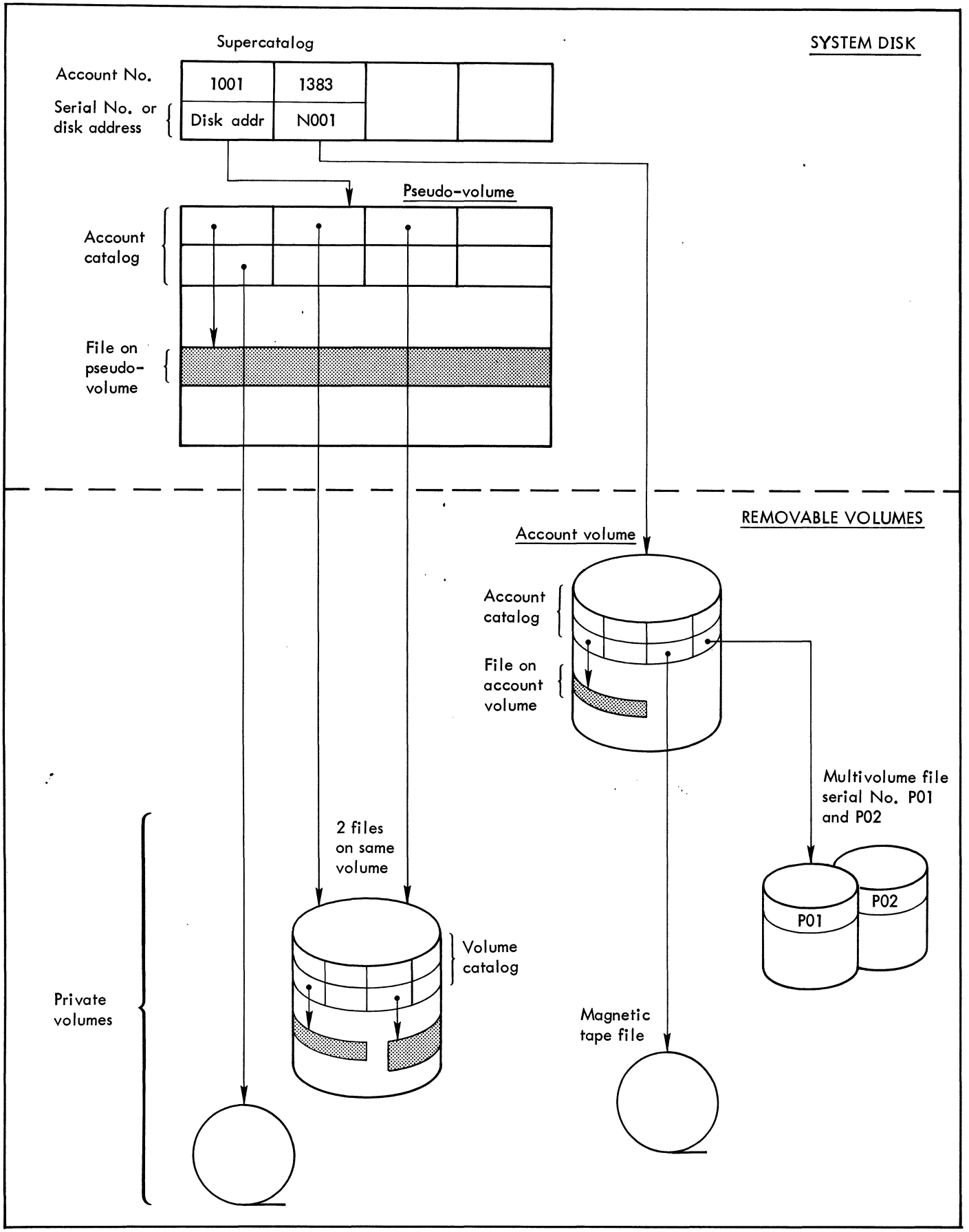
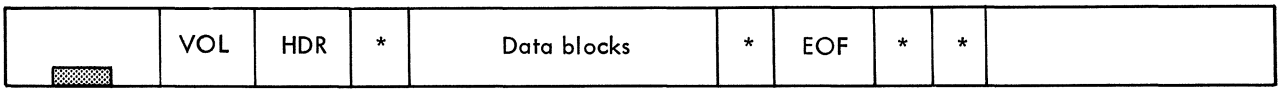


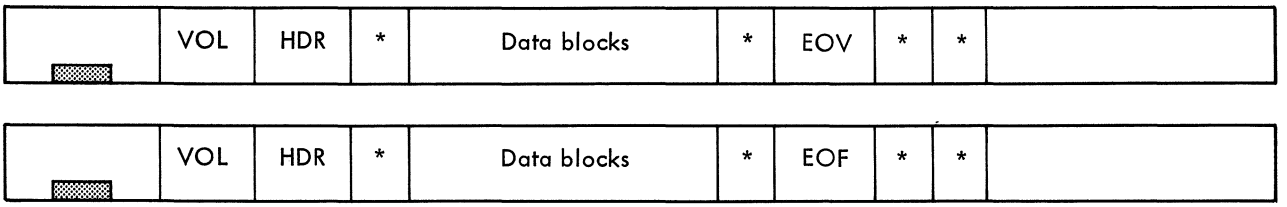
Figure 6-11. Super, Account, and Volume Catalogs

Monovolume File

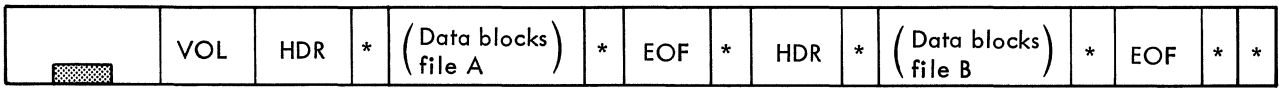


Load point

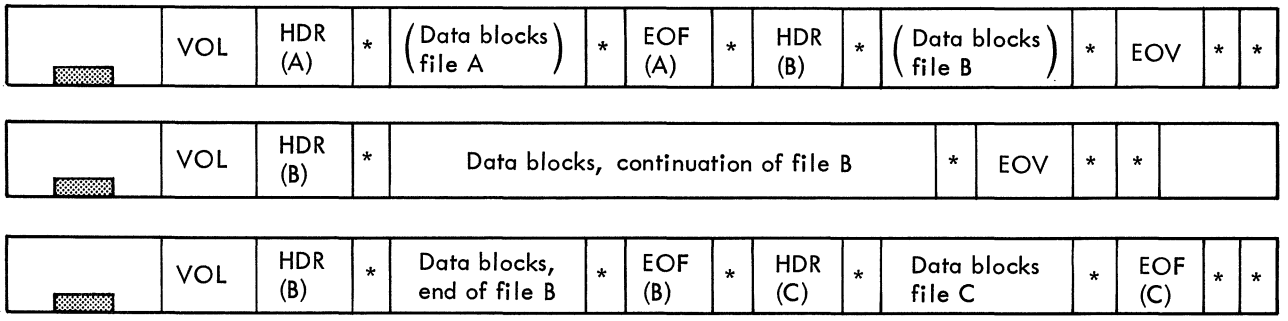
Multivolume File



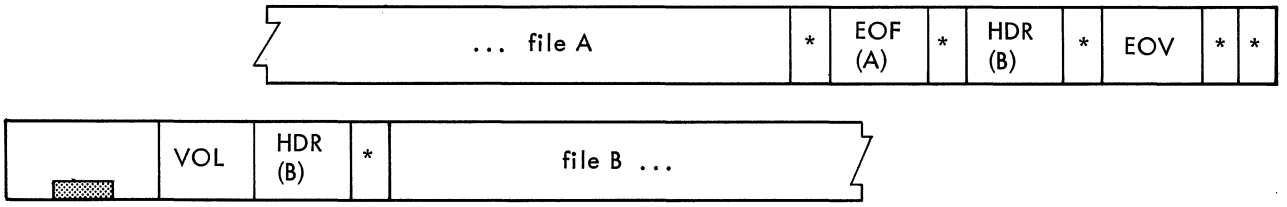
Multifile Volume



Multifile Multivolume



Note: When the end-of-volume coincides with the end-of-file, the configuration is the following:



Legend

: represents the tape-mark record

VOL : the volume header-label group

HDR : the file header-label group

EOF : the end-of-file trailer label group

EOV : the end-of-volume trailer label group

Figure 6-12. Structure of Volumes on Magnetic Tape

below. Under this mode, all volumes of a multivolume set must be mounted before processing begins.

The mounting options of the !ASSIGN command, including a deferred serial-mounting option, are decribed in Chapter 3.

PHYSICAL FILE/VOLUME STRUCTURES

The physical structures described below are those of volumes and files either created under XOS or recognized as standard by XOS (see also "Volume Classifications").

DISK PACK STRUCTURES

The structure of a disk pack volume is completely defined by its volume header, which contains:

1. The volume label, whose contents includes the volume serial number and, if a private volume, the owner's account number.

2. The volume catalog, which contains the labels of all the files on the volume. Each label describes the location of the file on the volume, as well as a description of the file characteristics.

3. An allocation table, which represents the available (i.e., unused) space remaining on the volume.

The location of these elements of the volume header are as follows:

Element	Sector Address (1024 bytes/sector)
Reserved for system	0
Volume Label	1
Volume Catalog (variable length)	3 through n
Allocation Table	n + 1

The remainder of the volume consists of allocated file space and/or available space. A detailed description of disk-volume and file labels may be found in Appendix C.

MAGNETIC TAPE STRUCTURES

A standard magnetic tape volume has a volume label group (recognizable as such by XOS); a nonstandard magnetic tape volume does not. The volume-label group contents include the volume serial number and, if a private volume, the owner's account number. Each standard file is delimited by a header label group and a trailer label group. Nonstandard files have no label groups (recognizable as such by XOS). Each label group is followed by a tape mark. For both standard and nonstandard files, the data portion of the file is followed by a tape mark. There is an additional tape mark at the logical end-of-tape. Each file label group includes file identification and protection information as well as some logical structure descriptors. A detailed description of tape volume and file labels may be found in Appendix C.

CATALOGED FILES

The XOS user, should he desire to reference an existing file residing on his account volume, need not supply the identification of the volume in his !ASSIGN command. The absence of the volume specification in the !ASSIGN command causes the system to perform a search of the supercatalog, from which it extracts the identification of the account volumes. In a similar fashion, XOS provides the user with the ability to reference files residing on volumes other than the account volume, such references being made by file name only, by providing the ability to make entries for such a file in the account catalog. The term used for a file for which an account catalog entry has been made is a "cataloged file". The function of making the entry in the catalog is termed "cataloging" the file.

An option on the !ASSIGN command, CTG, is provided to indicate to XOS that the file is to be cataloged. This option must be present at the time the file is created in order to catalog the file. Upon encountering this option, XOS enters in the user's account catalog sufficient information to locate the file on any subsequent reference. Since all files on the user's account volume are by definition cataloged files, the CTG option has no effect when creating files on the account volume.

Once a file has been created and cataloged, a reference to the file need not contain the location of the file, i.e., the volume serial number. The necessary information will be extracted from the account catalog (via the supercatalog) when reference to the file name is made. A reference to a cataloged file always produces the most recent absolute generation and version, unless a generation and version are entered as part of the file name.

In short, cataloged files free the user from keeping track of where a file is located, transferring that responsibility to the system.

ABSOLUTE FILE GENERATIONS AND VERSIONS

A set of files known by a single file name, each member of which is distinguishable one from another by an absolute generation number, is called a set of absolute file

generations. Each member thereof is referred to as a generation; a generation version number can be used to further distinguish between differenct versions of the same (absolute) generation.

For example, PAYMASTER,G0001 might identify one generation and PAYMASTER,G0005 another generation belonging to the absolute generation group known by the file name PAYMASTER. The generation number, of the form Gnnnn (n = 1 through 9) is assigned at file creation time, and specified when retrieving an existing generation, via the !ASSIGN command. If a generation number is not specified when referring to a cataloged set of file generations, the most recently cataloged generation is always retrieved. A generation number is, therefore, an extension of the filename; its default at file-creation time is 0001, and is (for cataloged files) the number of the last cataloged generation when referring to an existing file, i.e., status OLD or MOD. (Any permanent file created under XOS is implicitly a member of a generation set containing at least one member i.e., generation 0001, version 00.) Note that although file generations need not be cataloged, the file-generation facilities are intended specifically as a complement of the cataloging capability, so as to provide as much flexability as possible in the establishment and maintenance of a set of absolute generations. For uncataloged files, the user must consider the generation number (and also the version number discussed below) simply as part of the filename, and that general filename-usage rules apply to the total identification.

A two-digit version number functions as a further qualification of the file generation. All permanent files are assigned a version number of 00 by default, but a version number may be specified in conjuncation with an explicit generation number, either when creating or retrieving a file. The version number specifically allows for the replacement of a cataloged generation with another version of the same generation, i.e., another file with the same generation number but a different version number. Thus, two identically numbered files may exist in the same job step, e.g., one OLD and one NEW file, distinguished by differing version numbers. On cataloging of the new file, however, it replaces (rather than coexists with) the already-cataloged generation, since only one version of a generation can exist, i.e., be known, in an account catalog at a time – except during the cataloging process itself. (Note that deletion of a version entry from the catalog does not cause deletion of the actual file if it exists on a private volume; the user must perform the actual file deletion in this case.)

RELATIVE FILE GENERATIONS

Relative file generations, established through the use of the DEFG utility processor, allow the user to specify that XOS is to perform certain of the functions of job setup and volume retention normally done manually. If so specified, XOS will "keep track" of the volume serial numbers associated with specific files, and optionally, XOS will decide which of the volumes retained as backup to a current file may be reused. This capability of XOS frees the user of the responsibility of supplying the volume serial numbers associated with specific files (normally supplied in the !ASSIGN command), and of the responsibility of deciding which volumes are available for reuse.

For instance, a typical daily job might require one input tape and one output tape, the output tape to be used as the input tape for the next day's run. Further, the job requires that all tapes used for the job must be saved five days. The user may specify to XOS, through DEFG, the set of six magnetic tape volumes to be used for the job by their volume serial numbers, and XOS will automatically request the correct input and output tape each time the job is run. See Figure 6-13, for an illustration of this example.

In the example shown in Figure 6-13, all six files are referred to by the same file name, DAILY. Such a set of files, once specified to the system via DEFG, is called a relative generation group. The name of the generation group is the name of the file managed by the group, in this example, DAILY. Within a generation group, the user may refer to a specific file, or generation, by its relative chronological position within the group, i.e., -n or +n. See Figure 6-13 and Table 6-3. After the run for day 3, the current file is on volume TAPE04 and may be referred to by relative generation number 0. The other files in the generation group may be referred to at that time by their position in the group relative to generation 0. Thus, the last, or oldest, tape in the generation group is referred to as relative generation -5 – or relative generation +1 when viewed as the next tape to be reused.

After the run for day 4, TAPE03 becomes relative generation 0, TAPE04 relative generation number - 1, and TAPE02 relative generation -5. Figure 6-14 shows the !ASSIGN commands required for execution of a program EVERYDAY, which uses the current and oldest files in the generation group DAILY, and the program WEEKEND which requires reference to all tapes in the generation group DAILY.

In the preceding discussion and examples, the closed-loop type of generation-file management is employed. The closed-loop generation group always assumes that relative generation +1 refers to the oldest generation in the group. Opposed to the closed-loop type is the open-loop type of generation group, which functions as a "push-down" table. When a new generation is established, the user must specify the relative generation number +1 and the volume serial number, device type, etc., of the new generation. At job conclusion, the new generation becomes relative generation 0, relative generation 0 becomes - 1, etc., and the previously oldest generation is dropped from the generation group. Refer to Figure 6-15, in which an open-loop is employed for the program EVERYDAY.

In the preceding discussions and examples, the volumes used to contain the file DAILY have been magnetic tape. Generation files may also be used in a similar manner to refer to RAD or disk storage. The user need only specify the media type to DEFG at the time the group is created (closed-loop) or in his !ASSIGN command (open-loop).

DEFG also provides a means of maintaining and listing the generation group, should such operations become necessary.

In summary, generation files

- Provide the user with the ability to automatically reuse file space on a cyclical basis, whether tape, disk, or RAD
- Maintain backup copies of files on a cyclical basis
- Reduce the control command coding required for repetitive jobs (hence eliminating errors)

The user specifies to XOS, through DEFG, the following:

1. File name: DAILY
2. Number of volumes required: 6
3. The volume serial numbers:
   a. TAPE01
   b. TAPE02
   c. TAPE03
   d. TAPE04
   e. TAPE05
   f. TAPE06
4. That the oldest volume is available for output, and is to be used as such when the user requests the next volume.
5. That the current tape volume serial number is TAPE01, the oldest TAPE06.

Schematically, XOS manages the volumes as follows:

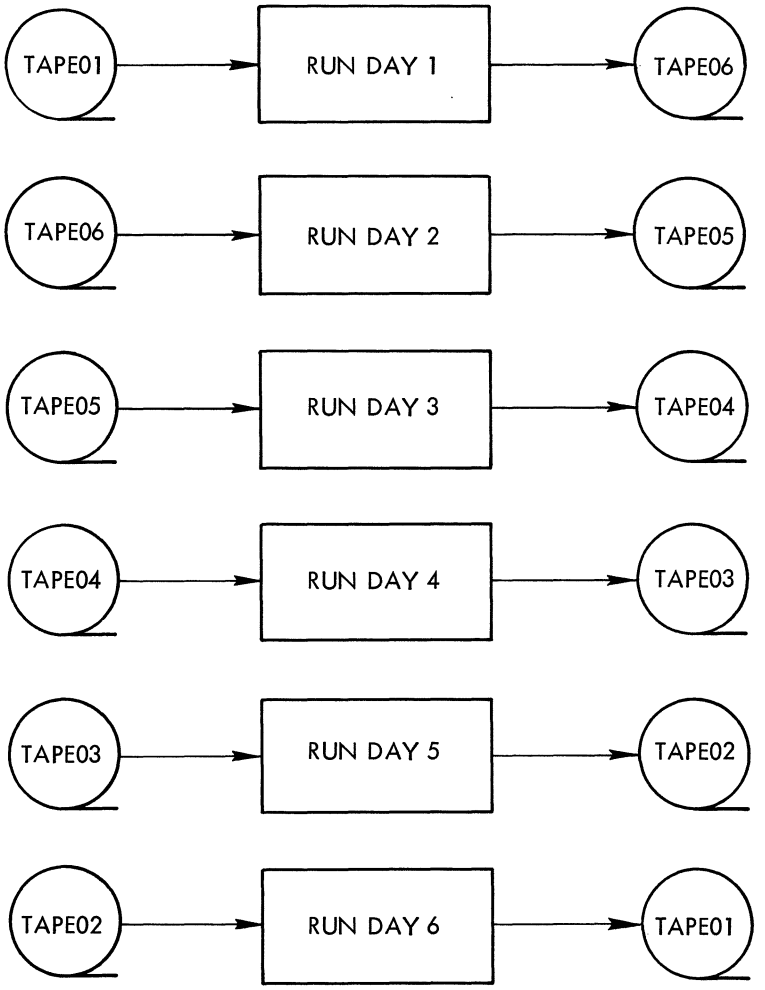


Figure 6-13. Relative File Generation Group

Table 6-3. Relative Generation Numbers of File DAILY

Generation group DAILY after run day 3	
Volume Serial Number	Relative Generation Number
TAPE04	0
TAPE05	-1 or +5
TAPE06	-2 or +4
TAPE01	-3 or +3
TAPE02	-4 or +2
TAPE03	-5 or +1
Generation group DAILY after run day 4	
Volume Serial Number	Relative Generation Number
TAPE03	0
TAPE04	-1 or +5
TAPE05	-2 or +4
TAPE06	-3 or +3
TAPE01	-4 or +2
TAPE02	-5 or +1

```
!JOB  EVERYDAY
!ASSIGN  INPT,FIL,(NAM,DAILY,0),(STS,OLD)      Input File
!ASSIGN  OPUT,FIL,(NAM,DAILY,+1),(STS,NEW)     Output File
!RUN  EVERYDAY
```

The above job uses the current file (relative generation 0) as input, and the oldest file (relative generation +1) as output. Both files are members of the generation group DAILY. At job conclusion, the file referred to as a relative generation +1 becomes relative generation 0, relative generation 0 becomes relative generation -1, etc.

```
!JOB  WEEKEND
!ASSIGN  INP1,FIL,(NAM,DAILY,0),(STS,OLD)      Input File 1
!ASSIGN  INP2,FIL,(NAM,DAILY,-1),(STS,OLD)     Input File 2
!ASSIGN  INP3,FIL,(NAM,DAILY,-2),(STS,OLD)     Input File 3
!ASSIGN  INP4,FIL,(NAM,DAILY,-3),(STS,OLD)     Input File 4
!ASSIGN  INP5,FIL,(NAM,DAILY,-4),(STS,OLD)     Input File 5
!ASSIGN  OPUT,FIL,(NAM,DAILY,+1),(STS,NEW)     Output File
!RUN  WEEKEND
```

The above job uses the five most current files (relative generations 0, -1, -2, -3, and -4) as input, and the oldest file (relative generation +1) as output. All files are members of the generation group DAILY. At job conclusion, the file referred to above as relative generation +1 becomes relative generation 0, relative generation 0 becomes -1, etc.

Figure 6-14. Relative-Generation-File Command Sets

```
!JOB  EVERYDAY

!ASSIGN  INPT,FIL,(NAM,DAILY,0),(STS,OLD)

!ASSIGN  OPUT,FIL,(NAM,DAILY,+1),(STS,NEW),(UNT,MT,(VOL,TAPE07))

!RUN  EVERYDAY
```

Relative Generation Group DAILY Before Above Job is Run	
Volume Serial Number	Relative Generation Number
TAPE01	0
TAPE02	-1
TAPE03	-2
TAPE04	-3
TAPE05	-4
TAPE06	-5
Relative Generation Group DAILY After Above Job is Run	
Volume Serial Number	Relative Generation Number
TAPE07	0
TAPE01	-1
TAPE02	-2
TAPE03	-3
TAPE04	-4
TAPE05	-5

Figure 6-15. Open-Loop Relative Generation Group

!ASSIGN COMMAND USAGE

The chain of program and job control elements that constitutes the relationship between a program and an actual physical file of information that it creates or accesses is (1) a Data Control Block (DCB) in the user's program, (2) a name, called an operational label, associated with that DCB, and (3) a !ASSIGN control command in the job-control deck for that program. The function of the !ASSIGN command is to define the physical medium and associate it, via the operational label, with the logical file defined by the DCB. This logical chain and the operational flexibility it affords is described in more detail in the next subsection. The several types of file assignments (i.e., definitions) are described in the subsequent subsections, as well as related topics: file identification, file-status specification, volume-mounting options, and the optional specification of DCB parameters via the !ASSIGN command.

The reference syntax of the !ASSIGN command is shown on foldout chart A1, Appendix A, and described in detail in Chapter 3.

PROGRAM/FILE RELATIONSHIP

The principal reason for the indirect chain of elements, described briefly above, that defines the program/file relationship is file and device independence: it provides not only file name independence but also, given a measure of forethought on the user's part during program design, a

large degree of program independence of I/O-device type, where this is desirable.

The data control block that is assembled in the user's program – typically one per program file – defines the file in terms of its logical characteristics: logical structure and file organization. (As noted under "File Characteristics", several combinations are compatible with more than one category or subcategory of file media.) The DCB is created with the M:DCB procedure. During program execution, the DCB may be modified, in terms of individual parameters, with the M:SETDCB procedure. Upon execution of an M:OPEN procedure that references the DCB, a connection is established between the program and the file defined by the corresponding !ASSIGN command. Also during file opening, DCB parameters specified in the !ASSIGN command may be used to complete or modify fields in the DCB prior to the reading of the file labels (existing file). If the file is an existing permanent file opened for reading, modification, extension, or rewriting, some or all of the file's characteristics (depending upon record format) as recorded in the file label are written into the DCB, overlaying any corresponding information previously filled in.

Usage of the M:DCB and M:SETDCB procedures is described in detail in Chapter 7.

The set of !ASSIGN commands associated with a job step defines for the system the combination of device resources that must be allocated to the job step before it is initiated for execution.

Several quite simple but nontrivial examples of the flexibility afforded by this mechanism are the ability at run-time to

1. Assign either of two different types of device, e.g., card reader or magnetic-tape drive, as a program input.

2. Assign a program output to a device or media other than the one intended as normal, e.g., assignment of card punch output to the line printer during program testing.

3. Assign a program output to a "dummy" file (DUM-type assignment), effectively nullifying this output when it is not required.

This kind of run-time variability without program modification can be of paramount value under both program-testing and production conditions. In general, the degree of flexibility that can be achieved is limited by the user's choice of, and nonrestrictive or generalized usage of, an appropriate access method.

ASSIGNMENT TYPES

The terms in which a physical file is defined in a !ASSIGN command differ according to the type of file involved.

Thus there are two basic types of assignments: the DEV or device type, and FIL or file type. (The OPL type assignment is effectively an indirect form of either DEV or FIL; the DUM type is a "null" assignment.) Within each basic type, several subtypes corresponding to different media catagories can be distinguished by characteristic syntax differences. The various forms of !ASSIGN usage are described in the following subsections.

DEV-TYPE ASSIGNMENTS

The DEV, or device, type of assignment is used for assigning a nonmagnetic device file, or for assigning a file on a nonstandard magnetic storage volume, e.g., a nonlabeled magnetic tape volume. (A standard volume may be assigned for processing as nonstandard with the device assignment also.) In either case, a common characteristic of the DEV assignment is that a file name is not specified.

NONMAGNETIC DEVICE ASSIGNMENTS

Nonmagnetic device files can be assigned via a symbiont. A symbiont is a system program that automatically creates an intermediate disk file – essentially either an input or output card-image file or an output print-image file – corresponding to the actual device input or output. The purpose of symbionts is to improve system throughput and operational flexibility; their existence and operation is completely transparent to the user's program. Symbionts are applicable to ASAM only. The names of the symbionts and the devices to which they correspond are:

IN – the job-stream card reader.

OUT – the line printer associated with the job-output file (a listing produced thereon may be interspersed with the system-produced print lines).

SLP – a line-printer output stream alternative to OUT that produces a contiguous listing (may be printed on the same physical device, but as a separate file).

SCP – the card punch.

Thus, a sample assignment to read a data file from the job-input stream, e.g., a card deck immediately following a !RUN command, would be:

```
!ASSIGN F1,DEV,IN
```

To print output on the line printer associated with the job-output file in which system-produced control and accounting information also appears:

```
!ASSIGN F2,DEV,OUT
```

To assign a contiguous print file:

```
!ASSIGN F3,DEV,SLP
```

To assign a card output file:

```
!ASSIGN F4,DEV,SCP
```

In the case of SLP and SCP, the symbionts normally operate in "catch-up" mode: they attempt to transmit the file records to the destination device – printer or punch, respectively – before the file is closed, i.e., as the symbiont file on disk is being written by the user program. The NKP option, if used, prevents the symbiont from writing to the output device until the user's program closes the file.

The sample operational labels F1, F2, etc., are of course arbitrary, one-to-four characters, programmer-chosen symbols.

The user can, alternatively, gain nonsymbiont access to a nonmagnetic device file by specifying a symbolic device code in place of a symbiont name:

CR – card reader

LP – line printer

CP – card punch

The symbiont versus nonsymbiont assignment does not imply any programming differences in the user's program. It is purely an operational consideration; the choice may be dictated by installation practice. Symbiont access to nonmagnetic device files is the more efficient mode, however. (Symbiont access is not available under VSAM.)

For direct device assignments, a device code (CR, LP, or CP) would be substituted for the symbiont name in the above examples; the NKP option does not apply in this case.

NONSTANDARD VOLUME ASSIGNMENTS

Although the DEV-type assignment is equally applicable to nonstandard magnetic tape or disk pack volumes (or such volume to be processed as if nonstandard), the most common usage is for the assignment of nonlabeled tape files. One mandatory use of the DEV-type assignment is for the assignment of disk volumes, either standard or nonstandard, for processing via the BDAM access method. In any case, a volume specified in a device-type assignment does not undergo Automatic-Volume-Recognition (AVR) processing. That is, no validity checking of volume identification or ownership is attempted.

The syntax (excluding DCB parameters) of a nonstandard volume assignment is

```
!ASSIGN  op-label[,{MTN}],DEV,media-type;
                   {FRE}
!  (VOL,serial-no[,...])[,mounting-option]
```

where

media-type is MT for 9-track magnetic tape, 7T for 7-track magnetic tape, and DM for disk pack (7242).

mounting-option is as described under "ASSIGN Command" in Chapter 3 (PAR, MNT, or DEF option).

A sample assignment for a nonlabeled file on 9-track magnetic tape would be

```
!ASSIGN F5,DEV,MT,(VOL,ABC123)
```

FIL-TYPE ASSIGNMENTS

The FIL-type assignment is used for either temporary or permanent files on magnetic storage media. For permanent files the assignment may be made to any standard volume; for temporary files, the assignment may be only to a public volume or to temporary file space on secondary storage. The syntactic distinction between a permanent and a temporary file assignment is the presence or absence, respectively, of the NAM option.

(See foldout chart A-1 of Appendix A for the complete syntax of the !ASSIGN command.

TEMPORARY FILE ASSIGNMENTS

On Secondary Storage. The form (excluding DCB parameters) of the FIL assignment for a temporary file on secondary storage space is

```
!ASSIGN op-label[,{MTN}],FIL[,(SIZ-option)]
                  {FRE}
```

Therefore, an example of the minimum required form of such an assignment would be:

```
!ASSIGN F1,FIL
```

This simple assignment reserves an installation-defined default amount of direct-access file space (minimum = 8,192 bytes), associated with the operational label F1. Since the default for the MTN/FRE (read "maintain/free") field is FRE, this assignment would remain in force only for the job step with which it is associated. (Specification of MTN

would cause the assignment to remain in force over subsequent job steps until superseded or suppressed. Thus the same file could be reopened in subsequent job steps, assuming a prior M:CLOSE with MTN specified.) Use of the SIZ option is described in a separate section, "Space Allocation", below.

On a Public Volume. The form (excluding DCB parameters) of the FIL assignment for a temporary file on a public volume is

!ASSIGN op-label$_1$ [,{MTN / FRE}],FIL,{(UNT,type) / (DEF,op-label$_2$)};

! [,(SIZ-option)]

where

type may be MT for 9-track magnetic tape, or DM for disk pack.

op-label$_2$ is the label of another !ASSIGN command in force during the same job step.

(See "Space Allocation" for SIZ-option usage, which optionally applies only in the case of a disk volume.)

The minimum forms reduce, therefore, to

1. !ASSIGN op-label,FIL,(UNT,{MT / DM})

2. !ASSIGN op-label$_1$,FIL,(DEF,op-label$_2$)

Form 1 causes the reservation of a system-selected public tape or disk volume and corresponding devices for the job step, associated with the specified operational label. In the case of a disk volume (type = DM), the default amount of file space is allocated.

Form 2 causes a deferred association of the resource assigned to op-label$_2$ by another !ASSIGN command, if that resource is available (i.e., free) at the time the DCB associated with op-label$_2$ is opened. The !ASSIGN command referenced by op-label$_2$ must be in force during the same job step, and must define the resource directly, as in form 1.

An example of form 1 would be

!ASSIGN F2,FIL,(UNT,MT)

which allocates some public tape volume to the job step, associated with label F2.

An example of form 2 would be

!ASSIGN F3,FIL,(DEF,F2)

which will cause the same tape volume and drive unit that was reserved by the first example assignment to be associated with label F3, if that volume is available when the DCB defining logical file F3 is opened – i.e., if no other DCB is open to that volume at the time. Several such deferred assignments, referring to the same resource definition (e.g., F2), may be made within one job step. The referenced assignment may have been maintained for a previous job step.

Note that the status option (STS,NEW/OLD/MOD) is not relevant, insofar as only its default, NEW, is applicable to a temporary file; if reopened for input subsequent to file creation, its status automatically becomes MOD. The reopening must, of course, be within the same job and the identifying op-label must have been maintained continuously. See M:OPEN, in Chapter 7, for information on automatic evolution of a file to MOD status.

Also, the disposition (DSP), protection (PRT), retention (RET), linking (LNK), and PAR/MNT multivolume mounting options are not relevant to temporary-file assignments. Public disk volumes are not sharable, i.e., only one DCB at a time can be open to a given public volume. (Magnetic tape volumes are inherently unsharable.)

PERMANENT FILE ASSIGNMENTS

Permanent file assignments are characterized syntactically by the appearance of the NAM option, since all permanent files are labeled files. A syntactic distinction exists between the assignment for a permanent file on an account volume and that for a permanent file on a private or public volume. The VOL suboption (of UNT option) does not appear in the former case, and does appear in the latter.

Common FIL-Assignment Options with Defaults. The following options have default values and therefore need not appear in any case for which the default is acceptable. (See Chapter 3, "ASSIGN Command", for full descriptions.)

({PAR / MNT,n / DEF,logical-label}) This option gives a choice of mounting modes: parallel, serial, or deferred, in the case of multivolume files; the default for magnetic tape is MNT: serial volume mounting on one drive unit.

(DSP,{RET / KEP / DMT}), This option allows control over the dismounting of volumes and retention of allocated resources; the default is DMT: dismount volume(s) and release device(s).

(SIZ [,({RST / SEP})] ,size[,increment]) This option allows control of the amount of the file space to be allocated for a new direct-access file, plus special-case suboptions; the default space allocation is installation-determined.

See "Space Allocation" for usage description. (Usage varies by file organization.)

(PRT, suboptions) This option allows control, during file creation, of file protection facilities (access control, password) for the new file; the default conditions are that all other accounts may subsequently read the file, no other account can modify or rewrite it, and no password is attached. See "Volume and File Sharability" for usage description.

(RET, period) This option allows control, during file creation, of the retention period (i.e., nominal useful lifetime) to be assigned to the new file. The default value is installation determined.

The above options will not be shown in the two basic formats given below for permanent file assignments, although they may be added to the essential elements shown, as appropriate, at the user's discretion. (Note: the mounting option PAR is mandatory in the special case of a multivolume disk file assigned for nonsequential processing and is the default for disk media).

Permanent File on Account Volume. The basic form of an assignment of a file on an account volume, excluding common options and DCB parameters, is as follows:

!ASSIGN op-label [, {MTN / FRE}], FIL, (NAM, filename);

! [, (UNT, AC, acctno)], [(STS, {NEW / OLD / MOD})]

where acctno is a four-character account number.

The UNT option need only be specified in the case of referencing an existing file on another user's account volume. The default value is the submitting user's own account number, identifying his account volume. The system automatically locates the account volume by account number, implicit or expressed. The STS (status) option need be specified only in the case of an existing file (OLD or MOD); the default status is NEW. Note that when either UNT or STS is omitted the other may be omitted also, since a file may be created on an account (or private) volume only by the owner of the volume.

A sample minimum-form assignment of a new file, i.e., one to be created, on the user's own account volume is as follows:

```
!ASSIGN F4,FIL,(NAM,TESTFILE1)
```

To reference the above-assigned file, subsequent to its creation, only the status option need be added:

```
!ASSIGN F5,FIL,(NAM,TESTFILE1),(STS,OLD)
```

The status MOD would be used instead of OLD if the file was being referenced for updating rather than reading (see the M:OPEN procedure description, in Chapter 7, for status/processing-mode relationships). However, if the new-file assignment were made in one step of a job and the assignment was maintained (MTN specified) over subsequent steps, the status of the file would automatically evolve to MOD on closing of the file in the file creation step. A subsequent job step could then open the file in input or update mode using the same operational label, thereby making use of the original !ASSIGN command as automatically modified (effectively) by the system.

To reference a file on another user's account volume, the other user's account number, e.g., A107, would be specified as follows:

```
!ASSIGN F6,FIL,(UNT,AC,A107),(NAM,PROG5);
!     ,(STATUS,OLD)
```

Note that the parenthesized FIL options may be given in any order following the positional keyword FIL, as is true of the options following DEV and DCB also.

Permanent File on Private or Public Volume. For file creation, the volume serial number is always specified in the case of a private volume; it may be specified in the case of a public volume, or the system may be allowed to select the specific public volume. (The public volume automatically becomes private to the file creator's account.)

In a reference to an existing file (necessarily on a private volume), the volume serial number is always specified. The basic form of the assignment, again excluding common options and DCB parameter, for either case is

!ASSIGN op-label [, {MTN / FRE}], FIL, (NAM, filename);

! , (UNT, type [, (VOL, serial-no, ...)]) [, (STS, status)]

where type is MT for (9-track) magnetic tape, DM for 7242 disk pack. The media-type specification is the minimal requirement for the UNT option (file creation on unspecified public volume). The status option need not be specified for file creation; default in NEW.

An example of a minimum-form assignment for file creation on a private disk volume follows:

```
!ASSIGN F7,FIL,(NAM,XYZFIL),(UNT,DM(VOL;
!     ,117354))
```

An assignment for creation on a public volume might either be the same as above, or with the VOL suboption omitted from the UNT option. To use the above sample assignment

for file reference, only the STS option – with OLD or MOD specified – need be added.

SPACE ALLOCATION

Space is allocated for the creation of a new disk or RAD file according to the specified or default values of the SIZ parameter of the !ASSIGN control command (or of the M:ASSIGN procedure, if used). The syntax of the SIZ parameter is described in Chapter 3.

The meaning and effect of the SIZ parameter values vary according to the organization of the file to be created. They are described for each organization in the following subsections.

Note that no new space allocation can be made for a disk/RAD file that is to be rewritten, i.e., a file replacing an existing identically-named one will occupy the same space allocated to the original file. (Status OLD; Output processing mode.)

CONSECUTIVE ORGANIZATION

Value1 of the SIZ parameter specifies, in quanta of 8K bytes, the initial amount of space to be allocated to the new file. Value2 specifies the size of the increments to be added to the file in case of either overflow of the initial allocation during creation, or extension of the file during subsequent status-MOD, Output-mode processing.

If all of the space specified by value1 is not available on the first of a series of volumes specified, the remainder will be allocated on succeeding volume(s).

INDEXED-SEQUENTIAL ORGANIZATION

For an indexed-sequential file:

value1 specifies, in quanta, the space to be allocated for base data blocks.

value2 specifies in quanta, the space to be allocated for (1) the creation of index blocks, and (2) overflow blocks.

Note that unlike C and P organization files, the entire and final amount of file space – including possible overflow space – is allocated at file creation time; no subsequent extensions are allowed.

If value2 is omitted or given a zero value, the system reserves index-block space as if all of the file were to be allocated for base data blocks (less index space), and does not allow any space for overflow.

METHOD OF CALCULATING THE NUMBER OF INDEX BLOCKS REQUIRED

Since the indexed-sequential organization is relatively complex, a method is presented for the calculation of the number of index blocks that will be needed (assuming that the substantive file grows to full size), given a specified amount of space for base data blocks (value1 – value2 X 8,192 bytes).

Preliminary Definitions. BKL defines the block length of base data blocks, overflow data blocks, and index blocks, in bytes. Since blocks of an indexed-sequential file, regardless of record format, contain a 4-byte block header and a 4-byte linkage field, the usable block size, b, is defined as follows:

$$b = BKL - 8$$

KYL defines the record key length, in bytes.

Both BKL and KYL are specified in the DCB corresponding to the file to be created.

Values to be computed:

N_o, the number of base data blocks.

E_x, the number of index entries per index block.

N_1, the number of first-level index blocks.

.
.
.

N_i, the number of i^{th} level index blocks.

Equations. Assuming that (value 1-value 2) > 0, the value N_o is derived as follows:

$$N_o = \text{integer portion of } \left[\frac{(\text{value 1-value 2})}{BKL}\right] \times 8192$$

Any index, entry is composed of a record key plus a 3-byte pointer to a late block. Therefore, the index-entry length, l, is given by

$$l = KYL + 3$$

and the number of entries per index block, E_x, by

$$E_x = \text{integer } \left[\frac{b}{l}\right]$$

The number of n^{th} level index blocks, N_n, is given by the number of base data blocks divided by the number of

entries in an index block, the result being rounded upwards to an integer, i.e.,

$$N_1 = \text{integer}\left[\frac{N_o}{E_x}\right]$$

Similarly

$$N_2 = \text{integer}\left[\frac{N_1}{E_x}\right]$$

.
.
.

$$N_i = \text{integer}\left[\frac{N_{i-1}}{E_x}\right]$$

Therefore, the total number of index blocks, N, that will be created is

$$N = \sum_i N_i$$

and the total amount of space that will be reserved therefore, is in bytes

N x BLK

The excess of value2 x 8,192 over the amount of space derived above will be available for overflow blocks. By making a preliminary estimate of value 1 and value 2, based on the prediction size of the data portion of the file, and then performing the calculations described above, the values of value 1 and value 2 may be adjusted to produce the most efficient allocation.

Caution: Since blocks on direct-access media always being on a sector boundary and take up as many full sectors as are required to accommodate the block length, the user must carefully relate BKL and the sector size of the device involved in order not to waste direct-access media space. Specifically, if BKL is not equal to of a multiple of sector size, the equation given above for deriving N_o, and thus the entire calculation, is invalidated.

PARTITIONED ORGANIZATION

Value1 of the SIZ parameter specifies, in quanta, the initial amount of space to be allocated to the new file. Value2 specifies the size of the increments to be added to the file in case of either overflow during creation, or extension during subsequent status-MOD, Output-mode processing.

To arrive at appropriate SIZ parameter values for a partitioned file the user should note that

1. The directory entries and the data records are kept in separate blocks and that in both cases the effective block length is BKL-8.

2. To compute the number of partition key entries a block can contain, E_d, one must consider that the length of each entry is, in bytes

   KYL + 5

   Thus

$$E_d = \text{integer}\left[\frac{b}{KYL + 5}\right]$$

The probable maximum number of partition keys, both principal and synonym, to be stowed is therefore a factor in estimating the best allocation.

DIRECT ORGANIZATION

Value1 is the total amount of space, in quanta, to be allocated to the direct-organization file. Value2 is not significant for this organization; i.e., a direct-organization file is not extendable beyond its creation-time allocation.

RULE FOR ALLOCATION OF MULTIVOLUME DIRECT-ACCESS FILES

For all files necessitating parallel mounting, i.e., multivolume direct-access files (I, P, or D organization), the amount of space specified by value1 of the SIZ parameter must be available exclusively on the volumes specified for mounting or the allocation request will be refused.

PASSWORD PROTECTION

When creating a file that is to be password protected, or when accessing a file that has been password protected, an X1-class abnormal return occurs at OPEN time. The abnormal return routine must detect abnormal code X'19' and must then load registers 6 and 7 with a value representing the password before executing an M:RETURN. If the file is being created, the password is entered into the HDR3 label. If an existing file is not being accessed, then the value is compared with the file's password in the HDR3 label; and if the values are identical, processing continues normally. When the passwords do not match, the job step is aborted.

At file creation time, the user informs the system that a password is to be applied to the file by means of the PAS option of the PRT (protection) field of the !ASSIGN command.

In either case (file creation or reference), the user must specify the X1 abnormal class code in the ABN parameter of the DCB in order to receive the abnormal return.

VOLUME/FILE SHARABILITY AND ACCESS AUTHORIZATION

XOS includes a utility program called PREP which is used to prepare a removable storage volume (tape, disk pack) for utilization in the system as a standard volume. In essence, PREP writes a volume label (VOL1) and other system control information as necessary on the volume.

For a disk pack[t] one of the fields of VOL1 is the "sharability" field which contains a one-byte code representing the sharability or nonsharability of the volume in its entirety.

VOLUME SHARABILITY

When a volume is sharable, more than one file on the volume can be accessed by one or more user jobs (under one or several accounts) concurrently. When a volume is not sharable, only one user job at a time is permitted to access that volume. The single user can access only one file on that volume at a time, i.e., only one DCB is permitted to be open to any file on the volume. The owner of a file to be created on either a sharable or nonsharable volume can choose to employ both password protection and file access authorization by account number.

[t] A tape volume is inherently nonsharable.

FILE SHARABILITY

When a volume is sharable, the files residing thereupon are potentially sharable. More than one user (or job) is permitted by the system to access that volume concurrently and file-sharing operates essentially as follows:

1. More than one DCB can be open to the same file for concurrent input-mode processing (including more than one DCB in a given user program).

2. The system regulates multiple access to a single file whenever one or more (authorized) users wish to modify it (i.e., for record update or file extension). User requests for opening in differing modes are queued in order of receipt by the system.

3. Account authorization and password protection control apply.

FILE ACCESS AUTHORIZATION

Authorization for read and/or write access to a given file can be granted or denied to other users (i.e., other accounts) by the owner of the file at file creation time (only). This is done by means of the PRT (protection) field of the !ASSIGN command. Read and write authorization can be separately granted (or denied) on the basis of (1) specified accounts, (2) "all" accounts, or (3) "no" accounts. The default authorizations, when the PRT field of !ASSIGN is omitted, are that all other accounts can read the file and no other accounts can write in, i.e., modify or extend the file. The special option NCT (no control) implies both read and write authorization for all other accounts and precludes password protection (the NCT and PAS options are mutually exclusive). Table 6-4 tabulates the effects of the possible combinations of PRT options.

Table 6-4. File Access Authorization: Effect of PRT-Option Combinations

PRT Options	File Accessibility Code in HDR1		Disk File Accessibility		
	No Password Protection (without PAS)	With Password Protection (with PAS)	INPUT, Status OLD	OUTPUT, Status NEW or OLD	OUTPUT, Status MOD or UPDATE, Any Status
NCT	Space (character space)		any account	volume owner only	any account
R,ALL W,ALL	A	J	any account	volume owner only	any account
R,ALL W,NO	B	K	any account	volume owner only	volume owner only
R,NO W,ALL	C	L	volume owner only	volume owner only	any account

Table 6-4. File Access Authorization: Effect of PRT-Option Combinations (cont.)

PRT Options	File Accessibility Code in HDR1		Disk File Accessibility		
	No Password Protection (without PAS)	With Password Protection (with PAS)	INPUT, Status OLD	OUTPUT, Status NEW or OLD	OUTPUT, Status MOD or UPDATE, Any Status
R,NO W,NO	D	M	volume owner only	volume owner only	volume owner only
R,accnt,... W,ALL	E	N	volume owner and authorized accounts only	volume owner only	any account
R,accnt,... W,NO	F	O	volume owner and authorized accounts only	volume owner only	volume owner only
R,accnt,... W,accnt,...	G	P	volume owner and authorized accounts only	volume owner only	volume owner and authorized accounts only
R,NO W,accnt,...	H	Q	volume owner only	volume owner only	volume owner and authorized accounts only
R,ALL W,accnt,...	I	R	any account	volume owner only	volume owner and authorized accounts only
	0 (zero)		any account	volume owner only	any account

Note: The password, if any, and/or authorized account numbers, if any, appear in the file's HDR3 label.

7. I/O PROCESSING FACILITIES

INTRODUCTION

This chapter describes in detail all of the DCB-related and I/O-processing procedures that constitute the six access methods available under XOS. For clarity, each procedure that is applicable (in a specific form) to a given access method is described separately in that form under that method, even though many of the procedures (in generalized form) are applicable to more than one of the methods.

The following subsections briefly describe the initialization and termination steps required for I/O processing of a file (common to all access methods); the general characteristics of the different access methods; and the general handling of abnormal, error, and abort conditions that may occur during file processing. The subsequent section contains a generalized description of buffer usage, preparatory to the descriptions of the specific access methods.

INITIALIZATION AND TERMINATION OF I/O PROCESSING

Prior to the execution of any of the I/O processing procedures, M:GET or M:PUT for example, certain initial conditions must be satisfied:

- A data control block (DCB) must have been created, describing the logical file to be created/processed, the access method to be used, and the general conditions that are to be in effect during processing. The procedures that explicitly achieve the creation and (minimum) completion of the DCB are M:DCB or M:MOVEDCB, and M:SETDCB. The control commands and procedures that optionally or implicitly effect the completion of the DCB are the !ASSIGN command and the M:OPEN procedure (in the case of existing labeled files).

- The physical file space, volume, or device must have been identified and the corresponding resources must have been allocated to the job by the system. This is achieved explicitly by use of the !ASSIGN command and/or M:ASSIGN procedure (the latter having a limited role), or implicitly by use of predefined operational labels (SI, LO, LM, etc.). The !ASSIGN syntax and usage is described in Chapters 3 and 6.

- A linkage between the logical file (the DCB) and the physical file (represented by the information given in the !ASSIGN command) must be established. This is effected by means of the M:OPEN procedure, whereby the user activates a specified DCB and, implicitly, the correspondingly physical file. At this point the user also specifies the processing mode desired: input, output, update, etc., and sets certain other conditions. The M:OPEN procedure performs many functions (some invariably and some conditionally) in activating the logical/physical file for processing. Among the functions performed are the setting up of communications between the DCB and the access method selected, consistency checking of the DCB parameters and the completion of these with default values if necessary. Also, in most cases of processing an existing file, record/block structure parameters and certain processing-condition parameters are read from the file's HDR2 header label, these values being substituted for those (if any) in the DCB. For the assisted access methods, I/O buffer space is also acquired if necessary.

- At the conclusion of I/O processing, the user may deactivate the DCB, and possibly the corresponding physical file, with the M:CLOSE procedure. This closure may be definite or temporary. A definite closing of the DCB severs the DCB/file linkage, freeing the DCB for further use and optionally freeing the physical file resource(s) also. A temporary closing of the DCB suspends but does not sever the DCB link, and allows a repositioning (within the file) to either end of a sequential file on magnetic media, and/or permits a change of processing mode on subsequent reopening. (For example, an intermediate work file often will be temporarily closed after output processing, repositioned to the beginning, and reopened in input mode.)

  If the user does not definitely close a given DCB prior to the end of job-step, the system automatically closes it with certain default assumptions.

CREATING AND MODIFYING THE DCB

A DCB is a table of system defined size and format that exists functionally as a DCB only when located in a read-only (or read-and-execute) area of the user's virtual memory space. The system provides a standard set of facilities for assembly-time and/or execution-time creation of DCBs, for the "saving" of DCBs in an overlay structured program, and for direct or indirect modification of DCBs at various points during program execution. The set of DCB related facilities is as follows:

- M:DCB Procedure. Allows assembly-time creation of either a partially specified or complete DCB.

- M:MOVEDCB Procedure. Allows dynamic creation of a DCB, essentially by execution-time replication, and allows the "saving" of an inactive DCB that would otherwise be overlaid.

- M:SETDCB Procedure. Allows execution-time modification or completion of a DCB prior to opening, and modification of error- and abnormal-return addresses subsequent to opening.

- !ASSIGN Command. Allows run-time specification of certain DCB parameters, which modify the DCB at open-time (see below).

- M:OPEN Procedure. Effects (as one of many functions) both explicit DCB modification as specified by the !ASSIGN command and/or implicit modification by information contained in the label of a file opened for input.

SUMMARY OF THE DCB CREATION, MODIFICATION, AND COMPLETION SEQUENCE

Assembly-Time Creation. Using the M:DCB procedure the user can cause a DCB to be created in a read-only area of his assembled program. (The M:DCB procedure is not executable.) Within the procedure, the user can specify values for some, all, or none of the set of DCB parameters that will eventually be required. System default values are applied to a limited set of basic parameters: an EBCDIC-encoded, fixed-format sequential file, to be processed by ASAM (move mode), is assumed. (No record or block size is assumed at this point.)

Execution-Time Creation. Using the M:MOVEDCB procedure the user can copy (duplicate) either a DCB "model" somewhere in his nonprotected program area or an actual unopen DCB (created via M:DCB). In either case the DCB is moved to a system selected location in common dynamic storage, with read-only protection type.

Execution-Time Modification Prior to Opening. The M:SETDCB procedure can be used to complete or modify a DCB created with either M:DCB or M:MOVEDCB. This procedure allows the user to specify any or all of the DCB parameters, as in M:DCB. If the referenced DCB is not open, all specified parameters will be written into the DCB overriding any previously assigned value. (No parameter coherence checking is done during M:SETDCB execution.)

APPLICABILITY AND CROSS-RELATIONSHIPS OF THE ACCESS METHODS

Each access method, excepting BDAM (which matches no file organization), describes in its name the type of file organization to which it primarily applies, i.e., C, I, P, or D. Beyond this primary file creation correspondence, the ASAM, VSAM, and VDAM access methods are applicable in other than output processing mode to other file organizations. Table 7-1 summarizes both the primary and cross-related applicability of all of the methods by type of processing (again excepting BDAM).

ABNORMAL, ERROR, AND ABORT CONDITIONS

A large number of circumstances may occur during file opening, I/O processing, and file closing which either interfere with, partially inhibit, or unconditionally preclude the normal continuance and completion of operations on a file. These circumstances are classified as abnormal conditions, error conditions, or abort conditions, respectively. An example of an abnormal condition would be the encountering of end-of-file, or of end-of-volume (under a basic access method), or the attempt to access a nonexistent record by key value. Such conditions are generally of a contingent nature. Error conditions are associated either with a hardware error or a serious but possibly recoverable software or programming error. Abort conditions are associated in general with irrecoverable job-initialization and programming errors.

At the user's option, abnormal and error conditions can cause a return to be made from the I/O procedure encountering or detecting the condition to a special abnormal or error routine in the user's program. This routine can determine what specific condition has occurred, can perform any contingency processing that may have been provided for the condition, and in most cases can decide whether or not to continue with further processing of the file.

The option of processing abnormal and error conditions is exercised through the ABN and ERR parameters of the DCB. If the user chooses not to provide either or both such special routines, the occurrence of any of the conditions not provided for automatically becomes a program abort condition, halting execution of the job-step. The last section of this chapter deals with abnormal and error processing in detail. Appendix B lists the program abort conditions.

BUFFER USAGE

The I/O operations requested via the access methods all involve physical records (or blocks) that are transmitted from or to buffers located in user memory. Two types of buffers must be distinguished: I/O buffers, managed by the system, for the assisted methods; and user buffers, managed by the user, for the basic methods.

ASSISTED ACCESS METHODS

Under assisted methods, three means of reserving I/O buffers are available to the user. He can choose between

1. Specifying at assembly time the symbolic address of one or a series of buffers as a parameter of the M:DCB procedure.

2. Dynamically loading at execution time the address of one or a series of buffers in the DCB using the M:SETDCB procedure.

Table 7-1. Applicability and Cross-Relationships of the Access Methods

Type of Processing File Organization	Creation NEW or OLD, First M:OPEN, O	Extension MOD, explicit or implicit on a new M:OPEN, O	Reading OLD or MOD, M:OPEN, I[t]	Updating MOD, M:OPEN, U	Undefined NEW or MOD, M:OPEN, S	Backward Read OLD or MOD, M:OPEN, B
C (Record structure)	ASAM VSAM[tt]	Same access method and organization as at Creation	ASAM VSAM VDAM	ASAM (F or V format)		ASAM (F or U format)
C[ttt] (No record structure)	ASAM (U format) VSAM		ASAM (U format) VSAM VDAM		VSAM	ASAM (U format) VSAM
I	AIAM		AIAM ASAM VSAM VDAM	AIAM ASAM		
P	APAM		APAM ASAM VSAM VDAM	APAM ASAM		
D	VDAM		VDAM VSAM ASAM (U format)		VDAM	

[t] In VSAM or VDAM only the block length parameter is taken from the HDR2 label. If record structure information exists, it is not used.

[tt] The DCB must include an F or V format indication and a nonzero record length; the user must create his logical record structure.

[ttt] A record structure is indicated by a nonzero record length and an F or V format indication in the DCB at creation, and in the HDR2 label afterwards.

3. Letting the system perform the buffer reservation dynamically when the DCB is opened. When no buffer address exists in the DCB at open time, the system reserves a number of buffers equal to the value specified by the NBF parameter of the DCB, each buffer equal in length to the block length (BKL parameter of the DCB). The release of such buffers is automatic and occurs at the definite close of the DCB.

USE OF BUFFERS BY THE ASSISTED ACCESS METHODS

The record blocking-deblocking functions applicable to assisted access methods are effected in one of the two following modes:

1. Move-Record (MOV) mode. The system transfers the logical record between a work area in the user program (whose address is specified in the I/O procedure) and the I/O buffer.

2. Locate-Record (LOC) mode. The system returns the address of the current record location within the I/O buffer to the user in a word of his program (whose address is specified in the I/O procedure). In the case of M:PUT, the user must subsequently construct the record, origined to the address so returned.

In both these modes, for variable (V) record format it is the body of the record (record header excluded) that is transmitted to or from the user program work area. The record length passed between the user and the system for all formats is the length of the body of the record. In F and U formats the record and the body of the record are equivalent.

UNASSISTED ACCESS METHODS

Transfers between memory and devices are executed directly to or from areas (user buffers) in the user program. The

address of such a buffer is specified directly in the I/O procedure. There is no automatic buffering as for the assisted access methods.

SPECIAL SYNTAX CONVENTIONS

The following syntax conventions are employed in the procedure syntax description in this chapter:

- Byte Addresses. A syntax symbol of the form

  badr . . .

  indicates a value taken as a byte address.

- Word Addresses. A syntax symbol of the form

  adr . . .

  indicates a value taken as a word address.

- Nonaddress Values. A syntax symbol of the form

  value . . .

  indicates a value of the form described in the text for the given symbol.

- Optional Indirectness. Optional indirectness is indicated by a bracketed asterisk, [*]. Use of the asterisk prefix always implies that the argument actually specified is a word address of a word containing a value of the form indicated in the syntax, whether that value is a byte address, word address, or a nonaddress value.

- Mandatory Indirectness. Mandatory indirectness is indicated by an unbracketed asterisk, *. The asterisk prefix must be used and must be followed by a word address (excepting as specifically noted) of a word whose contents are as described in the text for the given symbol.

  Note that indicated indirectness at the source-code level (i.e., in the procedure call), does not necessarily imply a corresponding indirectness at the machine-instruction level (i.e., in the procedure expansion).

ASAM (ASSISTED SEQUENTIAL ACCESS METHOD)

ASAM is the most generally applicable of the assisted methods, in terms of file organization, record format, and range of storage media. It is the only assisted method that is applicable to card reader, card punch, printer, and magnetic tape files, and to device type (DEV assignment) files in general. (VSAM is the only other access method applicable to nonmagnetic devices and magnetic tape files.)

See foldout Chart A-3 of Appendix A for the syntax of the ASAM I/O processing procedures, in reference form.

GENERAL USAGE RULES

The general usage rules for ASAM are listed below.

1. Creates sequential-organization files (output mode).

2. Can be used to read or modify sequential, indexed, and partitioned files (input or update mode for I and P files; input, update, or backward-read mode for C files).

3. Applicable to all media categories as follows:

   a. Input and output mode (all media).

   b. Backward-read mode (magnetic media only).

   c. Update mode (direct-access media only).

4. Not applicable to nonstandard (DEV type) disk volumes.

5. Applicable to F, V, or U format records, blocked or unblocked in the case of F or V format.

6. Applicable to nonstandard magnetic tape files, i.e., unlabeled or otherwise nonstandard (DEV) type) assignments.

7. Is compatible, in U format, with VSAM/VDAM created files.

8. Applicable I/O processing procedures:

M:GET	M:CVOL
M:PUT	M:NOTE
M:TRUNC	M:POINT
M:DELREC	M:DEVICE

9. Usage of either M:DCB or M:MOVEDCB and of M:OPEN and M:CLOSE is mandatory for a given DCB/file; usage of M:SETDCB and M:ASSIGN is optional.

M:DCB Assembly-Time DCB Creation

Syntax:

[label(s)] M:DCB (OPL, 'op-label')

[, dcb-parameter, . . .]

Refer to foldout Chart A-2 in Appendix A for the general syntax of M:DCB in reference form, showing the syntax of all DCB parameters. The parameters and values applicable under ASAM are described below, grouped by type of parameter.

FILE PROCESSING PARAMETERS

(ABN, address[, class-code, ...]) – Abnormal Return Parameter. Specifies the address of the user's routine to which control is to be returned if certain abnormal conditions occur during file opening, processing, and closing. Also specifies the class(es) of conditions for which control is to be returned. None, some, or all of the following abnormal class codes can be specified:

Code	Class
X1	File opening/closing abnormalities.
X2	End-of-file/volume abnormalities.
X3	File processing abnormalities.
X4	BOF user tape label processing.
X5	EOF/EOV user tape label processing.
X6	BOV user tape label processing.

The address field is mandatory if this parameter is specified. If no class code is specified, no abnormal conditions are returned for user routine handling (see M:SETDCB description for special usage). The section "Processing of Abnormal and Error Conditions" later in this chapter describes all abnormal conditions and corresponding codes in detail, and provides information on the type of processing allowed in user's abnormal- and error-handling routines. Occurrence of an abnormal condition that has not been selected, by class, for user routine handling will cause a program abort.

(AM, AS) – Access Method Parameter. The value AS specifies usage of ASAM for I/O processing via this DCB. This parameter need not appear, since AS is the default value.

(BFA, byte-address) – Buffer Address Parameter. Specifies the byte address of the first of a set of user reserved I/O buffers, if any. If more than one such buffer exists (BFA $\neq 0$ and NBF > 1), the set is assumed to be contiguous. The length of each buffer is assumed to equal the block length (BKL parameter). If this parameter is omitted (or the address specified is zero) at opening time, the system will reserve sufficient buffer space in an amount related to the BKL and NBF values in force at that time, or 256 x NBF words in the case of symbiont files (see "Buffer Usage" for further details).

(ERR[, address]) – Error Return Parameter. Specifies the address of the user's routine to which control is to be returned if any error conditions occur during processing via this DCB. If this parameter is omitted or the address is omitted, the occurrence of any error condition will cause a program abort. See the section "Processing of Abnormal and Error Conditions" later in this chapter for a detailed description of the possible error conditions and corresponding codes.

(MOD, BIN | BCD | EBC | PK | UPK) – Data Mode Parameter. Specifies the mode of data encoding to be expected on input or produced on output. The meaning and applicability of the alternative mode keywords (select one) are as follows:

Key	Mode	Applicable To
BIN	Binary	7-track magnetic tape.
BCD	"Old" (026) BCD	7-track magnetic tape.
EBC	EBCDIC	Cards, 7-track tape, disk, and printer.
PK	Packed	7-track magnetic tape.
UPK	Unpacked	7-track magnetic tape.

The default mode is EBC. Mode must be EBC for symbiont files (IN, OUT, SCR, SCP, SLP). BCD, PK, and UPK are primarily for compatibility with second generation equipment.

(NBF, value) – Number Of Buffers Parameter. Specifies the number of I/O buffers (either user- or system-provided) to be used by the M:GET/M:PUT procedures. The default is 1 in the case of the input symbiont IN (NBF forced to 1 if $\neq 1$ at open time); 2 in all other cases.

Note: A minimum of two buffers is required for any card punch file (CP or SCP) (see "Buffer Usage").

(NRT) – No Retry Parameter. This parameter specifies that the system is not to execute the standard error recovery procedure (i.e., repeated retries of the same I/O operation) in case of a device or transmission error.

(ORG, C) – File Organization Parameter. Specifies the organization of the file to be created or accessed. If this parameter is specified, C (sequential) must be specified for file creation (output mode), or for any magnetic tape or nonmagnetic device processing C is the default. I (indexed) or P (partitioned) may be specified if the respective indexed or partitioned file is to be opened in input or update mode only (U format not compatible with update mode). D (direct) may be specified along with U format (FRM parameter) only, for reading disk files created by VDAM. When accessing an existing disk file with FRM, F or FRM, V specified, the actual file organization will override any ORG specification.

(TLB, address) – User Label Area Parameter. Specifies the address of a user program area into which the system will read a user label from magnetic tape, or from which the system will write a user supplied tape label (if so directed) at open, volume-switching, or close time. At least one of the ABN class codes X4, X5, or X6 must also be specified for this parameter to be meaningful (see previous discussion of ABN parameter).

BLOCK LEVEL PARAMETERS

(BHR, value) – Block Header Parameter. Specifies the size of the data block header; it is applicable only to F or V record format files on magnetic media. If specified for disk files, the value 0 or 4 must be specified (see "Block

Formats" in Chapter 6 for the results of various combinations of BHR values and the NBC parameter for magnetic tape files). The default values for magnetic tape files are

F Format { 0 if NBC is specified.
4 if NBC is not specified.

V Format { 2 if NBC is specified.
4 if NBC is not specified.

These defaults also represent the minimum allowable (non-zero) specifications (a zero value always produces the default result).

(BKL, value) – Block Length Parameters. Specifies the block length, in bytes. The length specified must include the block header, if any. The default value, by media type, are

80 for EBCDIC card reader/punch files.

133 for printer files.

1024 for magnetic tape and disk files.

If a value greater than the applicable default is specified for nonmagnetic device files, the specified value will be replaced by the default value (open time). If a value less than the applicable default (80) is specified for card punch files, a program abort will result. For magnetic media files, a value less than or greater than the default value may be specified (up to a maximum of 32K-1 bytes).

(NBC) – No Block Count Parameter. Presence of this keyword parameter specifies that no block sequence numbering (on output) and no block counting (on input) will be performed. It is applicable to magnetic tape files only. The presence or absence of this parameter affects the default value of the BHR parameter (see BHR); its presence at file creation time effectively suppresses two bytes of the normal default tape-block header.

RECORD LEVEL PARAMETERS

(DLC) – Deletion Control Parameter. Presence of this keyword parameter specifies that the first byte of the record body (i.e., the user data) is to function as a deletion control character. It is applicable to disk files only. This parameter is effective only when specified at file creation time; an existing file's HDR2 label value for DLC will override in all other cases (excepting access in U format). See M:DELREC procedure for description of DLC character function.

(FRM, F|V|U) – Record Format Parameter. Specifies the record format of the file to be created or accessed: fixed, variable, or undefined. The default is F (fixed format). For card input files, only F may be specified. When accessing a labeled file and F or V is specified, the actual record format (as described in the file's HDR2 label) will override the format specified.

(LOC | MOV) – Locate/Move Record Parameter. This alternative-keyword parameter specifies that the records are to be accessed either in locate mode (LOC), i.e., no record movement, or in move mode (MOV). MOV is the default.

Note: M:PUT does not operate LOC mode when the DCB is opened for update (U) processing.

(REL, value) – Record Length Parameter. Specifies the length of the records to be processed, in bytes (the actual length for F format records or maximum length for V or U format). In V format, the length specified is that of the maximum size record body, excluding record header bytes. The default REL value is calculated (at open time) as follows:

Tape/disk file, F or U format – REL = BKL - BHR.

Tape/disk file, V format – REL = BKL - BHR - 4.

Printer/card-punch file – REL = BKL - (DTA-1).
(DTA default value = 1.)

Card input file – REL = BKL.

DEVICE LEVEL PARAMETERS

The following parameters apply only with respect to the specific type of device for which they are meaningful. They are otherwise ignored.

(CNT, value) – Page Count Parameter. Indicates, for line printer output, that a page number be printed starting in the column specified ($1 \leq$ value ≤ 129) at the top of each page of printed output.

(DTA, value) – First Print/Punch Column Parameter. Specifies, for printer or card punch files, the device column in which printing/punching of data records is to begin. Default value is 1.

(HDR, value, byte-address) – Page Header Parameter. Indicates, for line printer output, that a page header (i.e., title) is to be printed at the top of each page, starting in the specified column ($1 \leq$ value ≤ 131). The second variable specifies the byte address of a character string in the user's program that constitutes the title. The string must be in TEXTC format (see the TEXTC directive, Meta-Symbol Reference Manual, 90 09 52).

(LIN, value) – Lines Per Page Parameter. Specifies the number of lines per page of printer output (256 maximum), exclusive of page header if any. The default valus is installation dependent (set by SYSGEN).

(SEQ,'identifier') – Sequence Numbering Parameter. Specifies that an eight-character identifier-plus-sequence-number field is to be punched in columns 73 through 80 of each record of a card output file; the specified four-character identifier is to be punched as the first four characters (columns 73-76) of this field; columns 77 through 80 will contain an automatically incremented decimal sequence number.

(SPC, value-1, value-2) – Line Spacing Parameter. Specifies both line spacing and effective top-of-page line, for line printer output. Value-1 specifies the number of blank lines that are to separate printed lines, i.e., transmitted (blank or nonblank) data records. Value-2 specifies the line number, counted from the physical top-of-page, at which printing of data records is to begin. The default for both values is 1.

(TAB, value, . . .) – Tab Settings Parameter. Specifies a list of up to 16 horizontal tabulation settings; each value represents a printer-page column number used for left-justification of data. The change from one tabulation setting to the next is triggered by the occurrence of an X'05' (horizontal tab) character in the output data. The tab setting values can range from 1 through 131.

(VFC/NVF) – Vertical Format Control Parameter. VFC specifies, for a print file, that the first character of each record body is a vertical format control character. NVF specifies that the first character of the record is not to be so interpreted. NVF is the default value. See Table 7-2 for specific format control codes for Models 7440/7445 Sigma Buffered Line Printers.

Table 7-2. Format Control Codes, Sigma Buffered Line Printers, Models 7440/7445

Hexadecimal Code	Function
60, E0	Inhibit Automatic Space after printer
C1	Space 1 line
C2	Space 2 lines
C3	Space 3 lines
⋮	⋮
CF	Space 15 lines
F0	Skip to channel 0 (bottom of page)
F1	Skip to channel 1 (top of page)
F2	Skip to channel 2
⋮	⋮
F7	Skip to Channel 7

USAGE RULES

1. M:DCB is not an executable procedure. DCB space reservation and explicit parameter setting is performed at assembly time.

2. None of the parameters are indirectly addressable.

M:MOVEDCB Dynamic DCB Creation/Retention

The M:MOVEDCB procedure causes the allocation, during program execution, of a 19-word DCB area with read-only protection in the common-dynamic portion of the user's virtual memory space; the address of this area is returned to the user's program. It also causes the contents of a user specified sending area to be copied into the newly allocated DCB area. For example, the sending area may be a nonprotected area of the user's program where a DCB image or "skeleton" has been built, or a write protected DCB that is not open when the M:MOVEDCB is executed. Effectively, this procedure allows the user to create DCBs during program execution, and/or to move and thereby save inactive DCBs that would otherwise be overlaid or destroyed.

Syntax:

[label(s)] M:MOVEDCB [*]adr-1, (PTR, adr-2)

where

adr-1 is the word address of a 19-word sending area which is to be copied into the newly allocated DCB space.

adr-2 is the word address of a one-word pointer in the user's program into which the address of the newly allocated DCB is to be stored by the system.

USAGE RULES

1. No default values are supplied for any parameter field by M:MOVEDCB.

2. The effect of M:SETDCB, M:ASSIGN, M:OPEN, and M:CLOSE procedures referencing a dynamically created or saved DCB is identical to the effect of the same procedures on an assembled DCB.

M:SETDCB Execution-Time DCB Modification

The M:SETDCB procedure allows the user, during program execution, to (1) "fill in" empty parameter fields in a nonopen DCB, i.e., specify previously unspecified DCB parameters, (2) modify previously specified or defaulted parameters in a nonopen DCB, and (3) modify the ABN and ERR parameters in an open DCB. The effective parameters in cases 1 and 2 are all of the parameters that may be specified in the M:DCB procedure. See foldout Chart A-2 of Appendix A for the complete general syntax of M:SETDCB in reference form. The parameters applicable under ASAM are as previously described for the M:DCB procedure.

M:SETDCB also allows modification of the operational label associated with a DCB.

Syntax:

[label(s)] M:SETDCB [*]dcb-adr,

(OPL, {*adr-label / 'op-label'})[, dcb-parameter, ...]

where

dcb-adr is the address of the DCB to be modified.

adr-label is the address of a word containing an operational label, left-justified and space filled, to be associated with the DCB being modified.

'op-label' is a one- to four-character constant specifying an operational label to be associated with the DCB being modified.

dcb-parameter is a DCB parameter as described previously for the M:DCB procedure.

USAGE RULES

1. All of the DCB parameter addresses and values can be indirectly addressed.
2. If the OPL option appears, the operational label specified therein will replace the operational label (if any) previously specified in the referenced DCB.
3. If the referenced DCB is open at the time of M:SETDCB execution, only the ABN and/or ERR parameter will be effective; any other parameters will be ignored.
4. A DCB closed with the HLD option (temporary close) is considered as open for the purposes of rule 3.
5. If the ABN parameter appears, the set of abnormal class codes specified will replace any such set previously in effect, including the null case where no class code is specified (i.e., any class codes previously in effect can be "turned off" by not specifying any class code).

M:ASSIGN Execution-Time DCB Assignment

The M:ASSIGN procedure allows the user, during program execution, to (1) define a temporary file in secondary storage and assign an operational label to it, or (2) define a permanent file on a physical resource specified by a !ASSIGN command. Effectively, case 1 allows the user to eliminate from the job control deck !ASSIGN commands for invariant "scratch file" assignments, or to define additional temporary files to satisfy dynamically determined program requirements (as M:MOVEDCB can be used to create additional DCBs). Case 2 allows the user to make an execution time choice between several !ASSIGN commands each specifying a different resource, or to redefine a file assignment (status, file name, or space allocation) made by a !ASSIGN command. In case 2, the !ASSIGN command referred to must itself define a permanent, i.e., named, file.

Syntax:

Format 1, for temporary files

[label(s)] M:ASSIGN (OPL, {*adr-1 / 'op-label'})

[, (SIZ, {*adr-2 / size}, {*adr-3 / increment})]

where

adr-1 is the address of a word containing the operational label that identifies the DCB being assigned.

'op-label' is the character constant form of that label.

adr-2 is the address of a word containing the value specifying the primary size of the file.

size is the constant form of that size value.

adr-3 is the address of a word containing the value specifying the file increment (where applicable).

increment is the constant form of that increment value.

Format 2, for permanent files

[label(s)] M:ASSIGN (OPL, {*adr-1 / 'op-label'}),

(UNT, OPL, {*adr-4 / 'op-label-1'}, (NAM,

{*adr-5 / 'filename'}), (STS, {NEW / OLD / MOD})[, (SIZ,

{*adr-2 / size}, {*adr-3 / increment})]

where

adr-1, 'op-label', adr-2, size, adr-3, and increment have the same meaning as in format 1.

adr-4 is the address of a word containing the operational label that identifies a !ASSIGN command specifying the desired physical resource.

'op-label-1' is the character constant form of the operational label described immediately above.

adr-5 is the word (or byte) address of a TEXT-format field containing the name of the file to be created or accessed. The name must be terminated by one or more blanks (see the TEXT directive, Meta-Symbol Reference Manual, 90 09 52).

'filename' is the character constant form of the file name.

USAGE RULES

1. Assignments made via M:ASSIGN are valid only for the job-step in which the procedure is executed, they cannot be maintained over succeeding steps. (Effectively the !ASSIGN option FRE is implicit in the use of an M:ASSIGN procedure.)

2. The usage of M:ASSIGN in format 1 is identical to the usage of a !ASSIGN command with corresponding options specified.

3. In format 2, the UNT, OPL option implies use of the resource (device and volume) defined and allocated to the job-step by the operational label specified.

4. The !ASSIGN command referred to by the UNT, OPL option in format 2, must specify a permanent, i.e., named, file assignment and it must be in force during the job-step in which the M:ASSIGN is executed.

5. In format 2, all options other than UNT have the same meaning and usage as the corresponding options appearing in a !ASSIGN command, assuming the option FIL.

6. The M:ASSIGN procedure in format 1 may redefine an implicit assignment via a (installation dependent) predefined optional label, typically SI, LO, GO, etc.

7. In all cases, the optional label specified by the OPL keyword field establishes the link between a DCB and the file named by the procedure.

8. In format 2, the DCB parameters (if any) specified in the referenced !ASSIGN command are not applied to the DCB assigned via the procedure.

M:OPEN Opening a File

The M:OPEN procedure activates the link between a DCB and a physical file. The DCB is placed in an active, or open, status. ("Opening a file" is equivalent to "opening a DCB".) This procedure must be successfully executed before any I/O operation can be performed on a given file. The M:OPEN procedure performs some or all of the following steps while opening a file:

1. Location of the file to be processed, if it exists.

2. Creation of the file if it does not exist, which implies allocation of resources: volume and space on the volume, and creation of the file labels.

3. Positioning of magnetic tape.

4. Various initializations as necessary: reservation of buffers, completion of the DCB, reservation and completion of communication tables (IOBs) between the access methods and the I/O supervisor, etc. DCB parameter values from an existing file's HDR2 label are written into the DCB (excepting access in U format).

5. Validation and control of file identity, file expiration, file protection, and file sharing; checking for coherence among the DCB parameters, and consistency of these with the processing mode specified in M:OPEN and with the storage media.

Syntax:

[label(s)] M:OPEN [*]dcb-adr, mode[,{CID / NID / PID}]

where

dcb-adr is the address of the DCB to be opened.

mode specifies the desired processing mode, as follows:

- I Input mode (forward reading).
- B Backward reading.
- O Output mode (forward writing).
- U Update mode (reading and modification).

CID specifies file identify checking with no tape positioning (default option).

NID specifies no file identity checking (for tape only).

PID specifies file identity checking with tape positioning.

USAGE RULES

1. Processing mode B is applicable to magnetic media files only. Tape files must be monovolume.

2. Processing mode U is applicable to direct access media files only.

3. The file identity checking and positioning option (CID, NID, or PID) is meaningful for standard magnetic tape files only; for DEV assigned files it is ignored; for disk/RAD files CID is always assumed.

4. The effect of a given processing mode specification (and identity checking option, if tape) is conditioned by the declared status of the file versus the file's actual status. These interrelationships are described below.

RELATIONSHIP OF PROCESSING MODE AND FILE STATUS

The status of a file (NEW, MOD, or OLD) is declared in the file assignment. For each declared file status there is one or several normal combinations of status, file existence/nonexistence, and processing mode. The normal combination(s) does not occasion any abnormal return to the user's program during opening. Certain abnormal combinations (described below) are acceptable but will result in an X1 class abnormal return to the user's program during opening if the DCB specifies an abnormal return with the X1 class set (abort otherwise). Any combination other than those described below will result in a program abort.

The several normal and abnormal combinations are described for each declared file status in the following paragraphs.

- NEW

  Normal combination. Nonexistent file and output mode – the usual case for original file creation; output operations allowed only.

  Abnormal combination. Existent file and output mode – in this case an abnormal return to the user's program occurs. The user can then request recreation of the file by a special exit from his abnormal routine. If the expiration date of the file to be rewritten has not been reached, a further abnormal return occurs; another special exit will allow processing to continue, overriding the nonexpired condition.

- MOD

  Normal combinations.

  1. Existent file and update mode – input and updating operations allowed.

  2. Existent file and output mode – file extension output allowed only.

  3a. Existent file and input mode – input operations allowed only.

  3b. Existent file and backward-read mode – input operations allowed only.

  4. Nonexistent file and output mode – status is changed to NEW; output operations are allowed.

- OLD

  Normal combinations.

  1a. Existent file and input mode – input operations are allowed.

  1b. Existent file and backward-read mode – input operations are allowed.

  2. Existent, expired, labeled file and output mode – output operations allowed (i.e., the file may be rewritten).

  Abnormal combination. Existent, nonexpired, labeled file – in this case an abnormal return to the user's program occurs. A special exit from the user's abnormal routine allows processing to continue. The file can then be rewritten.

Note: See the section "Processing of Abnormal and Error Conditions" later in this chapter for specific abnormal codes and further details on normal/special exit results.

AUTOMATIC EVOLUTION OF FILE STATUS

The automatic evolution of the status of a file during a job-step when opened for output is shown below.

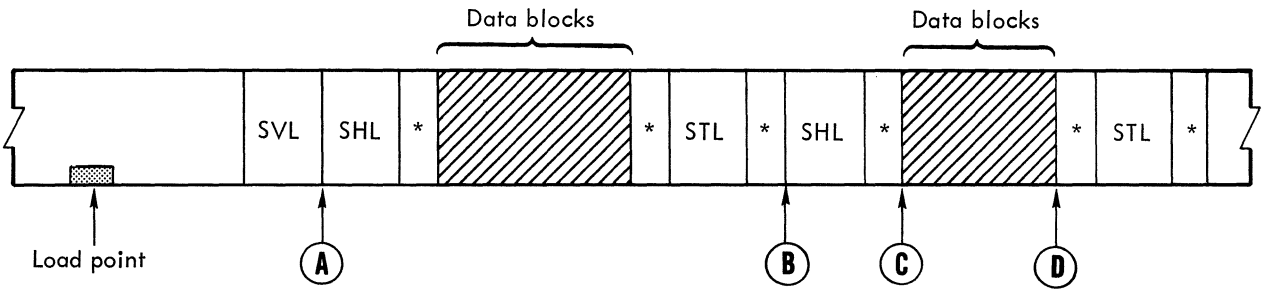
	Declared Status Prior to First M:OPEN, Output		
	NEW, Nonexistent File	MOD, Nonexistent File	OLD, Existent File
Status after successful opening	NEW	NEW	NEW
Status after M:CLOSE	MOD	MOD	MOD
Note: Does not apply to DEV assigned magnetic tape.			

MAGNETIC TAPE POSITIONING AND FILE IDENTITY CHECKING

If a standard magnetic tape volume has not been previously processed since its mounting, its initial position (relative to the tape unit read/write heads) during the opening process is between the standard-volume-label group, SVL (or user-volume-label group, UVL) and the first standard-file-header-label group, SHL. This position is shown in Figure 7-1 as position A.

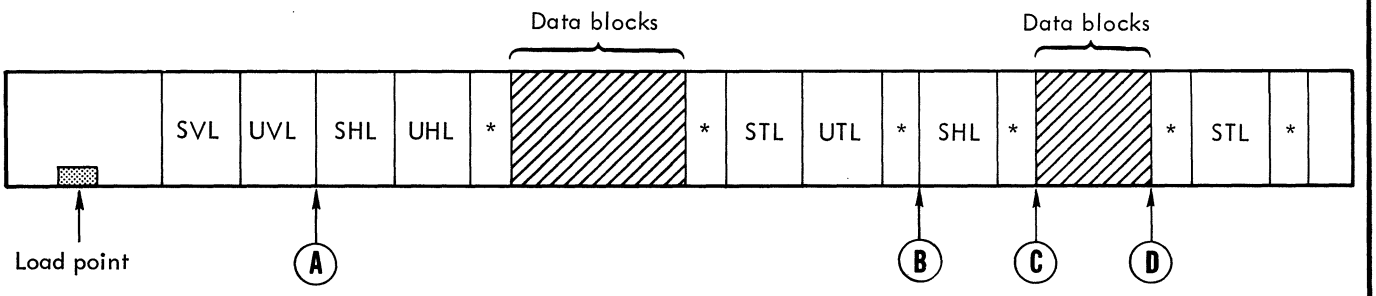
If the volume has been previously processed since mounting, its initial positioning will depend upon the action taken during the previous M:CLOSE operation: between two files (position B in Figure 7-1), in front of the first data block (position C), or after the last data block (position D), of some file on the volume.

The open process will then perform testing and positioning functions from these initial magnetic tape positions. These functions are controlled by the CID, NID, and PID options. The default option is CID. Only the CID option is significant for disk/RAD files. Several cases exist, as described in the following paragraphs.



* = tape mark

Volume With No User Labels



Volume With User Labels

Figure 7-1. Pre-positioning of Tape Volume During M:OPEN

First Case, File Opened for Creation (STS, NEW: Processing Mode Out)

- CID

  For disk files, this option ensures that the identification assigned to the file being created does not already exist in the volume or account catalog.

  For magnetic tape files, no positioning is executed. There is a verification that a file having the same identification as the file being created does not already exist at the current position on the tape. If the file at this location has a different identity and its expiration date has been reached, it is written over by the new file. In the cases where the file has the same name and/or the expiration date has not been reached, a return is made to the user's abnormal routine. This permits the user to request creation of a new file replacing the old (special exit), or not to complete the open (normal return). (See "Processing of Abnormal and Error Conditions".)

  If there is no file at the initial position on the magnetic tape, the opening of the file to be created continues normally.

- NID

  There is no checking of file identification; only a test of the expiration date of the file at the initial position is performed.

- PID

  This option causes automatic creation of the new file following all of the old files contained on the tape. A test of the file identification is made during positioning against all files encountered between the initial position and the final old file. If one of the files encountered has the same identity as the file to be created, a return is made to the user's routine. Then, using a special exit, the user can write over this old file; otherwise the M:OPEN is not executed.

  Figure 7-2 summarizes the positioning and identification test functions caused by these options.

Second Case, File Opened in Forward Read.

- CID

  For magnetic tape, this option causes testing of the current file identification. For disk, the catalog is searched for the corresponding file.

  When the identifications are not equal, a return is made to the users abnormal routine. For a magnetic tape file, the user can request reading of the present file (special exit); otherwise the open is not executed (normal exit).

- NID

  No checking or positioning is executed. The opening is to the first file encountered.

- PID

  This option causes a forward search for the identification specified by the !ASSIGN command, starting from the initial location on magnetic tape at opening time. If the two tape marks identifying the end of volume are encountered, a tape rewind is executed and a return is made to the user's abnormal routine. Using a special exit, the user can reinitialize a forward search for the file to be processed. If the file is not found after this second search, an abnormal return is again made but the open will not be executed.

Third Case, File Opened in Backward Read. In this case, the testing refers to the file preceding the one at which the tape is initially positioned. The initial positioning to end-of-file is the responsibility of the user program.

- CID

  This option causes identification checking of the preceding file.

- NID

  No checking is performed.

- PID

  This option, associated with B processing mode, is identical to CID. That is, no backward search is performed.

Note: See the M:CLOSE positioning options LVE and RRD for end-of-file position on a temporary (HLD) file closing.

M:CLOSE Closing a File

This procedure causes the closing of a file. The close effects a suspension or halt of activity via a DCB and possibly via the processed file. Its main functions are:

1. Waiting for and testing the last I/O operation(s).
2. Writing current (last) buffer where necessary.
3. Checking or writing the end-of-file labels on magnetic tape.
4. Updating and validating disk file labels.
5. Temporary or definite closing of the DCB.
6. Partial restitution of disk space or deletion of the file created by the job.
7. Releasing devices, buffers, tables, etc.

Functions 6 and 7 are conditional upon the type of closing.

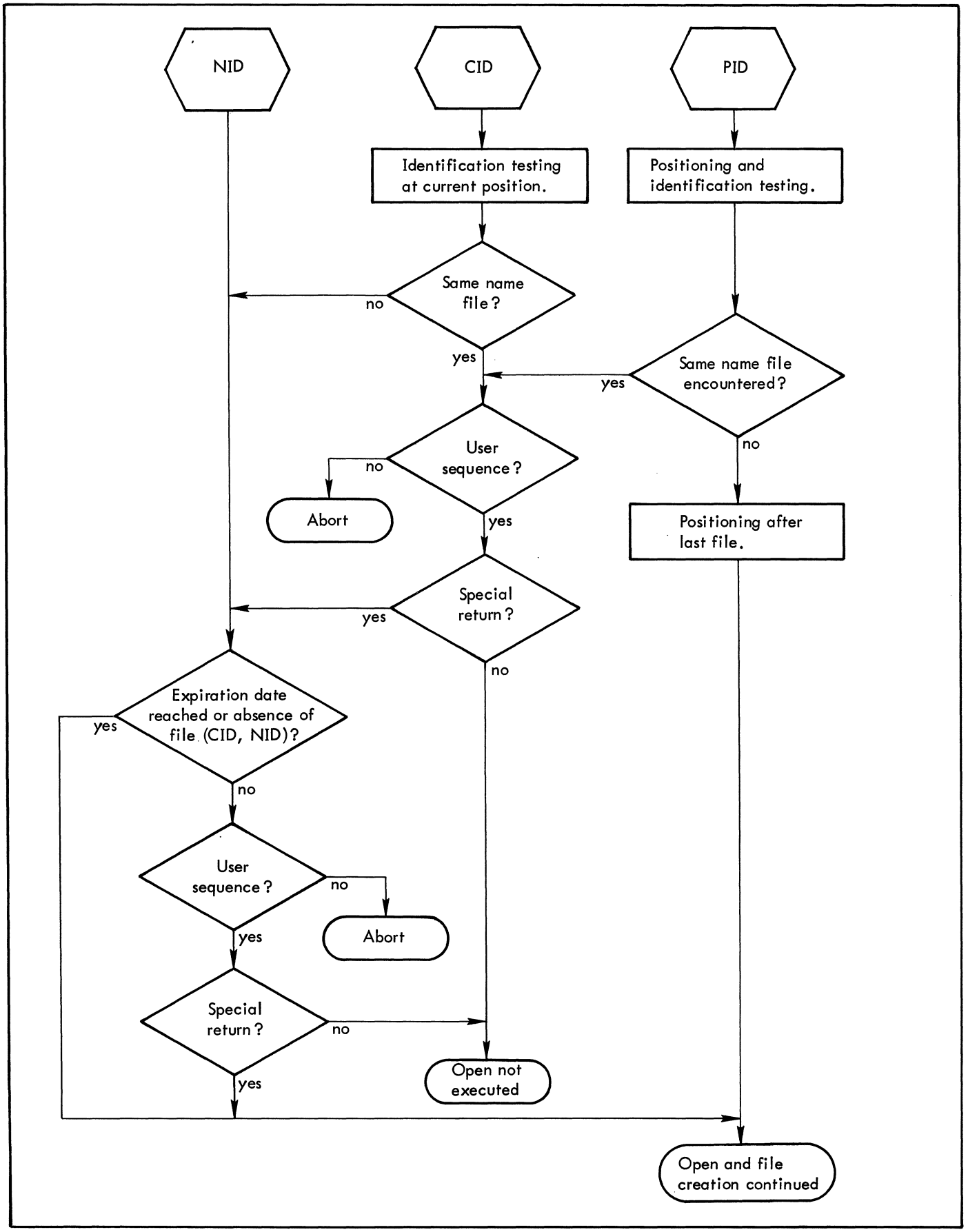


Figure 7-2. Flowchart of M:OPEN Action for File Creation on Magnetic Tape

Syntax:

[label(s)] M:CLOSE [*]dcb-adr [,{HLD / MTN / RLS}] [,{RRD / LVE / RWD}] [,{KEP / NCG}]

where dcb-adr is the address of the DCB to be closed.

TYPES OF CLOSE

HLD specifies a temporary closing of the files. The DCB-file link is maintained. A new M:OPEN applied to the DCB will only permit changing of the processing mode of the file, because only a small portion of the open functions will then be executed.

MTN specifies a definite close. It cancels the DCB-file link but maintains the job-file link. In the same job another DCB can be used to process the same file. The resource is not released even if the FRE option was specified in the assignment.

RLS specifies a definite close. It cancels the DCB-file link and the job-file link. The resource(s) allocated for the file is released if the FRE option was specified in the assignment.

POSITIONING AFTER CLOSING

The options controlling positioning, RRD (reread), LVE (leave), and RWD (rewind), are significant only for sequential organization files on magnetic media (refer to Figure 7-3).

The significance of these options depends on the type of close: temporary (HLD) or definite (MTN or RLS). It also depends on the processing mode of the file: forward (I, O, and U) or backward (B). The options have significance for a disk file only on a temporary close.

Temporary Close (HLD). Upon closing a file, positioning remains within the file being accessed, or within the volume of the file being accessed in the case of a serially mounted multivolume file. This means the file is positioned (in the case of tape) somewhere between the two tape marks delimiting the data blocks.

For I or O processing (see Figure 7-3):

RRD requests postioning before the first data block of the file (or portion of the file).

LVE requests positioning after the last data block of the file (or portion of the file).

RWD is not meaningful in this type of close. In this case this option is equivalent to RRD.

For B processing (see Figure 7-3), the RRD and LVE options have positioning meanings in reverse of those for forward processing:

RRD requests positioning after the last data block of the file (or portion of the file).

LVE requests positioning before the first data block of the file (or portion of the file).

RWD is equivalent to LVE in this case.

Definite Close (MTN or RLS). Positioning is always made outside the file limits (the labels and tape marks delimiting the file), or the volume limits in the case of a serially mounted multivolume file.

For I or O processing on magnetic tape (see Figure 7-3):

RRD requests positioning before the SHL of the file (or portion of the file).

LVE requests positioning after the STL of the file (or portion of the file).

RWD requests rewinding and positioning after the volume SVL.

For B processing on magnetic tape (see Figure 7-3), the RRD and LVE options have positioning meanings in reverse of those for forward operations:

RRD signifies positioning after the STL of the file (or portion of the file).

LVE requests positioning before the SHL of the file (or portion of the file).

RWD requests rewinding and positioning after the volume SVL.

The positioning described never crosses over volume boundaries for serially mounted multivolume files.

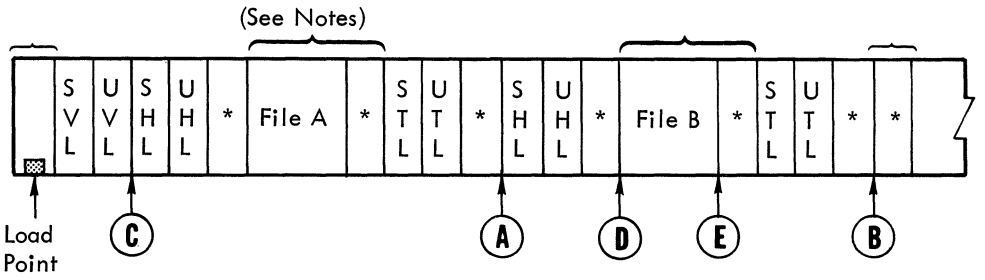
A rewind operation is used by M:CLOSE only for the RWD option in an RLS or MTN closing. In the other cases the positionings are executed using only tape-mark-search and skip-block operations.

LABEL VALIDATION AND FILE DISPOSITION

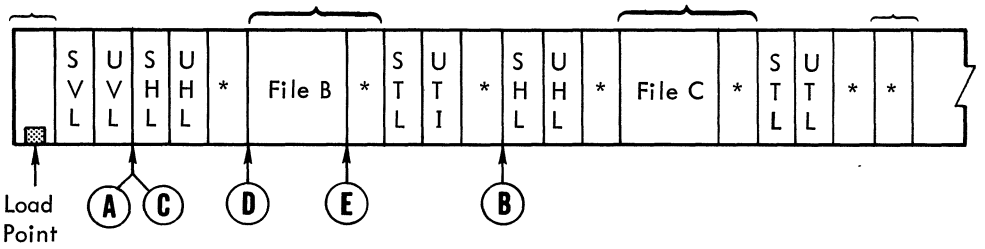
The KEP and NCG options specify the disposition of the permanent file. These are significant only for the O processing mode (file creation).

KEP Requests validation of the file labels for a disk/RAD file, and cataloging of the file if CTG was specified in the assignment (or the file was created on the account volume).

Volume 1



Volume 2



Mode of OPEN	Close Option					
	Definite			Temporary		
	RRD	LVE	RWD	RRD	LVE	RWD
I, O, or S	A	B	C	D	E	D
Backward	B	A	C	E	D	D

In all cases, File B is the file being closed.

Case 1: Multifile standard volume – consider volume 1 alone or volume 2 alone.

Case 2: Multifile nonstandard volume – same as case 1, but considering only the braced portions of the diagrams.

Case 3: Multifile multivolume – consider that volume 1 contains the first portion of File B and volume 2 the second (and last) portion of File B where volume 2 is being processed.

Notes: For an unlabeled tape volume, all label groups (SVL, UVL, SHL, UHL, STL, and UTL) are eliminated as are the tape marks which demarcate the end of each label group. The braces show the portions which do appear for an unlabeled volume.

For a standard tape volume, any user label group (UVL, UHL, or UTL) which is not created by the user does not appear.

Figure 7-3. Tape File Positioning at Close Time

NCG requests deletion of the file for a file created on a disk/RAD volume and suppression of cataloging if cataloging has been requested or implied. This option is applicable only in conjunction with RLS and if the status of the file is NEW prior to closing.

USAGE RULES

1. If no type-of-close option is specified, RLS (release) is assumed by default. Both the DCB-file connection and the job-file connection are severed (i.e., another job can attempt access to the file whether the file/volume is sharable or not). All device (i.e., common) resources allocated for the processing of this file are released if MTN was not specified in the assignment.

2. If no positioning option is specified, RWD (rewind) is assumed by default.

3. If no validation/disposition option is specified, KEP (keep) is assumed by default.

4. If a definite close (MTN or RLS) is not issued for a given DCB/file before the end of the job-step, the system forces an unconditional (RLS) close, disposition KEP.

M:GET Get Next Sequential Record

The M:GET procedure permits reading of the next logical record of an existing file, relative to the record last accessed, if any; otherwise the first sequential record of the file. (Exception: see M:POINT procedure.) The accessed records will either be moved into a work area specified in the procedure, or be located (i.e., pointed to in the I/O buffer) at the user's option (MOV/LOC mode parameter of the DCB).

Syntax: See foldout Chart A-3 in Appendix A.

MEANING OF THE PARAMETERS

dcb-adr specifies the address of the DCB through which records are to be read.

REC, $\left\{\begin{matrix}[*]badr-1\\adr-2\end{matrix}\right\}$ specifies the byte address (badr-1) of the user's receiving area in MOV mode, or the address (adr-2) of a word into which a byte address pointer to the accessed record is to be stored by the system in LOC mode.

RSA, [*]adr-3 optionally specifies the address (adr-3) of a word into which the byte length of the accessed record is to be stored (by the system).

USAGE RULES

1. If MOV mode is in effect (in the referenced DCB) a byte address (optionally indirect) must be specified for the REC parameter. If LOC mode is in effect, a word address (direct only) must be specified.

2. If the RSA parameter is specified, the length of the logical record moved or located is reported for all record formats. For V format records, the length of the record body only is reported.

3. For V format records, the record body only is moved or located (i.e., the four-byte record header is not included).

4. In backward-read processing mode, only F or U record format is applicable.

5. M:GET is not valid in output processing mode.

M:PUT Put Next Sequential Record

The M:PUT procedure permits writing of the next logical record of a file being created or rewritten, or, in update processing mode, rewriting of the logical record last accessed via M:GET. This procedure operates in either MOV mode (with record movement) or LOC mode (no record movement) except in update processing where MOV mode is implicit.

Syntax: See foldout Chart A-3 in Appendix A.

MEANING OF THE PARAMETERS

dcb-adr specifies the address of the DCB through which records are to be written.

REC, $\left\{\begin{matrix}[*]badr-1\\adr-2\end{matrix}\right\}$ specifies the byte address (badr-1) of the user's sending area in MOV mode, or the address (adr-2) of a word into which a pointer to a record position (in the I/O buffer) is to be stored by the system in LOC mode.

ARS, [*]value specifies, for output processing only, the length of the V or U format record to be written.

USAGE RULES

1. If MOV mode is in effect (in the referenced DCB), a byte address must be specified directly or indirectly for the REC parameter. If LOC is in effect, a word address (direct only) must be specified.

2. M:PUT operates only in MOV mode when the DCB is open for update processing. The MOV/LOC parameter in this case is significant only for M:GET.

3. M:PUT operates only in MOV mode for line printer and card punch files; the MOV/LOC parameter must be MOV.

4. The ARS option is effective only in output processing mode, and only for V and U format records; it is ignored in update processing and for F format records.

5. For F format records, the effective record length is taken from the DCB (REL parameter).

6. In update processing mode, the record to be written replaces the last record accessed with the M:GET procedure; the length of the latter record automatically determines the length of the record to be written.

7. In MOV mode, the REC parameter specifies the initial byte location of the record that is to be moved to the I/O buffer. In V format, the record body only is to be supplied by the user (the system affixes the record header).

8. In LOC mode, the REC parameter specifies the location of a word in the user's program into which the system stores a byte-address pointer to the beginning of the next available record space in the I/O buffer (where the user may build his record).

9. In LOC mode, the ARS parameter (when effective) specifies the length of the record to be constructed at the point indicated by the pointer returned (at adr-2) by the M:PUT execution.

10. In LOC mode, any execution of M:PUT returns a pointer to the origin in the buffer for the record to be built (subsequently) by the user. It establishes, explicitly or implicitly, the length of that record and causes it to be "written". That is, n executions of M:PUT in LOC mode cause n records to be written.

11. Following an opening in output mode of an existent file whose declared or evolved status is NEW, M:PUT executions cause the existing file to be overwritten (i.e., the existing file is recreated).

12. Following an opening in output mode of an existing file whose status is MOD, M:PUT executions cause the existing file to be extended (i.e., initial positioning for M:PUT is to end-of-file).

M:TRUNC Truncation of a Block

The M:TRUNC procedure permits termination of sequential operations on a partially processed block and passage to the next sequential block for subsequent processing. In output processing mode the current block is truncated (i.e., the partially completed block is written to the physical file). In input or update processing mode, the record accessed by the first M:GET following an M:TRUNC execution is the first record of the next sequential block. In update mode, the complete current block (with modifications, if any) is rewritten to the file.

This procedure is in general meaningful only for files containing blocked records (i.e., more than one record per block); it has no effect when used in conjunction with U record format.

Syntax:

[label(s)] M:TRUNC [*]dcb-adr

where dcb-adr is the address of the DCB.

USAGE RULES

1. M:TRUNC is not applicable to backward-read processing mode.

2. For F or V format records, if no record has yet been read from or written to the current block (i.e., the current I/O buffer), M:TRUNC has no effect. For U format record processing M:TRUNC is ignored.

3. M:TRUNC does not cause physical truncation of the current buffer in update processing mode.

M:DELREC Delete a Record

The M:DELREC procedure permits deletion of the last logical record accessed with M:GET in update processing mode (disk/RAD files only) when the DLC parameter is in effect in the DCB. Execution of this procedure causes the value X'FF' to be placed in the first byte of the record body, the first byte being defined as a deletion control character. This effectively deletes that record for the assisted access methods.

Syntax:

[label(s)] M:DELREC [*]dcb-adr

where dcb-adr is the address of the DCB.

USAGE RULES

1. To be effective, the DLC parameter, specifying existence of the deletion control character, must be specified at file creation time (see M:DCB). On subsequent file accesses the value carried in the HDR2 label of the file overrides the setting in the DCB prior to open time.

2. M:DELREC is applicable only in update processing mode. Therefore it is applicable to disk/RAD files in F or V format only.

M:CVOL Switch To Next Volume

The M:CVOL procedure permits switching to the next sequential volume during the processing of a multivolume file. It can be executed at any time during the processing of a given volume, in input, output, or update mode, if there is a subsequent volume to process. If the subsequent volume is not already mounted, the system will issue a mounting request to the operator.

Syntax:

[label(s)] M:CVOL [*]dcb-adr

where dcb-adr is the address of the DCB.

USAGE RULES

1. If no subsequent volume of the file exists (or has been allocated) at the time of an M:CVOL execution, an X2-class abnormal return is made to the user's program during input processing. (If the appropriate abnormal class is not set in the DCB, a program abort occurs.)

2. During output processing, an M:CVOL request will cause a public volume to be mounted if no more assigned volumes remain to be switched (serial mounting mode only).

M:NOTE Note Current Position

The M:NOTE procedure permits the user to obtain a pointer to his current block/record position during input or update processing of a file. This pointer can then subsequently be used with the M:POINT procedure, described below, to reposition to the point at which the M:NOTE was issued.

Syntax:

[label(s)] M:NOTE [*]dcb-adr, (RCI, [*]adr-1)

where

dcb-adr is the address of the DCB.

adr-1 is the address of a two-word area in the user's program into which the current-position pointer is to be stored by the system.

USAGE RULES

1. The address specified in the RCI parameter must reference the first of two contiguous words, into which the system will store a pointer to the last record accessed with an M:GET procedure.

2. On return, word adr-1 contains the relative block number of the current block and word adr-1 + 1 contains the byte displacement of the current record relative to the beginning of the block.

3. This procedure is applicable to input and update processing modes only.

M:POINT Reposition by Pointer

The M:POINT procedure permits repositioning, within the file being processed, to a record pointed to by information obtained via a previously issued M:NOTE. This repositioning can extend across block boundaries, forward or backward.

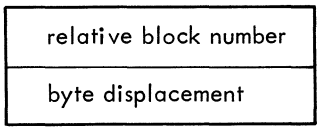
Syntax:

[label(s)] M:POINT [*]dcb-adr, (RCI, [*]adr-1)

where

dcb-adr is the address of the DCB.

adr-1 is the address of a two-word pointer containing repositioning information of the following form:



USAGE RULES

1. The pointer returned by the M:NOTE procedure may be used to supply the positioning information required by M:POINT.

2. If the pointer supplied to M:POINT refers to a position outside the limits of the file currently being processed, an X3-class abnormal return will occur. (If the X3 ABN class is not set in the DCB, a program abort will result.)

3. This procedure is applicable to input and update processing modes only.

M:DEVICE Device Dependent Operations

The M:DEVICE procedure permits the user to request device dependent operations (i.e., operations having a meaning specific to a certain type of device). This procedure is applicable either to line printer files or to magnetic tape files, depending on the chosen option.

The operations that may be specified are page ejection and print-form changing for printer files, and block level repositioning for magnetic tape files.

Syntax: See foldout Chart A-3 of Appendix A.

MEANING OF THE OPTIONS

(CHF,[*]badr) specifies the byte address (optionally indirect) of a user-supplied message to the operator, in TEXTC format, requesting a change of print forms. (See the TEXTC directive, Meta-Symbol Reference Manual, 90 09 52.)

(PAG) requests a page ejection during the printing of a file.

(POS,BKS|FWS|BOF|EOF) requests repositioning of a magnetic tape file as specified by one of the suboptions listed below:

BKS specifies a one-block backspacing.

FWS specifies a one-block forward spacing.

BOF specifies positioning to the first block of the file.

EOF specifies positioning behind the last block of the file (i.e., to end-of-file).

USAGE RULES

1. If the requested operation does not have a defined meaning for the type of device currently assigned, the request is ignored.

2. The POS suboptions BKS, FWS, and BOF effect positioning at the beginning of the block implied.

3. The POS suboption BKS results in positioning to the block immediately preceding the block at or within which the file was positioned when M:DEVICE was executed.

4. The POS suboption FWS results in positioning to the block immediately following the block at or within which the file was positioned when M:DEVICE was executed.

5. Following a CHF (change forms) request, a return is made to the user's program only after the request has been satisfied (if a print file is in fact being processed).

AIAM (ASSISTED INDEXED ACCESS METHOD)

AIAM is oriented toward direct access processing of indexed-sequential (I) organization files (i.e., files whose records are ordered according to the relative value of a key field contained in each record). (See Chapter 6 for a description of indexed-sequential file organization.) AIAM also allows sequential accessing of indexed files, starting from an initial record accessed by key value.

See foldout Chart A-4 of Appendix A for the syntax of the AIAM I/O procedures in reference form.

GENERAL USAGE RULES

The general usage rules for AIAM are listed below.

1. Creates and processes indexed-sequential organization (direct access) files only. (Multivolume files must be parallel mounted.)

2. Applicable to direct access (disk/RAD) storage media only.

3. Not applicable to nonstandard (DEV assigned) volumes.

4. Applicable processing modes:

   Output
   Input
   Update

5. Applicable to F or V format records only.

6. Applicable I/O processing procedures:

   M:GET M:TRUNC

   M:PUT M:DELREC

7. Usage of either M:DCB or M:MOVEDCB and of M:OPEN and M:CLOSE is mandatory for a given DCB/file; usage of M:SETDCB and M:ASSIGN is optional.

M:DCB Assembly-Time DCB Creation

Syntax:

[label(s)] M:DCB (OPL,'op-label')[,dcb-param,...]

Refer to foldout Chart A-2 in Appendix A for the general syntax of M:DCB, in reference form, showing the syntax of all DCB parameters. The parameters and values applicable under AIAM are described below, grouped by type of parameter.

FILE PROCESSING PARAMETERS

(ABN,address[,class-code,...]) – Abnormal Return Parameter. Specifies the address of the user's routine to which control is to be returned if certain abnormal conditions occur during file opening, processing, and closing. Also specifies the class(es) of conditions for which control is to be returned. None, some, or all of the following abnormal class codes can be specified:

Code	Class
X1	File opening abnormalities.
X2	End-of-file/volume abnormalities.
X3	File processing abnormalities.

The address field is mandatory if this parameter is specified. If no class code is specified, no abnormal conditions are returned for user routine handling (see M:SETDCB description for special usage). The section "Processing of Abnormal and Error Conditions" later in this chapter describes all abnormal conditions and corresponding codes in detail, and provides information on the type of processing allowed in users' abnormal- and error-handling routines. Occurrence of an abnormal condition that has not been selected, by class, for user routine handling will cause a program abort.

(AM, AI) - Access Method Parameter. The value AI specifies usage of AIAM for I/O processing via this DCB. This parameter must be specified.

(BFA, byte-address) – Buffer Address Parameter. Specifies the byte address of the first of a set of user reserved buffer areas, if any. If more than one such buffer exists (BFA $\neq$ 0 and NBF > 1), the set is assumed to be contiguous. The length of each buffer is assumed to equal the block length (BKL parameter). If this parameter is omitted (or the address specified is zero) at opening time, the system will acquire buffer space in an amount related to the BKL and NBF values in force at that time. (See "Buffer Usage" for further details.)

(ERR, address) – Error Return Parameter. Specifies the address of the user's routine to which control is to be returned if any error condition occurs during processing via this DCB. If this parameter is omitted or the address is omitted, the occurrence of any error condition will cause a program abort. See the section "Processing of Abnormal and Error Conditions" later in this chapter for a detailed description of the possible error conditions and corresponding codes.

(MOD, EBC) – Data Mode Parameter. Specifies the mode of data encoding to be expected on input or produced on output. The mode keyword EBC specifies EBCDIC encoding, the only mode applicable. The default mode is EBC.

(NBF, value) – Number Of Buffers Parameter. Specifies the number of buffers (either user- or system-provided) to be used by the M:GET/M:PUT procedures. The default value is 2. The minimum specifiable value for output and update mode is 2. The minimum value for input mode is 1.

(NRT) – No Retry Parameter. Specifies that the system is not to execute the standard error recovery procedure (i.e., repeated retries of the same I/O operation) in case of a device or transmission error.

(ORG, I) – File Organization Parameter. Specifies the organization (indexed-sequential) of the file to be created or accessed.

BLOCK LEVEL PARAMETERS

(BHR, value) – Block Header Parameter. Specifies the size of the data block header. Only the value 4 (or 0) may be specified (see "Block Formats" in Chapter 6). The default value is 4.

(BKL, value) – Block Length Parameters. Specifies the block length, in bytes. The length specified must include the header. The default value, by media type, is 1024 for disk/RAD files. A block length value less than or greater than the default value may be specified (up to a maximum of 32K-1 bytes). See "Block Formats" in Chapter 6 for a discussion of block length. Note that both a four-byte block header and a four-byte linkage word are included in each I-organization data and overflow block.

RECORD LEVEL PARAMETERS

(DLC) – Deletion Control Parameter. Presence of this keyword parameter specifies that the first byte of the record body (i.e., the user data) is to function as a deletion control character. This parameter is effective only when specified at file creation; an existing file's HDR2 label value for DLC will override in all other cases. See M:DELREC procedure for description of DLC character function.

(FRM, F | V) – Record Format Parameter. Specifies the record format of the file to be created or accessed, fixed or variable. The default is F (fixed format). When accessing a labeled file and F or V is specified, the actual record format (as described in the file's HDR2 label) will override the format specified.

(KYL, value) – Key Length Parameter. Specifies the length of the record key, in bytes. The length value may range from 1 through 255.

(KYP, value) – Key Position Parameter. Specifies the initial byte position of the record key, relative to the first byte of the record body. The first byte of the record body is numbered 0 (zero). The maximum value specifiable is equal to the maximum size of the record body less the key length. The default value is 0.

Note: If DLC is specified, the KYP value may not be 0 (i.e., must not be defaulted to or specified as 0), otherwise, the program will abort.

(LOC | MOV) – Locate/Move Mode Parameter. This alternative-keyword parameter specifies that the records are to be accessed either in locate mode (LOC), i.e., no record movement, or in move mode (MOV). MOV is the default.

Note: M:PUT does not operate LOC mode.

(REL, value) – Record Length Parameter. Specifies the length of the records to be processed, in bytes (the actual length for F format records or maximum length for V format). In V format, the length specified is that of the maximum size record body, excluding record header bytes. The default REL value is calculated (at open time) as follows:

F format – REL = BKL - BHR - 4.

V format – REL = BKL - BHR - 8.

Note that if the REL parameter is not specified for F format records, unblocked records (i.e., one record per block) are implied.

USAGE RULES

1. M:DCB is not an executable procedure. DCB space reservation and explicit parameter setting is performed at assembly time.

2. None of the parameters are indirectly addressable.

M:MOVEDCB Dynamic DCB Creation/Retention

The M:MOVEDCB procedure causes the allocation, during program execution, of a 19-word DCB area with read-only protection in the common-dynamic portion of the user's virtual memory space; the address of this area is returned to the user's program. It also causes the contents of a user specified sending area to be copied into the newly allocated DCB area. For example, the sending area may be a nonprotected area of the user's program where a DCB image or "skeleton" has been built, or a write protected DCB that is not open when the M:MOVEDCB is executed. Effectively, this procedure allows the user to create DCBs during program execution, and/or to move and thereby save inactive DCBs that would otherwise be overlaid or destroyed.

Syntax:

[label(s)] M:MOVEDCB [*]adr-1, (PTR, adr-2)

where

adr-1 is the word address of a 19-word sending area which is to be copied into the newly allocated DCB space.

adr-2 is the address of a one-word pointer in the user's program in which the address of the newly allocated DCB is to be returned.

USAGE RULES

1. No default values are supplied for any parameter field by M:MOVEDCB.

2. The effect of M:SETDCB, M:ASSIGN, M:OPEN, and M:CLOSE procedures referencing a dynamically created or saved DCB is identical to the effect of the same procedures on an assembled DCB.

M:SETDCB Execution-Time DCB Modification

The M:SETDCB procedure allows the user, during program execution, to (1) "fill in" empty parameter fields in a nonopen DCB, i.e., specify previously unspecified or undefaulted DCB parameters, (2) modify previously specified or defaulted parameters in a nonopen DCB, and (3) modify the ABN and ERR parameters in an open DCB. The effective parameters in cases 1 and 2 are all of the parameters that may be specified in the M:DCB procedure for AIAM. See foldout Chart A-2 of Appendix A for the complete general syntax of M:SETDCB in reference form. The parameters applicable under AIAM are as described for the M:DCB procedure.

M:SETDCB also allows modification of the operational label associated with a DCB.

Syntax:

[label(s)] M:SETDCB [*]dcb-adr[,(OPL, {*adr-label / 'op-label'})][, dcb-parameter, ...]

where

dcb-adr is the address of the DCB to be modified.

adr-label is the address of a word containing an operational label, left-justified and space filled, to be associated with the DCB being modified.

'op-label' is a one- to four-character constant specifying an operational label to be associated with the DCB being modified.

dcb-parameter is a DCB parameter as described for the M:DCB procedure.

USAGE RULES

1. All of the DCB parameter addresses and values can be indirectly addressed.

2. If the OPL option appears, the operational label specified therein will replace the operational label (if any) previously specified in the referenced DCB.

3. If the referenced DCB is open at the time of M:SETDCB execution, only the ABN and/or ERR parameter will be effective; any other parameters will be ignored.

4. A DCB closed with the HLD option (temporary close) is considered as open for the purposes of rule 3.

5. If the ABN parameter appears, the set of abnormal class codes specified will replace any such set previously in effect, including the null case where no class code is specified (i.e., any class codes previously in effect can be "turned off" by not specifying any class code).

M:ASSIGN Execution-Time DCB Assignment

The M:ASSIGN procedure allows the user, during program execution, to (1) define a temporary file in secondary storage and assign an operational label to it, or (2) define a

permanent file on a physical resource specified by a !ASSIGN command. Effectively, case 1 allows the user to eliminate from the job control deck !ASSIGN commands for invariant "scratch file" assignments, or to define additional temporary files to satisfy dynamically determined program requirements (as M:MOVEDCB can be used to create additional DCBs). Case 2 allows the user to make an execution time choice between several !ASSIGN commands, each specifying a different resource, or to redefine a file assignment (status, file name, or space allocation) made by a !ASSIGN command. In case 2, the !ASSIGN command referred to must itself define a permanent (i.e., named) file.

Syntax:

Format 1, for temporary files

[label(s)] M:ASSIGN (OPL, {*adr-1 / 'op-label'})

[,(SIZ, {*adr-2 / size}, {*adr-3 / increment})]

where

adr-1 is the address of a word containing the operational label that identifies the DCB being assigned.

'op-label' is the character constant form of that label.

adr-2 is the address of a word containing a value specifying the size of the data block portion of the file.

size is the constant form of that size value.

adr-3 is the address of a word containing a value specifying the size of the index/overflow (increment) portion of the file.

increment is the constant form of that increment value.

Format 2, for permanent files

[label(s)] M:ASSIGN (OPL, {*adr-1 / 'op-label'}),

(UNT, OPL, {*adr-4 / 'op-label-1'})

(NAM, {*adr-5 / 'filename'}) (STS, {NEW / OLD / MOD})

[,(SIZ, {*adr-2 / 'size'}, {*adr-3 / increment})]

where

adr-1, 'op-label', adr-2, size, adr-3, and increment have the same meaning as in format 1.

adr-4 is the address of a word containing the operational label that identifies a !ASSIGN command specifying the desired physical resource.

'op-label-1' is the character constant form of the operational label described immediately above.

adr-5 is the word (or byte) address of a TEXT-format format field containing the name of the file to be created or accessed. The name must be terminated by one or more blanks. (See the TEXT directive, Meta-Symbol Reference Manual, 90 09 52.)

'filename' is the character constant form of the file name.

USAGE RULES

1. Assignments made via M:ASSIGN are valid only for the job-step in which the procedure is executed; they cannot be maintained over succeeding steps. (Effectively, the !ASSIGN option FRE is implicit in the use of an M:ASSIGN procedure.)

2. The usage of M:ASSIGN in format 1 is identical to the usage of a !ASSIGN command with corresponding options specified.

3. In format 2, the UNT, OPL option implies use of the resource (device and volume) defined and allocated to the job-step by the operational label specified.

4. The !ASSIGN command referred to by the UNT, OPL option in format 2, must specify a permanent (i.e., named) file assignment on direct access media and must be in force during the job-step in which the M:ASSIGN is executed.

5. In format 2, all options other than UNT have the same meaning and usage as the corresponding options appearing in a !ASSIGN command, assuming the option FIL.

6. The M:ASSIGN procedure in format 1 may redefine an implicit assignment via a (installation dependent) predefined operational label, typically SI, LO, EI, etc.

7. In all cases, the operational label specified by the OPL keyword field establishes the link between a DCB and the file named by the procedure.

8. In format 2, the DCB parameters (if any) specified in the referenced !ASSIGN command are not applied to the DCB assigned via the procedure.

M:OPEN Opening a File

The M:OPEN procedure activates the link between a DCB and a physical file. The DCB is placed in an active, or open, status. ("Opening a file" is equivalent to "opening a DCB".) This procedure must be successully executed

before any I/O operation can be performed on a given file. The M:OPEN procedure performs some or all of the following steps while opening a file:

1. Location of the file to be processed, if it exists.

2. Creation of the file if it does not exist, which implies allocation of resources: volume and space on the volume, and creation of the file labels.

3. Various initializations as necessary: reservation of buffers, completion of the DCB, reservation and completion of communication tables (IOBs) between the access methods and the I/O supervisor, etc. DCB parameter values from an existing file's HDR2 label are written into the DCB.

4. Validation and control of file identity, file expiration, file protection, and file sharing; checking for coherence among the DCB parameters, and consistency of these with the processing mode specified in M:OPEN and with the storage media.

5. Optional creation of a temporary copy of the file index on secondary storage (i.e., system disk) for improved access speed.

Syntax:

[label(s)] M:OPEN [*]dcb-adr, mode[, CID][, ICY]

where

dcb-adr is the address of the DCB to be opened.

mode specifies the desired processing mode, as follows:

- I Input mode (reading).
- O Output mode (writing).
- U Update mode (reading and modification).

CID specifies file identity checking (see Usage Rule 1).

ICY requests a system-disk copy of the file index (see Usage Rule 3).

USAGE RULES

1. The file identity checking and positioning option NID and PID (see ASAM) are meaningful for magnetic tape files only. For AIAM (disk/RAD files only) this field is effectively ignored. CID is always assumed.

2. The effect of a given processing mode specification is conditioned by the declared status of the file versus the file's actual status. These interrelationships are described below.

3. If the access speed of the system disk device is significantly faster than that of the device on which the private volume is based, the ICY option will generally result in an improved processing rate.

RELATIONSHIP OF PROCESSING MODE AND FILE STATUS

The status of a file (NEW, MOD, or OLD) is declared in the file assignment. For each declared file status there is one or several normal combinations of status, file existence/nonexistence, and processing mode. The normal combinations do not occasion any abnormal return to the user's program during opening. Certain abnormal combinations (described below) are acceptable but will result in an X1 class abnormal return to the user's program during opening if the DCB specifies an abnormal return with the X1 class set (abort otherwise). Any combination other than those described below will result in a program abort.

The several normal and abnormal combinations are described for each declared file status in the following paragraphs.

- NEW

  Normal combination. Nonexistent file and output mode – the usual case for original file creation; output operations allowed only.

  Abnormal combination. Existent file and output mode – in this case an abnormal return to the user's program occurs. The user can then request recreation of the file by a special exit from his abnormal routine. If the expiration date of the file to be rewritten has not been reached, a further abnormal return occurs; another special exit will allow processing to continue, overriding the nonexpired condition.

- MOD

  Normal combinations.

  1. Existent file and update mode – input and updating operations allowed.
  2. Existent file and output mode – file extension output allowed only.
  3. Existent file and input mode – input operations allowed only.
  4. Nonexistent file and output mode – status is changed to NEW; output operations are allowed.

- OLD

  Normal combinations.

  1. Existent file and input mode – input operations are allowed.
  2. Existent, expired file and output mode – output operations allowed (i.e., the file may be rewritten).

Abnormal combination. Existent, nonexpired file – in this case an abnormal return to the user's program occurs. A special exit from the user's abnormal routine allows processing to continue. The file can then be rewritten.

Note: See the section "Processing of Abnormal and Error Conditions" later in this chapter for specific abnormal codes and further details on normal/special exit results.

AUTOMATIC EVOLUTION OF FILE STATUS

The automatic evolution of the status of a file during a job-step when opened for output is shown below.

	Declared Status Prior to First M:OPEN, Output		
	NEW, Nonexistent File	MOD, Nonexistent File	OLD, Existent File
Status after successful opening	NEW	NEW	NEW
Status after M:CLOSE	MOD	MOD	MOD

M:CLOSE Closing a File

This procedure causes the closing of a file. The close effects a suspension or halt of activity via a DCB and possibly via the processed file. It main functions are:

1. Waiting for and testing the last I/O operation(s).
2. Writing current (last) buffer where necessary.
3. Updating and validating disk file labels.
4. Temporary or definite closing of the DCB.
5. Partial restitution of disk space or deletion of the file created by the job.
6. Releasing devices, buffers, tables, etc.

Syntax:

$$[\text{label(s)}]\ \text{M:CLOSE}\ [*]\text{dcb-adr}\left[,\left\{\begin{matrix}\text{HLD}\\\text{MTN}\\\text{RLS}\end{matrix}\right\}\right]\left[,\left\{\begin{matrix}\text{KEP}\\\text{NCG}\end{matrix}\right\}\right]$$

where dcb-adr is the address of the DCB to be closed.

TYPES OF CLOSE

HLD specifies a temporary closing of the file. The DCB-file link is maintained. A new M:OPEN applied to the DCB will only permit changing of the processing mode of the file, because only a small portion of the open functions will then be executed.

RLS specifies a definite close. It destroys the DCB-file link and the job-file link. The device resource(s) allocated for the file are released if the FRE option was specified in the assignment.

MTN specifies a definite close. It cancels the DCB-file link but maintains the job-file link. In the same job another DCB can be used to process the same file. The resource is not released even if the FRE option was specified in the assignment.

LABEL VALIDATION AND FILE DISPOSITION

The KEP and NCG options specify the disposition of a newly created permanent file. They are significant only for the O processing mode (file creation).

KEP requests validation of the file labels, and cataloging of the file if CTG was specified in the assignment (or the file was created on the account volume).

NCG requests deletion of the file and suppression of cataloging if cataloging has been requested or implied. This option is applicable only in conjunction with RLS and if the status of the file is NEW prior to closing.

USAGE RULES

1. If no type-of-close option is specified, RLS (release) is assumed by default. Both the DCB-file connection and the job-file connection are severed (i.e., another job can attempt access to the file whether the file/volume is sharable or not). All device (i.e., common) resources allocated for the processing of this file are released if MTN was not specified in the assignment.
2. If no validation/disposition option is specified, KEP (keep) is assumed by default.
3. If a definite close (MTN or RLS) is not issued for a given DCB/file before the end of the job-step, the system forces an unconditional (RLS) close, disposition KEP.

M:GET Get Logical Record

The M:GET procedure permits reading of a record having a specified key value, or of the next logical record of an existing file relative to the last record accessed. The first record read must be accessed by its key value. The records will either be moved to an area specified by the procedure, or be located (i.e., pointed to in the I/O buffer), at the user's option (MOV/LOC mode parameter of the DCB).

Syntax: See foldout Chart A-4 in Appendix A.

MEANING OF THE PARAMETERS

dcb-adr specifies the address of the DCB through which records are to be read.

REC, {[*]badr-1 / adr-2} specifies the byte address (badr-1) of the user's receiving area in MOV mode, or the address (adr-2) of a word into which a byte address pointer to the accessed record is to be stored by the system in LOC mode.

RSA, [*]adr-3 optionally specifies the address (adr-3) of a word into which the byte length of the accessed record is to be stored (by the system).

KEY, {[*]badr-4 / adr-5} optionally specifies the byte address (badr-4) of a byte string containing the key value of the record to be read, or the direct word address (adr-5) of such a string (will be converted to byte address).

USAGE RULES

1. If MOV mode is in effect (in the referenced DCB) a byte address (optionally indirect) must be specified for the REC parameter. If LOC mode is in effect, a word address (direct only) must be specified.

2. If the RSA parameter is specified, the length of the record moved or located is reported for all record formats. For V format records, the length of the record body only is reported.

3. For V format records, the record body only is moved or located (i.e., the four-byte record header is not included).

4. The KEY parameter must be specified in the first execution of M:GET following opening of the file.

5. If the KEY parameter is not specified in any execution of M:GET subsequent to the first, the next sequential record having the next higher key value relative to the last record accessed will be read.

6. The length of the key string is fixed by the KYL parameter of the DCB.

7. M:GET is not valid in output processing mode.

8. A minimum of one I/O buffer is required, in either MOV or LOC mode, for input processing; a minimum of two is required for update processing.

M:PUT Put Logical Record

The M:PUT procedure permits writing of a key-ordered sequence of logical records of a file being created, rewritten, or extended; or, in update processing mode, rewriting of the logical record last accessed via M:GET or insertion of a new record. This procedure operates only in MOV mode (with record movement).

Syntax: See foldout Chart A-4 in Appendix A.

MEANING OF THE PARAMETERS

dcb-adr specifies the address of the DCB through which records are to be written.

REC, [*]badr-1 specifies the byte address (badr-1) of the user's sending area (from which the record is to be moved).

ARS, [*]value specifies, for output or update processing, the length of the V format record to be written.

NWK specifies, for update processing mode, that the record to be written is a new insertion record (rather than a replacement for the last record accessed).

DFW specifies, for update processing mode, that the block being processed is not to be written to the file until the next I/O operation implying a block change and/or an M:PUT without DFW is executed.

USAGE RULES

1. MOV mode is always in effect for M:PUT (regardless of the setting of MOV/LOC in the referenced DCB). A byte address (optionally indirect) must be specified for the REC parameter. The MOV/LOC parameter is significant only for M:GET.

2. If NWK (New Key) is specified in update mode, the record to be written must have a key value not already present in the file; the record will be inserted into the file in its proper key-value position.

3. The ARS parameter is effective both in output and update processing mode, and only for V format records. If specified in update processing, it indicates a length change for record modification (NWK not specified) or the length of an insertion record (NWK specified).

4. For F format records, the effective record length is taken from the DCB (REL parameter).

5. In update processing mode, unless NWK is specified the record to be written replaces the last record accessed with the M:GET procedure; the length of the latter record automatically determines the length of the record to be written if ARS is not specified (in V format).

6. The REC parameter specifies the location of the initial byte of the record that is to be moved to the I/O buffer. In V format the record body only is to be supplied by the user (the system affixes the record header).

7. The ARS parameter must be specified in output processing mode for V format.

8. Following an opening in output mode of an existent file whose declared or evolved status is NEW, M:PUT executions cause the existing file to be overwritten, (i.e., recreated).

9. Following an opening in output mode of an existing file whose declared or evolved status is MOD, M:PUT executions cause the file to be extended (i.e., only records having key values higher than those already present in the file may be written).

10. Each M:PUT in update mode implies a physical write operation unless the DFW option is employed.

M:TRUNC Truncation of a Block

The M:TRUNC procedure permits termination of sequential operations on a partially processed block and passage to the next sequential block for subsequent processing. In output processing mode the current block is truncated (i.e., the partially completed block is written to the physical file). This allows space for subsequent insertion of records without immediate creation of overflow blocks.

In input or update processing mode, the record accessed by the first sequential M:GET (i.e., no key specified) following an M:TRUNC execution is the first record of the next sequential block. In update mode, the complete current block (with modifications, if any) is rewritten to the file.

This procedure is in general meaningful only for files containing blocked records (i.e., more than one record per block).

Syntax:

[label(s)] M:TRUNC [*]dcb-adr

where dcb-adr is the address of the DCB through which processing is being conducted.

USAGE RULES

1. If no record has yet been read from or written to the current block (i.e., the current I/O buffer), M:TRUNC has no effect.

2. M:TRUNC does not cause actual truncation (i.e., shortening) of the current buffer in update processing mode.

M:DELREC Delete a Record

The M:DELREC procedure permits deletion of the last logical record accessed with M:GET in update processing mode when the DLC parameter is in effect in the DCB. Execution of this procedure causes the value X'FF' to be placed in the first byte of the record body, the first byte being defined as a deletion control character. This effectively deletes that record for any assisted access method.

Syntax:

[label(s)] M:DELREC [*]dcb-adr

where dcb-adr is the address of the DCB.

USAGE RULES

1. To be effective, the DLC parameter, specifying existence of the deletion control character, must be specified at file creation time (see M:DCB). On subsequent file accesses the value carried in the HDR2 label of the file overrides the setting in the DCB at open time.

2. M:DELREC is applicable only in update processing mode. It is applicable to both F and V format records.

APAM (ASSISTED PARTITIONED ACCESS METHOD)

The assisted partitioned access method (APAM) creates and processes partitioned-organization files. These are essentially sequential files subdivided into individually accessible segments called partitions. (See the section "File Organization" in Chapter 6 for a detailed description of the partitioned file organization.) A file partition is identified by one or more user assigned partition names, or keys. The initially assigned key is called the principal partition key; subsequently assigned keys (for the same partition) are called synonyms.

The procedures M:STOW and M:FIND, unique to APAM, are used to name and access an individual partition during file creation, reading, and updating, as applicable. Partitions may be added to or deleted from existing files in output processing mode. For record processing, all procedures applicable to disk/RAD files under ASAM, except M:CVOL, are also applicable under APAM. Except for direct accessing of partitions (i.e., positioning by key to the beginning of a partition) file processing is essentially sequential, both within a partition and across subsequent partition boundaries.

Partitioned files are particularly useful for program and subroutine libraries.

See foldout Chart A-5 of Appendix A for the syntax of the APAM I/O procedures in reference form.

GENERAL USAGE RULES

The general usage rules for APAM are listed below.

1. Creates and processes partitioned-organization (direct access) files only. Multivolume files must be parallel mounted.

2. Applicable to direct access (disk/RAD) storage media only.

3. Not applicable to nonstandard (DEV assigned) volumes.

4. Applicable processing modes:

   Output
   Input
   Update

5. Applicable to F or V format records only.

6. Applicable I/O processing procedures:

M:GET	M:TRUNC
M:PUT	M:DELREC
M:FIND	M:NOTE
M:STOW	M:POINT

7. Usage of either M:DCB or M:MOVEDCB and of M:OPEN and M:CLOSE is mandatory for a given DCB/file; usage of M:SETDCB and M:ASSIGN is optional.

M:DCB Assembly-Time DCB Creation

Syntax:

[label(s)] M:DCB (OPL, 'op-label')[, dcb-param, . . .]

Refer to foldout Chart A-2 in Appendix A for the general syntax of M:DCB, in reference form, showing the syntax of all DCB parameters. The parameters and values applicable under APAM are described below, grouped by type of parameter.

FILE PROCESSING PARAMETERS

(ABN, address[, class-code, . . .]) – Abnormal Return Parameter. Specifies the address of the user's routine to which control is to be returned if certain abnormal conditions occur during file opening, processing, and closing. Also specifies the class(es) of conditions for which control is to be returned. None, some, or all of the following abnormal class codes can be specified:

Code	Class
X1	File opening abnormalities.
X2	End-of-file/volume abnormalities.
X3	File processing abnormalities.

The address field is mandatory if this parameter is specified. If no class code is specified, no abnormal conditions are returned for user routine handling (see M:SETDCB description for special usage. The section "Processing of Abnormal and Error Conditions" later in this chapter describes all abnormal conditions and corresponding codes in detail, and provides information on the type of processing allowed in user's abnormal- and error-handling routines. Occurrence of an abnormal condition that has not been selected, by class, for user routine handling will cause a program abort.

(AM, AP) – Access Method Parameter. The value AP specifies usage of APAM for I/O processing via this DCB. This parameter must be specified.

(BFA, byte-address) – Buffer Address Parameter. Specifies the byte address of the first of a set of user reserved I/O buffer areas, if any. If more than one such buffer exists (BFA $\neq$ 0 and NBF > 1), the set is assumed to be contiguous. The length of each buffer is assumed to equal the block length (BKL parameter). If this parameter is omitted (or the address specified is zero) at opening time, the system will acquire buffer space in an amount related to the BKL and NBF values in force at that time. (See "Buffer Usage" for further details.)

(ERR[, address]) – Error Return Parameter. Specifies the address of the user's routine to which control is to be returned if any error condition occurs during processing via this DCB. If this parameter is omitted or the address is omitted, the occurrence of any error condition will cause a program abort. See the section "Processing of Abnormal and Error Conditions" later in this chapter for a detailed description of the possible error conditions and corresponding codes.

(MOD, EBC) – Data Mode Parameter. Specifies the mode of data encoding to be expected on input or produced on output. The mode keyword EBC specifies EBCDIC encoding, the only mode applicable. The default mode is EBC.

(NBF, value) – Number of Buffers Parameter. Specifies the number of I/O buffers (either user- or system-provided) to be used by the M:GET/M:PUT procedures. The default value is 2.

(NRT) – No Retry Parameter. This parameter specifies that the system is not to execute the standard error recovery procedure (i.e., repeated retries of the same I/O operation) in case of a device or transmission error.

(ORG, P) – File Organization Parameter. Specifies the organization (partitioned) of the file to be created or accessed. This parameter must be specified.

BLOCK LEVEL PARAMETERS

(BHR, value) – Block Header Parameter. Specifies the size of the data block header. For disk/RAD files only the value 4 (or 0) may be specified (see "Block Formats" in Chapter 6). The default value is 4.

(BKL, value) – Block Length Parameter. Specifies the block length, in bytes. The length specified must include the block header. The default value, by media type, is 1024 for disk/RAD files. A block length value less than or greater than the default value may be specified (up to a maximum of 32K-1 bytes). See "Block Formats" in Chapter 6 for a discussion of block length. Note that a four-byte block header is included in each P organization data block.

RECORD LEVEL PARAMETERS

(DLC) – Deletion Control Parameter. Presence of this keyword parameter specifies that the first byte of the record body (i.e., the user data) is to function as a deletion control character. It is applicable to disk files only. This parameter is effective only when specified at file creation; an existing file's HDR2 label value for DLC will override in all other cases. See M:DELREC procedure for description of DLC character function.

(FRM, F | V) – Record Format Parameter. Specifies the record format of the file to be created or accessed, fixed or variable. The default is F (fixed format). When accessing a labeled file and F or V is specified, the actual record format (as described in the file's HDR2 label) will override the format specified.

(KYL, value) – Key Length Parameter. Specifies the length of the partition key, in bytes. The length value may range from 1 through 255.

(LOC | MOV) – Locate/Move Mode Parameter. This alternative-keyword parameter specifies that the records are to be accessed either in locate mode (LOC), i.e., no record movement, or in move mode (MOV) (see "Buffer Usage" for details). MOV is the default.

Note: M:PUT does not operate in LOC mode when the file is opened for update (U) processing.

(REL, value) – Record Length Parameter. Specifies the length of the records to be processed, in bytes (the actual length for F format records or maximum length for V format). In V format, the length specified is that of the maximum size record body, excluding record header bytes. The default REL value is calculated (at open time) as follows:

F format – REL = BKL - BHR.

V format – REL = BKL - BHR-4.

Note that if the REL parameter is not specified for F format records, unblocked records (i.e., one record per block) are implied.

USAGE RULES

1. M:DCB is not an executable procedure. DCB space reservation and explicit parameter setting is performed at assembly time.

2. None of the parameters are indirectly addressable.

M:MOVEDCB Dynamic DCB Creation/Retention

The M:MOVEDCB procedure causes the allocation, during program execution, of a 19-word DCB area with read-only protection in the common-dynamic portion of the user's virtual memory space; the address of this area is returned to the user's program. It also causes the contents of a user specified sending area to be copied into the newly allocated DCB area. For example, the sending area may be a nonprotected area of the user's program where a DCB image or "skeleton" has been built, or a write protected DCB that is not open when the M:MOVEDCB is executed. Effectively, this procedure allows the user to create DCBs during program execution, and/or to move and thereby save inactive DCBs that could otherwise be overlaid or destroyed.

Syntax:

[label(s)] M:MOVEDCB [*]adr-1, (PTR, adr-2)

where

adr-1 is the word address of a 19-word sending area which is to be copied into the newly allocated DCB space.

adr-2 is the address of a one-word pointer in the user's program in which the address of the newly allocated DCB is to be returned.

USAGE RULES

1. No default values are supplied for any parameter field by M:MOVEDCB.

2. The effect of M:SETDCB, M:ASSIGN, M:OPEN, and M:CLOSE procedures referencing a dynamically created or saved DCB is identical to the effect of the same procedures on an assembled DCB.

M:SETDCB Execution-Time DCB Modification

The M:SETDCB procedure allows the user, during program execution, to (1) "fill in" empty parameter fields in a nonopen DCB, i.e., to specify previously unspecified DCB parameters, (2) modify previously specified or defaulted parameters in a nonopen DCB, and (3) modify the ABN and ERR parameters in an open DCB. The effective parameters in cases 1 and 2 are all of the parameters that may be specified in the M:DCB procedure for APAM. See foldout Chart A-2 of Appendix A for the complete general syntax

of M:SETDCB in reference form. The parameters applicable under APAM are as described for the M:DCB procedure.

M:SETDCB also allows modification of the operational label associated with a DCB.

Syntax:

[label(s)] M:SETDCB [*]dcb-adr[,(OPL, {*adr-label / 'op-label'})] [,dcb-parameter,...]

where

dcb-adr is the address of the DCB to be modified.

adr-label is the address of a word containing an operational label, left-justified and space filled, to be associated with the DCB being modified.

'op-label' is a one- to four-character constant specifying an operational label to be associated with the DCB being modified.

dcb-parameter is a DCB parameter as described previously for the M:DCB procedure.

USAGE RULES

1. All of the DCB-parameter addresses and values can be indirectly addressed.

2. If the OPL option appears, the operational label specified therein will replace the operational label (if any) previously specified in the referenced DCB.

3. If the referenced DCB is open at the time of M:SETDCB execution, only the ABN and/or ERR parameter will be effective; any other parameters will be ignored.

4. A DCB closed with the HLD option (temporary close) is considered as open for the purposes of rule 3.

5. If the ABN parameter appears, the set of abnormal class codes specified will replace any such set previously in effect, including the null case where no class code is specified (i.e., any class code previously in effect can be "turned off" by not specifying any class code).

M:ASSIGN Execution-Time DCB Assignment

The M:ASSIGN procedure allows the user, during program execution, to (1) define a temporary file in secondary storage and assign an operational label to it, or (2) define a permanent file on a physical resource specified by a !ASSIGN command. Effectively, case 1 allows the user to eliminate from the job control deck !ASSIGN commands for invariant "scratch file" assignments, or to define additional temporary files to satisfy dynamically determined program requirements (as M:MOVEDCB can be used to create additional DCBs). Case 2 allows the user to make an execution time choice between several !ASSIGN commands, each specifying a different resource, or to redefine a file assignment (status file name or space allocation) made by a !ASSIGN command. In case 2, the !ASSIGN command referred to must itself define a permanent, i.e., named, file.

Syntax:

Format 1 for temporary files

[label(s)] M:ASSIGN (OPL, {*adr-1 / 'op-label'}) [,(SIZ, {*adr-2 / size}, {*adr-3 / increment})]

where

adr-1 is the address of a word containing the operational label that identifies the DCB being assigned.

'op-label' is the character constant form of that label.

adr-2 is the address of a word containing a value specifying the size of the data block portion of the file.

size is the constant form of that size value.

adr-3 is the address of a word containing a value specifying the size of the index/overflow portion of the file.

increment is the constant form of that increment value.

Format 2, for permanent files

[label(s)] M:ASSIGN (OPL, {*adr-1 / 'op-label'}),
(UNT, OPL, {*adr-4 / 'op-label-1'}),
(NAM, {*adr-5 / 'filename'}),(STS, {NEW / OLD / MOD})
[,(SIZ, {*adr-2 / size}, {*adr-3 / increment})]

where

adr-1, 'op-label', adr-2, size, adr-3, and increment have the same meanings as in format 1.

adr-4 is the address of a word containing the operational label that identifies a !ASSIGN command specifying the desired physical resource.

'op-label-1' is the character constant form of the operational label described immediately above.

adr-5 is the word (or byte) address of a TEXT-format field containing the name of the file to be created or accessed. The name must be terminated with one or more blanks. (See the TEXT directive, Meta-Symbol Reference Manual, 90 09 52.)

'filename' is the character constant form of the file name.

USAGE RULES

1. Assignments made via M:ASSIGN are valid only for the job-step in which the procedure is executed, they cannot be maintained over succeeding steps. (Effectively the !ASSIGN option FRE is implicit in the use of an M:ASSIGN procedure.)

2. The usage of M:ASSIGN in format 1 is identical to the usage of a !ASSIGN command with corresponding options specified.

3. In format 2, the UNT, OPL option implies use of the resource (device and volume) defined and allocated to the job-step by the operational label specified.

4. The !ASSIGN command referred to by the UNT, OPL option in format 2 must specify a permanent, i.e., named, file assignment on direct access media and must be in force during the job-step in which the M:ASSIGN is executed.

5. In format 2, all options other than UNT have the same meaning and usage as the corresponding options appearing in a !ASSIGN command, assuming the option FIL.

6. The M:ASSIGN procedure in format 1 may redefine an implicit assignment via a (installation dependent) predefined operational label, typically SI, LO, EI, etc.

7. In all cases, the operational label specified by the OPL keyword field establishes the link between a DCB and the file named by the procedure.

8. In format 2, the DCB parameters (if any) specified in the reference !ASSIGN command are not applied to the DCB assigned via the procedure.

M:OPEN Opening a File

The M:OPEN procedure activates the link between a DCB and a physical file. The DCB is placed in an active, or open, status. ("Opening a file" is equivalent to "opening a DCB".) This procedure must be successfully executed before any I/O operation can be performed on a given file. The M:OPEN procedure performs some or all of the following steps while opening a file:

1. Location of the file to be processed, if it exists.

2. Creation of the file if it does not exist, which implies allocation of resources: volume and space on the volume, and creation of the file labels.

3. Various initializations as necessary: reservation of buffers, completion of the DCB, reservation and completion of communication tables (IOBs) between access methods and the I/O supervisor, etc. DCB parameter values from an existing file's HDR2 label are written into the DCB.

4. Validation and control of file identity, file expiration, file protection, and file sharing; checking for coherence among the DCB parameters, and consistency of these with the processing mode specified in M:OPEN and with the storage media.

Syntax:

[label(s)] M:OPEN [*]dcb-adr, mode[, CID]

where

dcb-adr is the address of the DCB to be opened.

mode specifies the desired processing mode, as follows:

I Input mode (reading).

O Output mode (writing).

U Update mode (reading and modification).

CID specifies file identity checking (see Usage Rule 1).

USAGE RULES

1. The file identity checking and positioning option NID and PID (see ASAM) are meaningful for magnetic tape files only. For disk/RAD files this field is effectively ignored. CID is always assumed.

2. The effect of a given processing mode specification is conditioned by the declared status of the file versus the file's actual status. These interrelationships are described below.

RELATIONSHIP OF PROCESSING MODE AND FILE STATUS

The status of a file (NEW, MOD, or OLD) is declared in the file assignment. For each declared file status there is one or several normal combinations of status, file existence/nonexistence, and processing mode. The normal combinations

do not occasion any abnormal return to the user's program during opening. Certain abnormal combinations (described below) are acceptable but will result in an X1 class abnormal return to the user's program during opening if the DCB specifies an abnormal return with the X1 class set (abort otherwise). Any combinations other than those described below will result in a program abort.

The several normal and abnormal combinations are described for each declared file status in the following paragraphs.

- NEW

  Normal combination. Nonexistent file and output mode – the usual case for original file creation; output operations allowed only.

  Abnormal combination. Existent file and output mode – in this case an abnormal return to the user's program occurs. The user can then request recreation of the file by a special exit from his abnormal routine. If the expiration date of the file to be rewritten has not been reached, a further abnormal return occurs; another special exit will allow processing to continue, overriding the nonexpired condition.

- MOD

  Normal combinations.

  1. Existent file and update mode – input and updating operations allowed.
  2. Existent file and output mode – file extension output allowed only.
  3. Existent file and input mode – input operations allowed only.
  4. Nonexistent file and output mode – status is changed to NEW; output operations are allowed.

- OLD

  Normal combinations.

  1. Existent file and input mode – input operations are allowed.
  2. Existent, expired file and output mode – output operations allowed (i.e., the file may be rewritten).

  Abnormal combination. Existent, nonexpired file – in this case an abnormal return to the user's program occurs. A special exit from the user's abnormal routine allows processing to continue; the file can then be rewritten.

Note: See the section "Processing of Abnormal and Error Conditions" later in this chapter for specific abnormal codes and further details on normal/special exit results.

AUTOMATIC EVOLUTION OF FILE STATUS

The automatic evolution of the status of a file during a job-step when opened for output is shown below.

	Declared Status Prior to First M:OPEN, Output		
	NEW, Nonexistent File	MOD, Nonexistent File	OLD, Existent File
Status after successful opening	NEW	NEW	NEW
Status after M:CLOSE	MOD	MOD	MOD

M:CLOSE Closing a File

This procedure causes the closing of a file. The close effects a suspension or halt of activity via a DCB and possibly via the processed file. Its main functions are:

1. Waiting for and testing the last I/O operation(s).
2. Writing current (last) buffer, where necessary.
3. Updating and validating disk file labels.
4. Temporary or definite closing of the DCB.
5. Partial restitution of disk space or deletion of the file created by the job.
6. Releasing devices, buffers, tables, etc.

Syntax:

$$[\text{label(s)}]\ \text{M:CLOSE}\ [*]\text{dcb-adr}\left[,\left\{\begin{matrix}\text{HLD}\\ \text{MTN}\\ \text{RLS}\end{matrix}\right\}\right]\left[,\left\{\begin{matrix}\text{KEP}\\ \text{NCG}\end{matrix}\right\}\right]$$

where dcb-adr is the address of DCB to be closed.

TYPES OF CLOSE

HLD specifies a temporary closing of the file. The DCB-file link is maintained. A new M:OPEN applied to the DCB will only permit changing of the processing mode of the file, because only a small portion of the open functions will then be executed.

MTN specifies a definite close. It cancels the DCB-file link but maintains the job-file link. In the same job another DCB can be used to process the same file. The resource is not released even if the FRE option was specified in the assignment.

RLS specifies a definite close. It cancels the DCB-file link and the job-file link. The device resource(s) allocated for the file are released if the FRE option was specified in the assignment.

LABEL VALIDATION AND FILE DISPOSITION

The KEP and NCG options specify the disposition of a newly created permanent file. They are significant only for the O processing mode (file creation).

KEP requests validation of the file labels and cataloging of the file when requested or implied.

NCG requests deletion of the file suppression of cataloging if cataloging has been requested or implied. This option is applicable only in conjunction with RLS and if the status of the file is NEW prior to closing.

USAGE RULES

1. If no type-of-close option is specified, RLS (release) is assumed by default. Both the DCB-file connection and the job-file connection are severed (i.e., another job can attempt access to the file whether the file/volume is sharable or not). All device resources allocated for the processing of this file are released if MTN was not specified in the assignment.

2. If no validation/disposition option is specified, KEP (keep) is assumed by default.

3. If a definite close (MTN or RLS) is not issued for a given DCB/file before the end of the job step, the system forces an unconditional (RLS) close, disposition KEP.

M:STOW Create/Delete Partition Key

The M:STOW procedure stores principal and synonym partition keys into the directory associated with a partitioned file or deletes such keys from the directory during output mode processing of a new or existing file. When a principal partition key is created (ADD option), positioning to the end of the existing file (if any) also occurs.

Synonym partition keys may be created (SYN option) following creation of the principal partition key (i.e., any time during creation of the partition). Either principal and/or synonym keys may be deleted at any time during output mode processing.

Effectively, this procedure allows the user to establish partition boundaries during sequential file creation (status NEW or OLD, output mode) or to add new partitions, at the current end-of-file, during file extension (status MOD, output mode). Each partition boundary is associated with one or more user chosen names. During either file creation or extension, such partition boundaries can also be deleted, by name. (The records previously constituting the "deleted" partition are not thereby deleted, however. A subsequent sequential reading of the file across the deleted boundary will access those records.

Note that (as discussed in Chapter 6 under "File Organization") a partition boundary, indicating the beginning of a given partition, does not also delimit the previous partition. That is, the system does not maintain or recognize any end-of-partition boundaries for accessing of the file. (A partitioned file may therefore be given an hierarchical structure.)

Syntax:

[label(s)] M:STOW [*]dcb-adr,(KEY,

$$\left\{\begin{matrix}[*]badr\text{-}1 \\ adr\text{-}2\end{matrix}\right\}\left[,\left(\left\{\begin{matrix}ADD \\ SYN \\ DEL\end{matrix}\right\}\right)\right]$$

where

dcb-adr is the address of the DCB.

badr-1 is the byte address (optionally indirect) of the initial byte of the partition key to be added to or deleted from the directory.

adr-2 is the word address (direct only) of the initial byte of the partition key (will be converted to a byte address).

ADD specifies creation of a principal partition key (default option).

SYN specifies creation of a synonym partition key.

DEL specifies deletion of the indicated partition key, principal or synonym.

USAGE RULES

1. Applicable only to output mode processing of either new or existing files.

2. All partition keys created must be unique within the file.

3. The length of a partition key is defined by the DCB parameter KYL (255 maximum). On each M:STOW execution, the system will read KYL bytes starting at the address badr-1 or adr-2 specified for the KEY parameter.

4. Partition keys are sorted and stored in the directory in ascending binary order. (Note that this provides a normal collating sequence for the EBCDIC alphanumeric characters, with the space and special graphic characters sorting lower than the alphabetics, and the numeric characters sorting higher.)

5. At least one M:STOW (ADD) must be executed, during either file creation or extension, prior to the first execution of M:PUT.

6. An M:STOW... (SYN) execution adds a synonym key to the partition established or referred to by the last effective M:STOW (ADD) procedure executed (since opening the file. That is, the synonym key refers to the last principal key effectively processed.

7. M:PUT and M:TRUNC procedure executions may intervene between a given M:STOW (SYN) execution and the previous M:STOW (ADD) execution to which it refers.

8. M:STOW (ADD) causes repositioning, if necessary, to the currently effective end-of-file for subsequent M:PUT executions.

9. M:STOW (DEL), referring to any partition in the file, can be executed at any time regardless of current positioning.

10. Neither M:STOW (SYN) nor M:STOW (DEL) cause repositioning to a new partition boundary.

11. During file creation, the creation of keys in an ascending binary order sequence results in appreciably faster processing of the file.

M:FIND Find a Partition Boundary

The M:FIND procedure permits, during input or update processing of a partitioned file, positioning to a partition boundary selected by either principal or synonym key. A subsequent M:GET procedure will read the first record of the partition so selected. Optionally, the procedure can also return to the user's program a pointer to the first record of the partition to which positioning is effected. This pointer can be utilized subsequently e.g., in an M:POINT procedure execution.

Effectively, the M:FIND procedure allows the user to directly access a partition of an existing file.

Format:

[label(s)] M:FIND [*]dcb-adr,(KEY,{[*]badr-1 / adr-2})

[,(RCI,[*]adr-3)]

where

dcb-adr is the address of the DCB.

badr-1 is the byte address (optionally indirect) of the initial byte of the partition key to be used for positioning.

adr-2 is the word address (direct only) of the initial byte of the partition key (it will be converted to a byte address).

adr-3 is the address of a two-word area in the user's program in which the first-record pointer is to be returned.

USAGE RULES

1. The address specified in the RCI parameter must reference, directly or indirectly, the first of two contiguous words into which the system will store a pointer to the first record of the selected partition.

2. On return, word adr-2 contains a relative block number and word adr-2 + 1 contains the byte displacement of the first record of the partition selected, relative to the beginning of the block.

3. Applicable to input and update processing modes only.

M:GET Get Next Sequential Record

The M:GET procedure permits reading of the next logical record of an existing file, relative to the record last accessed; the first record of a partition previously selected via M:FIND; or the first sequential record of the file. (Exception: see M:POINT procedure.) The records will either be moved to an area specified by the procedure, or be located (i.e., pointed to in the I/O buffer) at the user's option (MOV/LOC mode parameter of the DCB).

Syntax: See foldout Chart A-5 in Appendix A.

MEANING OF THE PARAMETERS

dcb-adr specifies the address of the DCB through which records are to be read.

REC,{[*]badr-1 / adr-2} specifies the byte address (badr-1) of the user's receiving area in MOV mode, or the address (adr-2) of a word into which a byte address pointer to the accessed record is to be stored by the system in LOC mode.

RSA,[*]adr-3 optionally specifies the address (adr-3) of a word in which the byte length of the accessed record is to be stored (by the system).

USAGE RULES

1. If MOV mode is in effect (in the referenced DCB) a byte address (optionally indirect) must be specified for the REC parameter. If LOC mode is in effect, a word address (direct only) must be specified.

2. If the RSA parameter is specified, the length of the logical record moved or located is reported for both record formats. For V format records, the length of the record body only is reported.

3. For V format records, the record body only is moved or located (i.e., the four-byte record header is not included).

4. The first M:GET executed after an execution of M:FIND reads the first record of the partition selected by the M:FIND.

5. If no prior M:GET or M:FIND procedure has been executed since opening of the file, M:GET will read the first record (of the first partition) of the file.

6. M:GET is not valid in output processing mode.

M:PUT Put Next Sequential Record

The M:PUT procedure permits writing of the next logical record of a file being created or rewritten or, in update processing mode, rewriting of the logical record last accessed via M:GET. This procedure operates in either MOV mode (with record movement) or LOC mode (no record movement), except in update processing where MOV mode is implicit.

Syntax: See foldout Chart A-5 in Appendix A.

MEANING OF THE PARAMETERS

dcb-adr specifies the address of the DCB through which records are to be written.

REC, $\left\{\begin{matrix}[*]\text{badr-1}\\ \text{adr-2}\end{matrix}\right\}$ specifies the byte address (badr-1) of the user's sending area in MOV mode, or the address (adr-2) of a word into which a pointer to a record position (in the I/O buffer) is to be stored by the system in LOC mode.

ARS, [*]value specifies, for output processing only, the length of the V format record to be written.

USAGE RULES

1. If MOV mode is in effect (in the referenced DCB), a byte address must be specified, directly or indirectly, for the REC parameter. If LOC is in effect, a word address (direct only) must be specified.

2. M:PUT operates only in MOV mode when the DCB is open for update processing. The MOV/LOC parameter in this case is significant only for M:GET.

3. The ARS option is effective only in output processing mode, and only for V format records; it is ignored in update processing and for F format records.

4. For F format records, the effective record length is taken from the DCB (REL parameter).

5. In update processing mode, the record to be written replaces the last record accessed with the M:GET procedure; the length of the latter record automatically determines the length of the record to be written.

6. In MOV mode, the REC parameter specifies the initial byte location of the record that is to be moved to the I/O buffer. In V format the record body only is to be supplied by the user (the system affixes the record header).

7. In LOC mode, the REC parameter specifies the location of a word in the user's program in which the system stores a byte-address pointer to the beginning of the next available record space in the I/O buffer where the user may build his record.

8. In LOC mode, the ARS parameter (when effective) specifies the length of the record to be constructed at the point indicated by the pointer returned (at adr-2) by the M:PUT execution.

9. In LOC mode, any execution of M:PUT returns a pointer to the origin in the buffer for the record to be built (subsequently) by the user. It establishes, explicitly or implicitly, the length of that record and causes it to be "written". That is, n executions of M:PUT in LOC mode causes n records to be written.

10. In output processing mode, the first M:PUT execution after opening the file must be preceded by an M:STOW procedure.

11. Following an opening in output mode of an existing file whose declared or evolved status is NEW, M:STOW and M:PUT executions cause the existing file to be overwritten (i.e., the existing file is recreated).

M:TRUNC Trunction of a Block

The M:TRUNC procedure permits termination of sequential operations on a partially processed block and passage to the next sequential block for subsequent processing. In output processing mode, only, the current block is truncated (i.e., the partially completed block is written to the physical file). In input or update processing mode, the record accessed by the first M:GET following an M:TRUNC execution is the first record of the next sequential block. In Update mode, the complete current block (with modifications, if any) is rewritten to the file.

This procedure is in general meaningful only for files containing blocked records (i.e., more than one record per block).

Syntax:

[label(s)] M:TRUNC [*]dcb-adr

where dcb-adr is the address of the DCB through which processing is being conducted.

USAGE RULES

1. M:TRUNC may immediately follow an M:STOW or M:FIND procedure.

2. If no record has yet been read from or written to the current block (i.e., the current I/O buffer), M:TRUNC has no effect.

3. M:TRUNC does not cause physical truncation of the current buffer in update processing mode.

M:DELREC Delete a Record

The M:DELREC procedure permits deletion of the last logical record accessed with M:GET in update processing mode when the DLC parameter is in effect in the DCB. Execution of this procedure causes the value X'FF' to be placed in the first byte of the record body, the first byte being defined as a deletion control character. This effectively deletes that record for access by any assisted access method.

Syntax:

[label(s)] M:DELREC [*]dcb-adr

where dcb-adr is the address of the DCB.

USAGE RULES

1. To be effective, the DLC parameter, specifying existence of the deletion control character, must be specified at file creation time (see M:DCB). On subsequent file accesses the value carried in the HDR2 label of the file overrides the setting in the DCB at open time.

2. M:DELREC is applicable only in update processing mode.

M:NOTE Note Current Position

The M:NOTE procedure permits the user to obtain a pointer to his current block/record position during input or update processing of a file. This pointer can then subsequently be used with the M:POINT procedure, described below, to reposition to the point at which the M:NOTE was issued.

Syntax:

[label(s)] M:NOTE [*]dcb-adr, (RCI, [*]adr-1)

where

dcb-adr is the address of the DCB.

adr-1 is the address of a two-word area in the user's program into which the current-position pointer is to be stored by the system.

USAGE RULES

1. The address specified in the RCI parameter must reference, directly or indirectly, the first of two contiguous words into which the system will store a pointer to the last record accessed with an M:GET procedure.

2. On return, word adr-1 contains the relative block number of the current block and word adr-1 + 1 contains the byte displacement of the current record relative to the beginning of the block.

3. This procedure is applicable to input and update processing modes only.

M:POINT Reposition by Pointer

The M:POINT procedure permits repositioning, within the file being processed, to a record or partition pointed to by information obtained via a previously issued M:NOTE or M:FIND. This repositioning can extend across block boundaries, forward or backward.

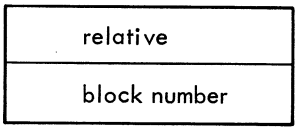
Syntax:

[label(s)] M:POINT [*]dcb-adr, (RCI, [*]adr-1)

where

dcb-adr is the address of the DCB.

adr-1 is the address of a two-word pointer containing repositioning information of the following format:



USAGE RULES

1. The pointer returned by the M:NOTE or M:FIND procedure may be used to supply the positioning information required by M:POINT.

2. If the pointer supplied to M:POINT refers to a position outside the limits of the file currently being processed, an X3 class abnormal return will occur. (If the X3 class is not set in the DCB, a program abort will result.)

3. This procedure is applicable to input and update processing modes only.

VSAM (VIRTUAL SEQUENTIAL ACCESS METHOD)

VSAM is the most generally applicable of the basic access methods in terms of range of storage media. It is the only basic method that is applicable to nonmagnetic device and magnetic tape files, as well as to FIL-assigned direct access

media. (ASAM is the only other access method applicable to the same range of media categories.) It creates files of the same sequential organization at the block level as ASAM, but no logical record structures are provided for or recognized. Undefined (U) record format is implicit in this access method.

GENERAL USAGE RULES

The general usage rules for VSAM are listed below.

1. Creates and processes sequential-organization files.
2. Applicable processing modes:

   Input.

   Output.

   Backward read (magnetic devices only).

   Scratch (reading and writing).

3. Applicable to all media categories.
4. Not applicable to nonstandard (DEV assigned) disk volumes.
5. Applicable (at the block level) to files created under ASAM and VDAM.
6. No I/O buffering is performed by the system. That is, physical transmission of data is directly to or from the user program area specified by the M:READ or M:WRITE procedure.
7. The amount of data transmitted (i.e., the length of the physical record) is user defined for each I/O operation (or defaults to BKL).
8. For magnetic tape files, physical records shorter or longer than the BKL value can be written (≤MXL); physical records shorter than the actual tape block can be read.
9. For disk/RAD files, physical records shorter or longer than the block size (BKL value) can be written and read. (Transmission length may not exceed the MXL value.)
10. Testing for successful completion of an I/O operation (or sequence of such operations) is not performed by VSAM. The user must himself check for I/O operation completions, successful or otherwise, with the M:CHECK procedure, possibly in conjunction with the M:WAIT procedure.
11. Abnormal and/or error returns (if selected) are made from the M:CHECK procedure only.
12. Requires that the user must himself check for end-of-volume condition and switch volumes, if necessary (M:CHECK and M:CVOL).
13. Applies to the following I/O processing procedures:

M:READ	M:NOTE
M:WRITE	M:POINT
M:CHECK	M:DEVICE
M:CVOL	

14. Usage of either M:DCB or M:MOVEDCB and of M:OPEN and M:CLOSE is mandatory for a given DCB/file; usage of M:SETDCB and M:ASSIGN is optional.

Refer to foldout Chart A-6 of Appendix A for the syntax of the VSAM I/O procedures in reference form.

M:DCB Assembly-Time DCB Creation

Syntax:

[label(s)] M:DCB (OPL, 'op-label'), dcb param, . . .

Refer to foldout Chart A-2 in Appendix A for the general syntax of M:DCB, in reference form, showing the syntax of all DCB parameters. The parameters and values applicable under VSAM are described below, grouped by type of parameter.

FILE PROCESSING PARAMETERS

(ABN, address[, class-code, . . .]) – Abnormal Return Parameter. Specifies the address of the user's routine to which control is to be returned if certain abnormal conditions occur during file opening, processing, and closing. Also specifies the class(es) of conditions for which control is to be returned. None, some or all of the following abnormal class codes can be specified:

Code	Class
X1	File opening abnormalities.
X2	End-of-file/volume abnormalities.
X3	File processing abnormalities.
X4	BOF user tape label processing.
X5	EOF/EOV user tape label processing.
X6	BOV user tape label processing.

The address field is mandatory if this parameter is specified. If no class code is specified, no abnormal conditions are returned for user routine handling (see M:SETDCB description

for special usage). The section "Processing of Abnormal and Error Conditions" later in this chapter describes all abnormal conditions and corresponding codes in detail, and provides information on the type of processing allowed in user's abnormal- and error-handling routines. Occurence of an abnormal condition that has not been selected, by class, for user routine handling will cause a program abort.

(AM, VS) – Access Method Parameter. The value VS specifies usage of VSAM for I/O processing via this DCB. This parameter must appear since AS is the default value.

(ERR[, address]) – Error Return Parameter. Specifies the address of the user's routine to which control is to be returned if any error condition occurs during processing via this DCB. If this parameter is omitted, the occurrence of any error condition will cause a program abort. See the section "Processing of Abnormal and Error Conditions" later in this chapter for a detailed description of the possible error conditions and corresponding codes.

(MOD, BIN | BCD | EBC | PK | UPK) – Data Mode Parameter. Specifies the mode of data encoding to be expected on input or produced on output. The meaning and applicability of the alternative mode keyword (select one) are as follows:

Key	Mode	Applicable To
BIN	Binary	Cards and 7-track tape.
BCD	"Old" (026) BCD	7-track magnetic tape.
EBC	EBCDIC	Cards, tape, disk, and printer.
PK	Packed	7-track magnetic tape.
UPK	Unpacked	7-track magnetic tape.

The default mode is EBC.

(NRT) – No Retry Parameter. This parameter specifies that the system is not to execute the standard error recovery procedure (i.e., repeated retries of the same I/O operation) in case of a device or transmission error.

(ORG, C) – File Organization Parameter. Specifies the organization of the file to be created or accessed. If this parameter is specified, C (sequential) must be specified for file creation (output mode) or for any magnetic tape or nonmagnetic device processing. C is the default. When accessing an existing disk file, the actual file organization will override any ORG specification.

(SIM, value) – Simultaneous Operations Parameter. Specifies the maximum number of queued I/O requests to be accepted by the system through this DCB.

(TLB, address) – User Label Area Parameter. Specifies the address of a user program area into which the system will read a user label from magnetic tape, or from which the system will write a user supplied tape label (if so directed) at open or close time. At least one of the ABN class codes X4, X5, or X6 must also be specified for this parameter to be meaningful (see previous discussion of ABN parameter).

BLOCK-LEVEL PARAMETERS

(BKL, value) – Block Length Parameters. Specifies the block length, in bytes. The default values, by media type, are:

80 for EBCDIC card reader/punch files.

120 for binary card reader/punch files.

133 for printer files.

1024 for magnetic tape and disk files.

If a value greater than the applicable default is specified for nonmagnetic device files, the specified value will be replaced by the default value (open-time). If a value less than the applicable default (80 or 120) is specified for card punch files, a program abort will result. Otherwise, a value less than or greater than the default value may be specified (up to a maximum of 32K-1 bytes).

(MXL, value) – Maximum Transfer Length Parameter. Specifies the maximum length limit to be imposed on physical record transmissions for both reading and writing. If specified, the MXL value may be equal to or greater than the BKL value. The default value is the specified or default BKL value.

USAGE RULES

1. M:DCB is not an executable procedure. DCB space reservation and explicit parameter setting is performed at assembly time.
2. None of the parameters are indirectly addressable.

M:MOVEDCB Dynamic DCB Creation/Retention

The M:MOVEDCB procedure causes the allocation, during program execution, of a 19-word DCB area with read-only protection in the common-dynamic portion of the user's virtual memory space; the address of this area is returned to the user's program. It also causes the contents of a user specified sending area to be copied into the newly allocated DCB area. For example, the sending area may be a non-protected area of the user's program where a DCB image or "skeleton" has been built, or a write protected DCB that is not open when the M:MOVEDCB is executed. Effectively, this procedure allows the user to create DCBs during program

execution, and/or to move and thereby save inactive DCBs that would otherwise be overlaid or destroyed.

Syntax:

[label(s)] M:MOVEDCB [*]adr-1, (PTR, adr-2)

where

adr-1 is the word address of the 19-word sending area which is to be copied into the newly allocated DCB space.

adr-2 is the word address of a one-word pointer in the user's program in which the address of the newly allocated DCB is to be returned.

USAGE RULES

1. No default values are supplied for any parameter field by M:MOVEDCB.

2. The effect of M:SETDCB, M:ASSIGN, M:OPEN, and M:CLOSE procedures referencing a dynamically created or saved DCB is identical to the effect of the same procedures on an assembled DCB.

M:SETDCB Execution-Time DCB Modification

The M:SETDCB procedure allows the user, during program execution, to (1) "fill in" empty parameter fields in a nonopen DCB, i.e., specify previously unspecified or undefaulted DCB parameters, (2) modify previously specified or defaulted parameters in a nonopen DCB, and (3) modify the ABN and ERR parameters in an open DCB. The effective parameters in cases 1 and 2 are all of the parameters that may be specified in the M:DCB procedure for VSAM. See foldout Chart A-2 of Appendix A for the complete general syntax of M:SETDCB in reference form. The parameters applicable under VSAM are as previously described for the M:DCB procedure.

M:SETDCB also allows modification of the operational label associated with a DCB.

Syntax:

[label(s)] M:SETDCB [*]dcb-adr[,(OPL, {*adr-label / 'op-label'})][, dcb-parameter, ...]

where

dcb-adr is the address of the DCB to be modified.

adr-label is the address of a word containing an operational label, left-justified and space filled, to be associated with the DCB being modified.

'op-label' is a one- to four-character constant specifying an operational label to be associated with the DCB being modified.

dcb-parameter is a DCB parameter as described previously for the M:DCB procedure.

USAGE RULES

1. Any of the DCB parameter addresses and values can be indirectly addressed.

2. If the OPL option appears, the operational label specified therein will replace the operational label (if any) previously specified in the referenced DCB.

3. If the referenced DCB is open at the time of M:SETDCB execution, only the ABN and/or ERR parameter will be effective; any other parameters will be ignored.

4. A DCB closed with the HLD option (temporary close) is considered as open for the purposes of rule 3.

5. If the ABN parameter appears, the set of abnormal class codes specified will replace any such set previously in effect, including the null case where no class code is specified (i.e., any class codes previously in effect can be "turned off" by not specifying any class code).

M:ASSIGN Execution-Time DCB Assignment

The M:ASSIGN procedure allows the user, during program execution, to (1) define a temporary file in secondary storage and assign an operational label to it, or (2) define a permanent file on a physical resource specified by a !ASSIGN command. Effectively, case 1 allows the user to eliminate from the job control deck !ASSIGN commands for invariant "scratch file" assignments, or to define additional temporary files to satisfy dynamically determined program requirements (as M:MOVEDCB can be used to create additional DCBs). Case 2 allows the user to make an execution time choice between several !ASSIGN commands, each specifying a different resource, or to redefine a file (status, file name, or space allocation) assignment made by a !ASSIGN command. In case 2, the !ASSIGN command referred to must itself define a permanent, i.e., named, file.

Syntax:

Format 1, for temporary files

[label(s)] M:ASSIGN (OPL, {*adr-1 / 'op-label'})

[, (SIZ, {*adr-2 / size}, {*adr-3 / increment})]

where

adr-1 is the address of a word containing the operational label that identifies the DCB being assigned.

'op-label' is the character constant form of that label.

adr-2 is the address of a word containing the value specifying the primary size of the file.

size is the constant form of that size value.

adr-3 is the address of a word containing the value specifying the file increment (where applicable).

increment is the constant form of that increment value.

Format 2, for permanent files

[label(s)] M:ASSIGN (OPL, {*adr-1 / 'op-label'}),

(UNT, OPL, {*adr-4 / 'op-label-1'}), (NAM,

{*adr-5 / 'filename'}), (STS, {NEW / MOD / OLD}) [, (SIZ,

{*adr-2 / size}, {*adr-3 / increment})]

where

adr-1, 'op-label', adr-2, size, adr-3, and increment have the same meanings as in format 1.

adr-4 is the address of a word containing the operational label that identifies a !ASSIGN command specifying the desired physical resource.

'op-label-1' is the character constant form of the operational label described immediately above.

adr-5 is the word (or byte) address of a TEXT-format field containing the name of the file to be created or accessed. The name must be terminated by one or more blanks. (See the TEXT directive, Meta-Symbol Reference Manual, 90 09 52.)

'filename' is the character constant form of the file name.

USAGE RULES

1. Assignments made via M:ASSIGN are valid only for the job-step in which the procedure is executed, they cannot be maintained over succeeding steps. (Effectively, the !ASSIGN option FRE is implicit in the use of an M:ASSIGN procedure.)

2. The usage of M:ASSIGN in format 1 is identical to the usage of a !ASSIGN command with corresponding options specified.

3. In format 2, the UNT, OPL option implies use of the resource (device and volume) defined and allocated to the job-step by the operational label specified.

4. The !ASSIGN command referred to by the UNT, OPL option must specify a permanent, i.e., named, file assignment and must be in force during the job-step in which the M:ASSIGN is executed.

5. In format 2, all options other than UNT have the same meaning and usage as the corresponding options appearing in a !ASSIGN command, assuming the option FIL.

6. The M:ASSIGN procedure in format 1 may redefine an implicit assignment via a (installation dependent) predefined operational label, typically SI, GO, LM, etc.

7. In all cases, the operational label specified by the OPL keyword field establishes the link between a DCB and the file named by the procedure.

8. In format 2, the DCB parameters (if any) specified in the referenced !ASSIGN command are not applied to the DCB assigned via the procedure.

M:OPEN Opening a File

The M:OPEN procedure activates the link between a DCB and a physical file. The DCB is placed in an active, or open, status. ("Opening a file" is equivalent to "opening a DCB".) This procedure must be successfully executed before any I/O operation can be performed on a given file. The M:OPEN procedure performs some or all of the following steps while opening a file:

1. Location of the file to be processed, if it exists.

2. Creation of the file if it does not exist, which implies allocation of resources: volume and space on the volume, and creation of the file labels.

3. Positioning of magnetic tape.

4. Various initializations as necessary: completion of the DCB, reservation and completion of communication tables (IOBs) between the access methods and the I/O supervisor, etc.

5. Writing of the BKL value only from an existing file's HDR2 label into the DCB.

6. Validation and control of file identity, file expiration, file protection, and file sharing; checking for coherence among the DCB parameters, and consistency of these with the processing mode specified in M:OPEN and with the storage media.

Syntax:

[label(s)] M:OPEN [*]dcb-adr, mode [, {CID / NID / PID}]

where

dcb-adr is the address of the DCB to be opened.

mode specifies the processing mode in which the DCB is to be opened, as follows:

I Input mode (forward reading).

B Backward reading.

O Output mode (forward writing).

S Scratch mode (reading and writing).

CID specifies file identity checking, no tape positioning (default option).

NID specifies no file identity checking (for tape only).

PID specifies file identity checking with tape positioning.

USAGE RULES

1. Processing mode B is applicable to magnetic media files only.

2. The file identity checking and positioning option (CID, NID, or PID) is meaningful for standard magnetic tape files only; for DEV assigned files it is ignored; for disk/RAD files CID is always assumed.

3. The effect of a given processing mode specification (and identity checking option, if tape) is conditioned by the declared status of the file versus the file's actual status. These interrelationships are described below.

RELATIONSHIP OF PROCESSING MODE AND FILE STATUS

The status of a file (NEW, MOD, or OLD) is declared in the file assignment. For each declared file status, there is one or several normal combinations of status, file existence/nonexistence, and processing mode. The normal combination(s) does not occasion any abnormal return to the user's program during opening. Certain abnormal combinations (described below) are acceptable but will result in an X1 class abnormal return to the user's program during opening if the DCB specifies an abnormal return with the X1 class set (abort otherwise). Any combination other than those described below will result in a program abort.

The several normal and abnormal combinations are described for each declared file status in the following paragraphs.

- NEW

  Normal combination. Nonexistent file and output mode – original file creation; output operations allowed only.

  Abnormal combination. Existent file and output mode – in this case an abnormal return to the user's program occurs. The user can then request recreation of the file by a special exit from his abnormal routine. If the expiration date of the file to be rewritten has not been reached, a further abnormal return occurs; another special exit will allow processing to continue, overriding the nonexpired condition.

- MOD

  Normal combinations.

  1. Existent file and scratch mode – read and write operations allowed.

  2. Existent file and output mode – file extension output allowed only.

  3a. Existent file and input mode – input operations allowed only.

  3b. Existent file and backward-read mode – input operations allowed only.

  4. Nonexistent file and output mode – status is changed to NEW; output operations are allowed.

- OLD

  Normal combinations.

  1a. Existent file and input mode – input operations are allowed.

  1b. Existent file and backward-read mode – input operations are allowed.

  2. Existent, expired, labeled file and output mode – output operations allowed (i.e., the file may be rewritten).

  Abnormal combination. Existent, nonexpired, labeled file – in this case an abnormal return to the user's program occurs. A special exit from the user's abnormal routine allows processing to continue. The file can then be rewritten.

Note: See the section "Processing of Abnormal and Error Conditions" for specific abnormal codes and further details on normal/special exit results.

AUTOMATIC EVOLUTION OF FILE STATUS

The automatic evolution of the status of a file during a job-step when opened for output is shown below.

	Declared Status Prior to First M:OPEN, Output		
	NEW, Nonexistent File	MOD Nonexistent File	OLD, Existent File
Status after successful opening	NEW	NEW	NEW
Status after M:CLOSE	MOD	MOD	MOD
Note: Does not apply to DEV assigned magnetic tape.			

MAGNETIC TAPE POSITIONING AND FILE IDENTITY CHECKING

If a standard magnetic tape volume has not been previously processed since its mounting, its initial position (relative to the tape unit read/write heads) during the opening process is between the standard-volume-label group, SVL (or user-volume-label group, UVL) and the first standard-file-header-label group, SHL. This position is shown in Figure 7-1 as position A.

If the volume has been previously processed since mounting, its initial positioning will depend upon the action taken during the previous M:CLOSE operation: between two files (position B in Figure 7-1), in front of the first data block (position C), or after the last data block (position D), of some file on the volume.

The open process will then perform testing and positioning functions from these initial magnetic tape positions. These functions are controlled by the CID, NID, and PID options. The default option is CID. Only the CID option is significant for disk/RAD files. Several cases exist, as described in the following paragraphs.

First Case, File Opened for Creation (STS, NEW: Processing Mode Out).

- CID

  For disk files, this option ensures that the identification assigned to the file being created does not already exist in the volume or account catalog.

  For magnetic tape files, no positioning is executed. There is a verification that a file having the same identification as the file being created does not already exist at the current position on the tape. If the file at this location has a different identity and its expiration date has been reached, it is written over by the new file. In the cases where the file has the same name and/or the expiration date has not been reached, a return is made to the user's abnormal routine. This permits the user to request creation of a new file replacing the old (special exit), or not to complete the open (normal return). (See "Processing of Abnormal and Error Conditions".)

  If there is no file at the initial position on the magnetic tape, the opening of the file to be created continues normally.

- NID

  There is no checking of file identification; only a test of the expiration date of the file at the initial position is performed.

- PID

  This option causes automatic creation of the new file following all of the old files contained on the tape. A test of the file identification is made during positioning against all files encountered between the initial position and the final old file. If one of the files encountered has the same identity as the file to be created, a return is made to the user's routine. Then, using a special exit, the user can write over this old file; otherwise the M:OPEN is not executed.

  Figure 7-2 summarizes the positioning and identification test functions caused by these options.

Second Case, File Opened in Forward Read.

- CID

  For magnetic tape, this option causes testing of the current file identification. For disk, the catalog is searched for the corresponding file.

  When the identifications are not equal, a return is made to the users abnormal routine. For a magnetic tape file, the user can request reading of the present file (special exit); otherwise the open is not executed (normal exit).

- NID

  No checking or positioning is executed. The opening is to the first file encountered.

- PID

  This option causes a forward search for the identification specified by the !ASSIGN command, starting from the initial location on magnetic tape at opening time. If the two tape marks identifying the end of volume are encountered, a tape rewind is executed and a return is made to the user's abnormal routine. Using a special exit, the user can reinitialize a forward search for the

file to be processed. If the file is not found after this second search, an abnormal return is again made but the open will not be executed.

Third Case, File Opened in Backward Read. In this case, the testing refers to the file preceding the one at which the tape is initially positioned. The initial positioning to end-of-file is the responsibility of the user program.

- CID

  Causes identification checking of the preceding file.

- NID

  No checking is performed.

- PID

  This option, associated with B processing mode, is identical to CID. That is, no backward search is performed.

Note: See the M:CLOSE positioning options LVE and RRD for end-of-file positioning on a temporary (HLD) file closing.

M:CLOSE Closing a File

This procedure causes the closing of a file. The close effects a suspension or halt of activity via a DCB and possibly via the processed file. Its main functions are:

1. Waiting for and testing the last I/O operation(s).
2. Checking or writing the end-of-file labels on magnetic tape.
3. Updating and validating disk file labels.
4. Temporary or definite closing of the DCB.
5. Partial restitution of disk space or deletion of the file created by the job.
6. Releasing devices, tables, etc.

Functions 5 and 6 are conditional upon the type of closing.

Syntax:

[label(s)] M:CLOSE [*]dcb-adr[,{HLD / MTN / RLS}] [,{RRD / LVE / RWD}][,{KEP / NCG}]

where dcb-adr is the address of the DCB to be closed.

TYPES OF CLOSE

HLD specifies a temporary closing of the file. The DCB-file link is maintained. A new M:OPEN applied to the DCB will only permit changing of the processing mode of the file, because only a small portion of the open functions will then be executed.

MTN specifies a definite close. It cancels the DCB-file link but maintains the job-file link. In the same job another DCB can be used to process the same file. The resource is not released even if the FRE option was specified in the assignment.

RLS specifies a definite close. It destroys the DCB-file link and the job-file link. The resource(s) allocated for the file are released if the FRE option was specified in the assignment.

POSITIONING AFTER CLOSING

The options controlling positioning, RRD (reread), LVE (leave), and RWD (rewind), are significant only for sequential organization files on magnetic media (refer to Figure 7-3).

The significance of these options depends on the type of close: temporary (HLD) or definite (MTN or RLS). It also depends on the processing mode of the file: forward (I, O, and S) or backward (B). The options have significance for a disk file only on a temporary close.

Temporary Close (HLD). Upon closing a file, positioning remains within the file being accessed, or within the volume of the file being accessed in the case of a serially mounted multivolume file. This means the file is positioned (in the case of tape) somewhere between the two tape marks delimiting the data blocks.

For I, O, or S processing (see Figure 7-3):

RRD requests positioning before the first data block of the file (or portion of the file).

LVE requests positioning after the last data block of the file (or portion of the file).

RWD is not meaningful in this type of close. In this case this option is equivalent to RRD.

For B processing (see Figure 7-3), the RRD and LVE options have positioning meanings in reverse of those for forward processing:

RRD requests positioning after the last data block of the file (or portion of the file).

LVE requests positioning before the first data block of the file (or portion of the file).

RWD is equivalent to LVE in this case.

Definite Close (MTN or RLS). Positioning is always made outside the file limits (the labels and tape marks delimiting the file), or the volume limits in the case of a serially mounted multivolume file.

For I, O, or S processing on magnetic tape (see Figure 7-3):

RRD requests positioning before the SHL of the file (or portion of the file).

LVE requests positioning after the STL of the file (or portion of the file).

RWD requests rewinding and positioning after the volume SVL.

For B processing on magnetic tape (see Figure 7-3), the RRD and LVE options have meanings in reverse of those for forward operations:

RRD signifies positioning after the STL of the file or portion of the file.

LVE requests positioning before the SHL of the file (or portion of the file).

RWD requests rewinding and positioning after the volume SVL.

The positioning described never crosses over volume boundaries for serially mounted multivolume files.

A rewind operation is used by M:CLOSE only for the RWD option in an RLS or MTN closing. In the other cases the positionings are executed using only tape-mark-search and skip-block operations.

LABEL VALIDATION AND FILE DISPOSITION

The KEP and NCG options specify the disposition of the permanent file. These are significant only for the O processing mode (file creation).

KEP requests validation of the file labels for a disk/RAD file, and cataloging of the file if cataloging was requested or implied.

NCG signifies deletion of the file for a file created on a disk/RAD volume and suppression of cataloging where cataloging has been requested or implied. This option is applicable only in conjunction with RLS and if the status of the file is NEW prior to closing.

USAGE RULES

1. If no type-of-close option is specified, RLS (release) is assumed by default. Both the DCB-file connection and the job-file connection are severed (i.e., another job can attempt access to the file whether the volume is sharable or not). All resources allocated for the processing of this file are released if MTN was not specified in the assignment.

2. If no positioning option is specified, RWD (rewind) is assumed by default.

3. If no validation/disposition option is specified, KEP (keep) is assumed by default.

4. If a definite close (MTN or RLS) is not issued for a given DCB/file before the end of the job-step, the system forces an unconditional (RLS) close, disposition KEP.

M:READ Read Next Sequential Physical Record

The M:READ procedure requests reading of the next (or first) sequential physical record of a file. (Exception: see M:POINT procedure). The record read may be a device determined physical record in the case of card files, or a program determined physical record in the case of disk/RAD files. Either a full or partial physical tape block may be read from magnetic tape files.

For existing permanent files on standard volumes, the block length (BKL) is determined from the file's HDR2 label. MXL limits the maximum physical record transfer length. Records are transmitted from the physical file directly to the user's buffer area specified in the procedure.

Return from the procedure is made immediately following the system's acceptance of the request (i.e., not after initiation or completion of the I/O operation). Successive I/O requests (via the same DCB) may be issued, prior to executing an M:CHECK and/or M:WAIT procedure referring to these requests, up to the limit specified by the SIM parameter of the DCB.

Syntax: See foldout Chart A-6 in Appendix A.

MEANING OF THE PARAMETERS

dcb-adr specifies the address of the DCB through which the read is to be performed.

BUF, $\left\{\begin{matrix}[*]\text{badr-1}\\ \text{adr-2}\end{matrix}\right\}$ specifies either the initial byte address, badr-1 (optionally indirect), of the user's buffer area into which the record is to be read; or the word address, adr-2 (direct only), of the buffer area (which will be converted to a byte address).

TRL, [*]value specifies the number of bytes to be transmitted, i.e., the length of the physical record to be read (may be indirect). If not specified, the default value is BKL.

PTR, [*]adr-3 specifies the address, adr-3 (optionally indirect), of a pointer word into which the system

is to return the address of the event control block (ECB) associated with this I/O request.

USAGE RULES

1. The TRL value may not exceed the value of the MXL parameter in the referenced DCB.

2. The TRL value, if specified, must be 80 (EBCDIC) or 120 (binary) for card files.

3. For tape or disk/RAD files, TRL values less than, equal to, or greater than BKL are valid.

4. A maximum of one physical tape block will be transmitted, i.e., transmissions of less than TRL or BKL (default for TRL) may occur.

5. Use of the PTR option causes the system to supply, on return from the procedure, a pointer to the (system constructed) ECB associated with the specific request and the implied I/O operation. This pointer must then be used in a subsequent M:CHECK procedure to identify the operation to be checked.

6. An abnormal or error return, if selected, for a given I/O operation will be made only from an M:CHECK procedure referring (explicitly or implicitly) to that operation.

7. If two or more I/O requests have been queued (i.e., accepted by the system for execution) and the nth implied operation terminates unsuccessfully, and the system attempts to initiate the nth + 1 implied operation before an M:CHECK referring to the nth operation is executed, the nth + 1 and all following requests will be purged.

8. If the number of queued I/O requests, including requests for which I/O operations have been completed but not checked, exceeds the value of the SIM parameter (in the DCB), a program abort will occur.

9. In scratch processing mode, a read operation may not immediately follow a write operation without intervening repositioning. That is, for existing files, a write operation in midfile effectively truncates the file at the end of the record just written.

M:WRITE Write Next Sequential Physical Record

The M:WRITE procedure requests writing of the next (or first) sequential physical record of a file in output or scratch processing modes. (Exception: see M:POINT.) The record written may be a device determined physical record in the case of card files, or a program determined physical record in the case of magnetic tape and disk/RAD files. Records written to a printer may vary in length up to the device determined maximum, e.g., 133 bytes.

The maximum data transfer length (MXL parameter in the DCB) may exceed the block length as defined by the BKL parameter. Records are transmitted directly from the user's buffer area specified in the procedure to the physical file.

Return from the procedure is made immediately following the system's acceptance of the request (not after initiation or completion of the I/O operation). Successive I/O requests (via the same DCB) may be issued, prior to the execution of an M:CHECK and/or M:WAIT procedure referring to these requests, up to the limit specified by the SIM parameter of the DCB.

Syntax: See foldout Chart A-6 in Appendix A.

MEANING OF THE PARAMETERS

dcb-adr specifies the address of the DCB through which the write is to be performed.

BUF, $\left\{\begin{matrix}[*]\text{badr-1}\\ \text{adr-2}\end{matrix}\right\}$ specifies either the initial byte address, badr-1 (optionally indirect), of the user's buffer area from which the record is to be written; or the word address adr-2 (direct only), of the buffer area (which will be converted to a byte address).

TRL, [*]value specifies the number of bytes to be transmitted, i.e., the length of the physical record to be written (may be indirect). If not specified the default value is BKL.

PTR, [*]adr-3 specifies the address adr-3 (optionally indirect), of a pointer word into which the system is to return the address of the event control block (ECB) associated with this I/O request.

USAGE RULES

1. The TRL value may not exceed the value of the MXL parameter in the referenced DCB.

2. The TRL value, if specified, must be 80 (EBCDIC) or 120 (binary) for card files.

3. For tape or disk/RAD files, TRL values less than, equal to, or greater than block length (BKL) are valid.

4. Use of the PTR option causes the system to supply, on return from the procedure, a pointer to the (system constructed) ECB associated with the specific request and the implied I/O operation. This pointer must then be used in a subsequent M:CHECK procedure to identify the operation to be checked.

5. An abnormal or error return, if selected, for a given I/O operation will be made only from an M:CHECK procedure referring (explicitly or implicitly) to that operation.

6. If two or more I/O requests have been queued (i.e., accepted by the system for execution) and the nth implied operation terminates unsuccessfully, and the system attempts to initiate the nth + 1 implied operation before an M:CHECK referring to the nth operation is executed, the nth + 1 and all following requests will be purged.

7. If the number of queued I/O requests, including requests for which I/O operations have been completed but not checked, exceeds the value of the SIM parameter (in the DCB), a program abort will occur.

8. In Output processing mode, any record structure parameters present in the DCB will be written into the file's HDR2 label, if any, permitting subsequent processing by ASAM. All logical record structuring, however, must be performed by the user, including block and record level headers where applicable.

M:CHECK Check I/O Event Completion

The M:CHECK procedure tests a given I/O operation for proper completion, placing the issuing program in a wait state if necessary to await such completion. If the tested operation terminates unsuccessfully, an appropriate abnormal or error return is made to the user's program (if selected in the user's program, abort otherwise). Optionally, for a read operation, the actual length of the record (or partial transmission) read is reported.

The specific operation to be checked may be indicated in either of two modes:

- Explicitly, in the procedure (PTR parameter), by means of a pointer to the event control block (ECB) associated with the operation (obtained via the corresponding M:READ or M:WRITE request).

- Implicitly (no pointer to the ECB), as the operation corresponding to the oldest outstanding (i.e., yet unchecked) I/O request.

The program must use one of these two modes exclusively for all operations via a given DCB.

The M:CHECK procedure may be used in conjunction with the M:WAIT procedure, described in Chapter 8.

Syntax: See foldout Chart A-6 in Appendix A.

MEANING OF THE PARAMETERS

dcb-adr specifies the address of the DCB through which the check is to be performed.

PTR, [*]adr-1 specifies the address, adr-1 (optionally indirect), of a pointer to the ECB associated with the operation to be checked (see M:READ and M:WRITE).

RSA, [*]adr-2 specifies the address, adr-2 (optionally indirect), of a word into which the system is to return the actual length (in bytes) of the I/O transmission in the case of a read operation.

USAGE RULES

1. Between the initial opening and the definite closing of a DCB, all M:CHECK procedures referring to the DCB must either always specify the PTR parameter or always omit that parameter.

2. If the PTR parameter is specified, all operations via a given DCB must be checked in the order in which they were requested.

3. If the PTR parameter is not specified, the earliest request, still unchecked operation is checked.

4. If the RSA parameter is specified for a write operation, it is ignored.

5. The M:CHECK procedure, when executed, effectively releases the system table (IOB) entry associated with a given operation, allowing it to be used for queuing of subsequent requests.

M:CVOL Switch to Next Volume

The function and usage of M:CVOL is the same as under the Assisted Sequential Access Method, ASAM.

M:NOTE Note Current Position

Except that the position pointer obtained always refers to a block boundary, the function and usage of M:NOTE is the same as under ASAM.

M:POINT Reposition by Pointer

The function and usage of M:POINT is the same as under ASAM.

M:DEVICE Device Dependent Operations

The functions and usages of M:DEVICE are the same as under ASAM, but with the following additional form and usage rules.

Additional Syntax:

[label(s)] M:DEVICE [*]dcb-adr, (WEOF)

where

dcb-adr is the address of the DCB through which the operation is to be performed.

WEOF specifies that a tape mark is to be written at the current position on the tape.

USAGE RULES

1. The WEOF option is effective only for magnetic tape files assigned as device type (DEV assignment) files (see !ASSIGN command).

2. Used under VSAM, any M:DEVICE execution must be preceded by verification, via M:CHECK, of the completion of all previously requested I/O operations.

VDAM (VIRTUAL DIRECT ACCESS METHOD)

VDAM is a direct access method, operating on program defined physical records, and applicable to files on standard disk/RAD volumes only. It creates and processes direct (D) organization files. Access to the physical records of such files is by relative block number. (Direct organization files can also be accessed sequentially with VSAM or ASAM, U format.) No logical record structures are provided for or recognized; undefined (U) record format is implicit in this access method.

GENERAL USAGE RULES

The general usage rules for VDAM are listed below.

1. Creates and processes direct organization files.

2. Accesses (in input mode) files of any organization, by relative block number.

3. Applicable processing modes:

   Input.

   Output.

   Scratch (reading and writing).

4. Applicable to direct access media only.

5. Not applicable to nonstandard (DEV assigned) disk volumes.

6. No I/O buffering is performed by the system. That is, physical transmission of data is directly to or from the user program area specified by the M:READ or M:WRITE procedure.

7. The amount of data transmitted (i.e., the length of the physical record) is user defined for each I/O operation or defaults to BKL.

8. Physical records shorter or longer than the block size (BKL value) can be written and read.

9. Testing for successful completion of an I/O operation (or sequence of such operations) is not performed by VDAM. The user must himself check for I/O operation completions, successful or otherwise, with the M:CHECK procedure, possibly in conjunction with the M:WAIT procedure.

10. Abnormal and/or error returns (if selected) are made from the M:CHECK procedure only.

11. Requires that multivolume files be parallel mounted.

12. Applicable I/O processing procedures:

    M:READ

    M:WRITE

    M:CHECK

See foldout Chart A-7 of Appendix A for the syntax of the VDAM I/O procedures in reference form.

M:DCB Assembly-Time DCB Creation

Syntax:

[label(s)] M:DCB (OPL,'op-label'[,dcb-param,...]

Refer to foldout Chart A-2 in Appendix A for the general syntax of M:DCB, in reference form, showing the syntax of all DCB parameters. The parameters and values applicable under VDAM are described below, grouped by type of parameter.

FILE PROCESSING PARAMETERS

(ABN,address[,class-code,...]) – Abnormal Return Parameter. Specifies the address of the user's routine to which control is to be returned if certain abnormal conditions occur during file opening, processing, and closing. Also specifies the class(es) of conditions for which control is to be returned. None, some, or all of the following abnormal class codes can be specified:

Code	Class
X1	File opening abnormalities.
X2	End-of-file/volume abnormalities.
X3	File processing abnormalities.

The address field is mandatory if this parameter is specified. If no class code is specified, no abnormal conditions are returned for user routine handling (see M:SETDCB description for special usage). The section "Processing of Abnormal and Error Conditions" later in this chapter describes all abnormal conditions and corresponding codes in detail, and provides information on the type of processing allowed in user's abnormal- and error-handling routines. Occurrence of an abnormal condition that has not been selected, by class, for user routine handling will cause a program abort.

(AM,VD) – Access Method Parameter. The value VD specifies usage of VDAM for I/O processing via this DCB. This parameter must be specified.

(ERR[, address]) – Error Return Parameter. Specifies the address of the user's routine to which control is to be returned if any error condition occurs during processing via this DCB. If this parameter is omitted or the address is omitted, the occurrence of any error condition will cause a program abort. See the section "Processing of Abnormal and Error Conditions" later in this chapter, for a detailed description of the possible error conditions and corresponding codes.

(MOD, EBC) – Data Mode Parameter. Specifies the mode of data encoding to be expected on input or produced on output. The mode keyword EBC specifies EBCDIC encoding, the only mode applicable. The default mode is EBC.

(NRT) – No Retry Parameter. This parameter specifies that the system is not to execute the standard error recovery procedure (i.e., repeated retries of the same I/O operation) in case of a device or transmission error.

(ORG, D|C|I|P) – File Organization Parameter. Specifies the organization of the file to be created or accessed. D (direct) must be specified for file creation or modification (output or scratch mode). C, I, or P may be specified for reading disk files created by VSAM, ASAM, AIAM, or APAM.

(SIM, value) – Simultaneous Operations Parameter. Specifies the maximum number of I/O requests to be accepted by the system through this DCB.

BLOCK LEVEL PARAMETERS

(BKL, value) – Block Length Parameter. Specifies the block length, in bytes. The default value is 1024. A block length value less than or greater than the default value may be specified (up to a maximum of 32K-1 bytes). See "Block Formats" in Chapter 6 for a discussion of block length. At open time, the BKL value in an existing file's HDR2 label will override an unlike BKL value in the DCB (unless status OLD, output processing mode).

(MXL, value) – Maximum Transfer Length Parameter. Specifies the maximum length limit to be imposed on physical record transmissions for both reading and writing. The maximum specifiable value is 32K-1. The default value is the specified or default BKL value.

USAGE RULES

1. M:DCB is not an executable procedure. DCB space reservation and explicit parameter setting is performed at assembly time.

2. None of the parameters are indirectly addressable.

M:MOVEDCB Dynamic DCB Creation/Retention

The M:MOVEDCB procedure causes the allocation, during program execution, of a 19-word DCB area with read-only protection in the common-dynamic portion of the user's virtual memory space; the address of this area is returned to the user's program. It also causes the contents of a user specified sending area to be copied into the newly allocated DCB area. For example, the sending area may be a nonprotected area of the user's program where a DCB image or "skeleton" has been built, or a write protected DCB that is not open when the M:MOVEDCB is executed. Effectively, this procedure allows the user to create DCBs during program execution, and/or to move and thereby save inactive DCBs that would otherwise be overlaid or destroyed.

Syntax:

[label(s)] M:MOVEDCB [*]adr-1, (PTR, adr-2)

where

adr-1 is the word address of the 19-word sending area which is to be copied into the newly allocated DCB space.

adr-2 is the word address of a one-word pointer in the user's program into which the address of the newly allocated DCB is to be stored by the system.

USAGE RULES

1. No default values are supplied for any parameter field by M:MOVEDCB.

2. The effect of M:SETDCB, M:ASSIGN, M:OPEN, and M:CLOSE procedures referencing a dynamically created or saved DCB is identical to the effect of the same procedures on an assembled DCB.

M:SETDCB Execution-Time DCB Modification

The M:SETDCB procedure allows the user, during program execution, to (1) "fill in" empty parameter fields in a nonopen DCB, i.e., specify previously unspecified or undefaulted DCB parameters, (2) modify previously specified or defaulted parameters in a nonopen DCB, and (3) modify the ABN and ERR parameters in an open DCB. The effective parameters in cases 1 and 2 are all of the parameters that may be specified in the M:DCB procedure for VDAM. See foldout Chart A-2 of Appendix A for the complete general syntax of M:SETDCB in reference form. The parameters applicable under VDAM are as previously described for the M:DCB procedure under ASAM.

M:SETDCB also allows modification of the operational label associated with a DCB.

Syntax:

[label(s)] M:SETDCB [*]dcb-adr[, (OPL {*adr-label / 'op-label'})] [, dcb-parameter, ...]

where

dcb-adr is the address of the DCB to be modified.

adr-label is the address of a word containing the operational label, left-justified and space filled, to be associated with the DCB being modified.

'op-label' is a one- to four-character constant specifying the operational label to be associated with the DCB being modified.

dcb-parameter is a DCB parameter as described previously for the M:DCB procedure.

USAGE RULES

1. Any of the DCB parameter addresses and values can be indirectly addressed.

2. If the OPL option appears, the operational label specified therein will replace the operational label (if any) previously specified in the referenced DCB.

3. If the referenced DCB is open at the time of M:SETDCB execution, only the ABN and/or ERR parameter will be effective; any other parameters will be ignored.

4. A DCB closed with the HLD option (temporary close) is considered as open for the purposes of rule 3.

5. If the ABN parameter appears, the set of abnormal-class codes specified will replace any such set previously in effect, including the null case where no class code is specified (i.e., any class codes previously in effect can be "turned off" by not specifying any class code).

M:ASSIGN Execution-Time DCB Assignment

The M:ASSIGN procedure allows the user, during program execution, to (1) define a temporary file in secondary storage and assign an operational label to it, or (2) define a permanent file on a physical resource specified by a !ASSIGN command. Effectively, case 1 allows the user to eliminate from the job control deck !ASSIGN commands for invariant "scratch file" assignments, or to define additional temporary files to satisfy dynamically determined program requirements (as M:MOVEDCB can be used to create additional DCBs). Case 2 allows the user to make an execution time choice between several !ASSIGN commands, each specifying a different resource, or to redefine a file assignment (status, file name, or space allocation) made by a !ASSIGN command. In case 2, the !ASSIGN command referred to must itself define a permanent, i.e., named, file.

Syntax:

Format 1, for temporary files

[label(s)] M:ASSIGN (OPL, {* adr-1 / 'op-label'})

[, (SIZ, {*adr-2 / size})]

where

adr-1 is the address of a word containing the operational label that identifies the DCB being assigned.

'op-label' is the character constant form of that label.

adr-2 is the address of a word containing the value specifying the size of the data block portion of the file.

size is the constant form of that size value.

Format 2, for permanent files

[label(s)] M:ASSIGN (OPL, {*adr-1 / 'op-label'}),

(UNT, OPL, {*adr-4 / 'op-label-1'}),

(NAM, {*adr-5 / 'filename'}), (STS, {NEW / MOD / OLD})

[, (SIZ, {*adr-2 / size})]

where

adr-1, 'op-label', adr-2, and size have the same meaning as in format 1.

adr-4 is the address of a word containing the operational label that identifies a !ASSIGN command specifying the desired physical resource.

'op-label-1' is the character constant form of the operational label described immediately above.

adr-5 is the word (or byte) address of a TEXT-format field containing the name of the file to be created or accessed. The name must be terminated by one or more blanks. (See the TEXT directive, Meta-Symbol Reference Manual, 90 09 52.)

'filename' is the character constant form of the file name.

USAGE RULES

1. Assignments made via M:ASSIGN are valid only for the job-step in which the procedure is executed; they cannot be maintained over succeeding steps. (Effectively, the !ASSIGN option FRE is implicit in the use of an M:ASSIGN procedure.)

2. The usage of M:ASSIGN in format 1 is identical to the usage of a !ASSIGN command with corresponding options specified.

3. In format 2, the UNT, OPL option implies use of the resource (device and volume) defined and allocated to the job-step by the operational label specified.

4. The !ASSIGN command referred to by the UNT, OPL option must specify a permanent, i.e., named, file assignment on direct access media and must be in force during the job-step in which the M:ASSIGN is executed.

5. In format 2, all options other than UNT have the same meaning and usage as the corresponding options appearing in a !ASSIGN command, assuming the option FIL.

6. The M:ASSIGN procedure in format 1 may redefine an implicit assignment via a (installation dependent). predefined operational label, typically SI, GO, LM, etc.

7. In all cases, the operational label specified by the OPL keyword field establishes the link between a DCB and the file defined by the procedure.

8. In format 2, the DCB parameters (if any) specified in the referenced !ASSIGN command are not applied to the DCB assigned via the procedure.

M:OPEN Opening a File

The M:OPEN procedure activates the link between a DCB and a physical file. The DCB is placed in an active, or open, status. ("Opening a file" is equivalent to "opening a DCB".) This procedure must be successfully executed before any I/O operation can be performed on a given file. The M:OPEN procedure performs some or all of the following steps while opening a file:

1. Location of the file to be processed, if it exists.

2. Creation of the file if it does not exist, which implies allocation of resources: volume and space on the volume, and creation of the file labels.

3. Various initializations as necessary: reservation of buffers, completion of the DCB, reservation and completion of communication tables (IOBs) between the access methods and the I/O supervisor, etc.

4. Writing of the BKL value only from an existing file's HDR2 label into the DCB.

5. Validation and control of file identity, file expiration, file protection, and file sharing; checking for coherence among the DCB parameters, and consistency of these with the processing mode specified in M:OPEN and with the storage media.

Syntax:

[label(s)] M:OPEN [*]dcb-adr, mode[, CID]

where

dcb-adr is the address of the DCB to be opened.

mode specifies the processing mode in which the DCB is to be opened, as follows:

I Input mode (reading).

O Output mode (writing).

S Scratch mode (reading and writing).

CID specifies file identity checking (see Usage Rule 1).

USAGE RULES

1. The file identity checking and position options NID and PID (see VSAM) are meaningful for magnetic tape files only. For disk/RAD files this field is effectively ignored. CID is always assumed.

2. The effect of a given processing mode specification is conditioned by the declared status of the file versus the file's actual status. These interrelationships are described below.

RELATIONSHIP OF PROCESSING MODE AND FILE STATUS

The status of a file (NEW, MOD, or OLD) is declared in the file assignment. For each declared file status there is one or several normal combinations of status, file existence/nonexistence, and processing mode. The normal combinations do not occasion any abnormal return to the user's program during opening. Certain abnormal combinations (described below) are acceptable but will result in an X1 class abnormal return to the user's program during opening if the DCB specifies an abnormal return with the X1 class set (abort otherwise). Any combination other than those described below result in a program abort. The several normal and abnormal combinations are described for each declared file status in the following paragraphs.

- NEW

  Normal combination. Nonexistent file and output mode – original file creation; output operations allowed only.

Abnormal combination. Existent file and output mode – in this case an abnormal return to the user's program occurs. The user can then request recreation of the file by a special exit from his abnormal routine. If the expiration date of the file to be rewritten has not been reached, a further abnormal return occurs; another special exit will allow processing to continue, overriding the nonexpired condition.

- MOD

  Normal combinations.

  1. Existent file and scratch mode – read and write operations allowed.
  2. Existent file and input mode – input operations allowed only.
  3. Nonexistent file and output mode – status is changed to NEW; output operations are allowed.

- OLD

  Normal combinations.

  1. Existent file and input mode – input operations are allowed.
  2. Existent, expired file and output mode – output operations allowed (i.e., the file may be rewritten).

  Abnormal combination. Existent, nonexpired file – in this case an abnormal return to the user's program occurs. A special exit from the user's abnormal routine allows processing to continue; the file can then be rewritten.

Note: See the section "Processing of Abnormal and Error Conditions" for specific abnormal codes and further details on normal/special exit results.

AUTOMATIC EVOLUTION OF FILE STATUS

The automatic evolution of the status of a file during a job-step when opened for output is shown below.

	Declared Status Prior to First M:OPEN, Output		
	NEW, Nonexistent File	MOD, Nonexistent File	OLD, Existent File
Status after successful opening	NEW	NEW	NEW
Status after M:CLOSE	MOD	MOD	MOD

M:CLOSE Closing a File

This procedure causes the closing of a file. The close effects a suspension or halt of activity via a DCB and possibly via the processed file. Its main functions are:

1. Waiting for and testing the last I/O operation(s).
2. Updating and validating disk file labels.
3. Temporary or definite closing of the DCB.
4. Partial restitution of disk space or deletion of the file created by the job.
5. Releasing devices, tables, etc.

Syntax:

[label(s)] M:CLOSE [*]dcb-adr [,{HLD / MTN / RLS}] [,{KEP / NCG}]

where dcb-adr is the address of the DCB to be closed.

TYPES OF CLOSE

HLD specifies a temporary closing of the file. The DCB-file link is maintained. A new M:OPEN applied to the DCB will only permit changing of the processing mode, because only a small portion of the open functions will then be executed.

MTN specifies definite close. It cancels the DCB-file link but maintains the job-file link. In the same job another DCB can be used to process the same file. The resource is not released even if the FRE option was specified in the assignment.

RLS specifies a definite close. It destroys the DCB-file link and the job-file link. The device resource(s) allocated for the file are released if the FRE option was specified in the assignment.

LABEL VALIDATION AND FILE DISPOSITION

The KEP and NCG options specify the disposition of a newly created permanent file. They are significant only for the O processing mode (file creation).

KEP requests validation of the file labels, and cataloging of the file if requested or implied.

NCG requests deletion of the file and suppression of cataloging if cataloging has been requested or implied. This option is applicable only in conjunction with RLS and if the status of the file is NEW prior to closing.

USAGE RULES

1. If no type-of-close option is specified, RLS (release) is assumed by default. Both the DCB-file connection and the job-file connection are severed (i.e., another job can attempt access to the file whether the file/volume is sharable or not). All device (i.e., common) resources allocated for the processing of this file are released if MTN was not specified in the assignment.

2. If no validation/disposition option is specified, KEP (keep) is assumed by default.

3. If a definite close (MTN or RLS) is not issued for a given DCB/file before the end of the job-step, the system forces an unconditional (RLS) close, disposition KEP.

M:READ Read Virtual Physical Record

The M:READ procedure requests reading of a program determined physical record, starting from a block boundary specified by relative block number. The length of the record to be read is either specified in the procedure (TRL parameter) or defaults to block length (BKL). In either case it is limited by the maximum transfer length (MXL) specified in the DCB. A relative block number (ADR parameter) specifying the block (relative to the beginning of the file) at which the read is to begin, must always be given.

For existing permanent files, the block length (BKL) is determined from the file's HDR2 label. Records are transmitted from the physical file directly to the user's buffer area specified in the procedure.

Return from the procedure is made immediately following the system's acceptance of the request (i.e., not after initiation or completion of the I/O operation). Successive I/O requests (via the same DCB) may be issued, prior to execution of M:CHECK and/or M:WAIT procedures referring to these requests, up to the limit specified by the SIM parameter of the DCB.

Syntax: See foldout Chart A-7 in Appendix A.

MEANING OF THE PARAMETERS

dcb-adr specifies the address of the DCB through which the read is to be performed.

BUF, $\left\{\begin{matrix}[*]badr-1\\adr-2\end{matrix}\right\}$ specifies either the initial byte address badr-1 (optionally indirect), of the user's buffer area into which the record is to be read; or the word address, adr-2 (direct only), of the buffer area (which will be converted to a byte address).

ADR, [*]value-1 specifies the relative block number (where block 0 is the beginning of the file) of the block at which the read operation is to begin (may be indirect).

TRL, [*]value-2 specifies the number of bytes to be transmitted, i.e., the length of the physical record to be read (may be indirect). If not specified, the default value is BKL.

PTR, [*]adr-3 specifies the address, adr-3 (optionally indirect), of a pointer word into which the system is to return the address of the event control block (ECB) associated with this I/O request.

USAGE RULES

1. The TRL value may never exceed the effective value of the MXL parameter in the referenced DCB.

2. TRL values less than, equal to, or greater than block length (BKL) are valid.

3. Use of the PTR option causes the system to supply, on return from the procedure, a pointer to the ECB associated with the specific request and the implied I/O operation. This pointer must then be used in a subsequent M:CHECK procedure to identify the operation to be checked.

4. An abnormal or error return, if selected, for a given I/O operation will be made only from an M:CHECK procedure referring to that operation.

5. If two or more I/O requests have been queued (i.e., accepted by the system for execution) and the nth implied operation terminates unsuccessfully, and the system attempts to initiate the nth + 1 implied operation before an M:CHECK referring to the nth operation is executed, the nth + 1 and all following requests will be purged.

6. If the number of queued I/O requests, including requests for which I/O operations have been completed but not checked, exceeds the value of the SIM parameter (in the DCB), a program abort will occur.

7. If a read operation is attempted outside the limits of the file, an end-of-file return is made from the corresponding M:CHECK procedure.

8. No order is imposed on M:READ/M:WRITE sequences.

M:WRITE Write Virtual Physical Record

The M:WRITE procedure requests writing of a program determined physical record to a file on direct access media. All write requests must specify the relative block number of the first (or only) block to be written. The blocks of the file can be written (or rewritten) in any order.

The length of the record to be written either is specified in the procedure (TRL parameter) or defaults to the block length (BKL). In either case, it is limited by the maximum transfer length (MXL) specified in the DCB. Physical record transfers may extend across block boundaries, or may be shorter than block length. (For modification of existing permanent

files, the value BKL is determined from the file's HDR2 label.) Records are transmitted to the physical file directly from the user's buffer area specified in the procedure.

Return from the procedure is made immediately following the system's acceptance of the request (not after initiation or completion of the I/O operation). Successive I/O requests (via the same DCB) may be issued, prior to execution of M:CHECK and/or M:WAIT procedures referring to these requests, up to the limit specified by the SIM parameter of the DCB.

Syntax: See foldout Chart A-7 in Appendix A.

MEANING OF THE PARAMETERS

dcb-adr specifies the address of the DCB through which the write is to be performed.

BUF, {[*]badr-1 / adr-2} specifies either the initial byte address, badr-1 (optionally indirect), of the user's buffer area from which the record is to be written; or the word address, adr-2 (direct only), of the buffer area (which will be converted to a byte address).

ADR, [*]value-1 specifies the relative block number (where block 0 is the beginning of the file) of the block at which the write operation is to begin (may be indirect).

TRL, [*]value-2 specifies the number of bytes to be transmitted, i.e., the length of the physical record to be written (may be indirect). If not specified, the default value is BKL.

PTR, [*]adr-3 specifies the address adr-3 (optionally indirect), of a pointer word into which the system is to return the address of the event control block (ECB) associated with this I/O request.

USAGE RULES

1. The TRL value may never exceed the effective value of the MXL parameter in the referenced DCB.

2. TRL values less than, equal to, or greater than block length (BKL) are valid.

3. Use of the PTR option causes the system to supply, on return from procedure, a pointer to the ECB associated with the specific request and the implied I/O operation. This pointer must then be used in a subsequent M:CHECK procedure to identify the operation to be checked.

4. An abnormal or error return, if selected, for a given write operation will be made only from an M:CHECK procedure referring (explicitly or implicitly) to that operation, except as described in rule 5.

5. If a write operation is attempted outside the limits of the file, an X3 class abnormal return, code 09, is made from the M:WRITE procedure itself.

6. If two or more I/O requests have been queued (i.e., accepted by the system for execution) and the nth implied operation terminates unsuccessfully, and the system attempts to initiate the nth + 1 implied operation before an M:CHECK referring to the nth operation is executed, the nth + 1 and all following requests will be purged.

7. If the number of queued I/O requests, including requests for which I/O operations have been completed but not checked, exceeds the value of the SIM parameter (in the DCB), a program abort will occur.

8. All write operations begin on a block boundary.

9. Rewriting of a record within the limits of an existing file does not truncate the file at the end of the new record. More than one originally existing record can be overwritten (intentionally or inadvertently), however, depending upon relative lengths of the original and replacement record. Proper one-for-one record replacement, if desired, is the user responsibility.

10. No logical record structures are provided for or recognized either as existing in the file or indicated in the DCB.

M:CHECK Check I/O Event Completion

The M:CHECK procedure tests a given I/O operation for proper completion, placing the issuing program in a wait state, if necessary, to await such completion. If the tested operation terminates unsuccessfully, an appropriate abnormal or error return is made to the user's program (if selected in the user's program; abort otherwise). Optionally, for a read operation the actual length of the record (or partial transmission) read is reported.

The specific operation to be checked may be indicated in either of two modes:

- Explicitly, in the procedure (PTR parameter), by means of a pointer to the event control block (ECB) associated with the operation (obtained via the corresponding M:READ or M:WRITE request).

- Implicitly, (no pointer to the DCB) as the operation corresponding to the oldest outstanding (i.e., yet unchecked) I/O request.

The program must use one of these two modes exclusively for all operations via a given DCB.

The M:CHECK procedure may be used in conjunction with the M:WAIT procedure, described in Chapter 8.

Syntax: See foldout Chart A-7 in Appendix A.

MEANING OF THE PARAMETERS

dcb-adr specifies the address of the DCB through which the check is to be performed.

PTR, [*]adr-1 specifies the address, optionally indirect, of a pointer to the ECB associated with the operation to be checked (see M:READ and M:WRITE).

RSA, [*]adr-2 specifies the address, optionally indirect, of a word into which the system is to return the actual length (in bytes) of the I/O transmission in the case of a read operation.

USAGE RULES

1. Between the initial opening and the definite closing of a DCB, all M:CHECK procedures referring to the DCB must either always specify the PTR parameter or always omit that parameter.

2. If the PTR parameter is not specified, the earliest requested, still unchecked operation is checked.

3. If the RSA parameter is specified for a write operation, it is ignored.

4. The M:CHECK procedure, when executed, effectively releases the system table (IOB) entry associated with a given operation, allowing it to be used for queuing of subsequent requests.

BDAM (BASIC DIRECT ACCESS METHOD)

The BDAM access method permits reading and writing on a private direct access volume by reference to relative disk sector addresses. No record, block, or file level structures are provided for or recognized. The disk volume must be assigned with a DEV type assignment; the logical file is equated to the entire storage volume. System disk cannot be accessed with this method.

GENERAL USAGE RULES

The general usage rules for BDAM are listed below.

1. Applicable I/O procedures:

   M:READ

   M:WRITE

   M:CHECK

   M:CVOL

2. Usage of all procedures is the same as for VDAM except as differentiated by the following rules. See fold-out Chart A-8 of Appendix A for M:READ, M:WRITE, and M:CHECK syntax. M:CVOL usage is the same as for VSAM.

3. Applicable DCB parameters:

   AM, BD (Basic Direct)

   ERR

   BKL

   MXL

   NRT

   SIM

4. Requires that the mandatory ADR parameter of both M:READ and M:WRITE specify a disk sector by relative sector number.

5. All reads and writes begin on a sector boundary (hardware determined).

6. Applicable processing modes as defined, implicitly, for VDAM:

   Input

   Output

   Scratch

PROCESSING OF ABNORMAL AND ERROR CONDITIONS

The normal execution of an I/O procedure, or of a file opening or closing procedure, may be precluded by the occurrence of any one of a large number of abnormal or error conditions, including:

1. Programming errors, e.g., the record length specified by an M:PUT request being greater than the block length appearing in the DCB.

2. Job initialization errors, e.g., unassigned operational label, or file status incompatibility with processing mode.

3. Abnormalities in the file content from the procedure's point of view, e.g., end-of-volume, or specified record key not existing in the file.

4. Device related errors, e.g., device locked by operator.

5. Errors in transmission between memory and device during a physical I/O operation.

Some of these conditions, primarily instances of categories 1 and 2 described above, always cause the executing program to be aborted. These are referred to as unconditional abort conditions. An abort code identifies the specific type of abort condition on the job control file.

Many other conditions, including all abnormal conditions, device and transmission errors, and many programming errors can be handled by the user's program at his discretion. They are conditional program aborts in that they also cause the program to be aborted if no user routine has been provided to process the indicated condition. The presence of an ABN and/or ERR address in the DCB, and selection of one or more classes of abnormal condition, by the Xn class codes in the ABN parameters (see M:DCB), indicates that the user has provided abnormal and/or error handling routines.

The absence of either the ABN and/or ERR addresses, or lack of X1, X2, or X3 class codes causes the program to be aborted if the corresponding conditions occur.

The absence of the X4, X5, or X6 codes relating to user label processing, simply inhibits the corresponding abnormal condition entry to the user's routine (if any) during label processing. See Appendix D for full information on user label processing.

ABNORMAL AND ERROR HANDLING ROUTINES

An abnormal condition or error can cause a user provided routine to be entered, as explained above. The address of these routines are specified in the DCB via the ABN and ERR parameters, respectively.

At the time of entry to either of these routines, certain registers have been set by the system in the following manner:

R5 Abnormal or error code (leftmost byte). Return address (right-justified).

R6 DCB address (right-justified).

R7 Event control block (ECB) address in the case of an abnormality or error detected by M:CHECK.

Address of the key in the case of an abnormal condition detected during an access by key of an indexed sequential file.

Not significant in other cases.

The only I/O procedures that can be executed in an abnormal or error routine are M:CLOSE, M:CVOL, and M:CHECK. Such a routine must exit by an M:RETURN procedure.

The exit from the user's routine is a special return when register 6 is set to zero before the execution of the M:RETURN procedure. It is a normal return in any other case (R6 $\neq$ 0).

ABNORMAL CONDITIONS DURING OPEN AND CLOSE

The abnormal conditions that can occur during M:OPEN/M:CLOSE or end-of-volume processing are selected by the X1, X4, X5, or X6 class codes at the time the ABN address is specified in the DCB via M:DCB or M:SETDCB. Class codes X4, X5, and X6 refer only to user label processing (see Appendix D).

Occurrence of such a condition causes a transfer of control to the specified ABN address, assuming that the corresponding class code has been selected (abort otherwise in the case of X1 class only).

For each of these conditions, Table 7-3 describes the specific abnormal code returned, the controlling ABN class code, the cause, origin, occurrence condition, and result of a normal or special return from the user's abnormal routine.

Table 7-3. Abnormal Conditions on M:OPEN and M:CLOSE

ABN Condition Class	ABN Code	Cause	Occurrence Condition	Origin	Result on Return from User Routine	
					Normal	Special
X1	01	End-of-file (DUM assignment).	Mode I, B, U, or S.	M:OPEN	DCB not opened.	DCB not opened.
X1	02	End-of-volume.	MT only.	M:OPEN	DCB not opened.	With PID and I mode, there is a rewind and the search is continued on same volume (or on the following volume if M:CVOL was executed in user routine). In other cases, M:CVOL must be executed in user routine to permit continuation on the following volume.

Table 7-3. Abnormal Conditions on M:OPEN and M:CLOSE (cont.)

ABN Condition Class	ABN Code	Cause	Occurrence Condition	Origin	Result on Return from User Routine	
					Normal	Special
X1	0A	No more volumes to switch.	MT or serially mounted disk.	M:CVOL within open abnormal routine.	DCB not opened.	DCB not opened.
X1	14	File not found: I, B, O, or U, but not MOD, O.	MT or disk; status OLD or MOD, mode I, B, O, or U.	M:OPEN	DCB not opened.	For disk files, the DCB is not opened. For tape files, M:CVOL procedure must be executed in the user routine to permit continuation on the following volume.
X1	15	File already exists.	MT or disk; status NEW; mode O or S; CID or PID.	M:OPEN	DCB not opened.	The old file is overwritten if it has expired; if not yet expired, see code 18 below.
X1	16	File identification does not match.	MT only; OLD or MOD; I, B, O, or S; CID.	M:OPEN	DCB not opened.	Corresponding file is opened to be read, extended, or overwritten.
X1	18	File not expired.	MT or disk; NEW or OLD; O or S; CID, NID, or PID.	M:OPEN	DCB not opened.	File is opened for input, extension, or overwriting as appropriate, depending on status and mode.
X1	19	Password requested.	MT or disk; any status and mode.	M:OPEN	The contents of registers 6 and 7 are compared with the password of the file being read or placed into the entry in the label for a file being created.	Same as normal.
X4	10	Exit for beginning-of-file user labels.	MT only.	M:OPEN	No read or write of user label has to be done.	New user label must be read or written.
X6	11	Exit for beginning-of-volume user labels.	MT only.	M:OPEN	No read or write of user label has to be done.	New user label must be read or written.
X5	12	Exit for end-of-file or end-of-volume user labels.	MT only.	M:CLOSE	No read or write of user label has to be done.	New user label must be read or written.

ERROR CONDITIONS DURING FILE OPENING

Several error conditions can occur during M:OPEN processing. Selection of these conditions for user handling is by specification of an ERR address in the DCB (prior to M:OPEN execution). Occurrence of any of these conditions causes a transfer of control to the specified ERR address.

Table 7-4 describes these error conditions.

ABNORMAL CONDITIONS DURING FILE PROCESSING

Among the abnormal conditions that may arise subsequent to successful opening of the file, the end-of-file and end-of-volume conditions, selected by ABN class code X2, can be distinguished from other miscellaneous processing abnormalities selected by class code X3.

For each of these conditions, Table 7-5 describes the abnormal code returned, the controlling class code, cause, origin, and result of a normal or special return from the user's abnormal routine.

ERROR CONDITIONS DURING FILE PROCESSING

The error conditions that can occur subsequent to successful opening of the file are related to either data errors, too many queued I/O requests, or physical device malfunctions. Entry to the user's routine for these conditions is selected by specification of the ERR address in the DCB.

For each error condition, Table 7-6 describes the error code returned, the cause, origin, and result of a normal or special return from the user's error routine.

ERRORS CAUSING THE PROGRAM TO ABORT

All the error or abnormal codes described previously are also possible abort codes if no routine was provided in the user program to process the corresponding conditions. The specific unconditional abort codes described in Appendix B can also accompany an abnormal end-of-job or end-of-task. If the numeric portion of the abort code is less than X'80', which is the case for all of the abort codes related to file management, the following steps in the job can be executed.

An abort always causes a message to be printed on the job control file. This message has the following format:

ABORT	label	symbolic code	program state when abort condition was detected

where label indicates the operational label of the file being processed when the abort condition was detected.

Table 7-4. Error Conditions on M:OPEN

ERR Code	Cause	Origin	Result on Return from User Routine
23	Permanent device error.	M:OPEN	DCB not opened.
2A	Device locked by operator	M:OPEN	DCB not opened.
2C	Irrecoverable permanent error.	M:OPEN	DCB not opened.

Table 7-5. Abnormal Conditions During Processing

ABN Condition Class	ABN Code	Cause	Origin	Result (Normal and Special Return)
X2	01	End-of-file.	M:CHECK M:GET M:CVOL	The DCB is no longer open except for magnetic tapes with DEV assignment accessed by VSAM.
X2	02	End-of-volume.	M:CHECK	The DCB remains open only if the M:CVOL procedure has been executed in the user routine processing this abnormality.
X3	07	Read length less than tape block length.	M:CHECK M:GET (U format)	DCB remains open, processing may continue.

Table 7-5. Abnormal Conditions During Processing (cont.)

ABN Condition Class	ABN Code	Cause	Origin	Result (Normal and Special Return)
X3	09	Attempt to access outside file limits.	M:WRITE (ADR) M:DEVICE (BKS/FWS) M:POINT	Normal: DCB closed; no further processing allowed. Special: File extended by secondary increment; processing may continue.
X3	13	More than 23 extents required (disk).	M:PUT M:WRITE	DCB is no longer open.
X3	0A	File is full or no more volumes to switch.	M:PUT M:WRITE M:CVOL	The DCB is no longer open if there are no more volumes to be switched.
X3	0B	Key out of sequence.	M:PUT (indexed in O mode)	DCB remains open, processing may continue.
X3	0D	Key not found.	M:FIND M:STOW (DEL) M:GET (KEY)	DCB remains open, processing may continue.
X3	0F	Key already exists.	M:STOW (ADD, SYN) M:PUT (indexed)	DCB remains open, processing may continue.

Table 7-6. Error Conditions During Processing

ERR Code	Cause	Origin	Result on Exit	
			Normal	Special
20	Sequence break in the block numbers on magnetic tape.	M:GET M:CHECK	DCB no longer open.	Error condition is ignored, processing can continue.
21	Wrong length record.	M:GET (F or V format).	DCB no longer open.	Error condition is ignored, processing can continue.
22	Queued I/O request purged due to bad termination of prior operations.	M:CHECK	DCB open.	DCB open.
23	Permanent device error.	All I/O procedures except M:READ and M:WRITE.	DCB no longer open.	Error conditions is ignored, processing can continue.
2A	Device locked.	All I/O procedures except M:READ and M:WRITE.	DCB no longer open, processing cannot continue on the device.	DCB no longer open, processing cannot continue on the device.
2C	Irrecoverable permanent error.	All I/O procedures except M:READ and M:WRITE.	DCB no longer open, processing cannot continue on the device.	DCB no longer open, processing cannot continue on the device.

8. SYSTEM SERVICES

INTRODUCTION

The XOS monitor provides the user with an extensive set of execution-time programming services collectively called system services. They include facilities that allow the user to dynamically

- Obtain and release working storage space.
- Load and execute a load module with or without returning to the calling module.
- Load an overlay segment.
- Manage CPU-detected program abnormalities (traps).
- Test and modify the Job Switch Word (a set of control-command-initialized switches used primarily for communication between job steps).
- Communicate with the system operator.
- Interrogate the system clock.
- Set, suspend, and process clock interrupts within a job step.
- Specify and manage certain event-triggered interrupts (e.g., I/O completion).
- Specify job step termination in a normal or error state.

A user could invoke a specific system service by executing a CALl instruction containing a code that identifies the specific system service desired. Typically, when the CALl is executed, the user also would make available to the Monitor a set of parameter specifications in a form required by the Monitor for the specific service requested.

The detailed coding of system service requests is often extensive and complex, however. Therefore, a set of Meta-Symbol system procedures (included in the SYSTEM XOS procedure library) is provided and described herein. These procedures interpret a given user source statement as a functional-level request for a system service. The functional parameters are then translated into the detailed interface code required by the Monitor to request the performance of that function.

CONVENTIONS

In order to use any of the procedures contained in SYSTEM XOS, it is necessary that the first such use (in statement number order) in any Meta-Symbol assembly be preceded by a source statement containing the command SYSTEM with the argument XOS. That is, the statement

```
SYSTEM  XOS
```

must precede the first system procedure reference.

SYNTAX

Rules for referencing an XOS system procedure are a subset of the general rules for referencing any Meta-Symbol procedure. The subset is bounded by

```
[label(s)]    M:command    [parameter list]
```

where

label(s) is the symbol or list of symbols (up to 255) to be defined as the address of the first executable instruction generated by the procedure.

M:command is the name of the system procedure being referenced. By convention these begin with M: (letter M and character colon).

parameter list is a set of functional parameters that specify the manner in which the requested service is to be performed.

Refer to the Meta-Symbol/LN, OPS Reference Manual (90 09 52) for the complete general format of a procedure call. If the asterisk character is used, indirect addressing is indicated, i.e., the symbol or value following the asterisk specifies the location of a value rather than the value itself. No space is permitted between the asterisk and the character that follows. If brackets are used the items contained by them are optional. Discussions within this chapter usually refer to a general register symbolically according to the following notation:

Register	Symbol	Register	Symbol
0	R0	8	R8
1	R1	9	R9
2	R2	10	R10
3	R3	11	R11
4	R4	12	R12
5	R5	13	R13
6	R6	14	R14
7	R7	15	R15

MEMORY MANAGEMENT

XOS utilizes the Sigma memory map option; user jobs are executed in virtual memory. All memory addresses used by a user job are interpreted through the memory map. Memory mapping is explained in the appropriate Sigma Computer Reference Manual for Sigma 6, 7, or 9 computers and discussed in Chapter 1 of this manual.

Most users need not be concerned with virtual memory and memory mapping. XOS performs the mapping function in a way transparent to the user. Consequently, virtual memory appears to the user as a single contiguous set of actual memory locations. Only the master-mode user need be concerned with the distinction between operation in virtual vs. physical memory.

The usable memory space available to a given job step extends over N pages of memory, where N is the greater of the values shown on the !LIMIT or !SLIMIT control commands. If neither of these commands is used, the default value specified at system generation is used. If only P pages of memory are actually being used by the job step, the remaining (N-P) pages constitute the dynamic space (or area) available to the program. The dynamic space is only allocated to the program upon explicit request; any attempt to access an address in the dynamic space without its previous allocation causes the job step to trap with a memory-protection violation.

ORGANIZATION OF MEMORY SPACE

A user's program space is divided into two areas

- Program Area
- Dynamic Area

Figure 8-1 diagrams a user's virtual memory. See the appropriate Sigma Computer Reference Manual for a discussion of memory access protection.

PROGRAM AREA

The program area of a job consists of three portions:

1. Variable Data Area. The access protection type for this area is 00 (all access). This area contains
   - Task Control Block (TCB). This is a system table accessible by the program and contains control information.
   - FORTRAN blank common.
   - The (variable) data of the program itself. In the case of a segmented program, this area is divided into several parts corresponding to the root and the different overlays.
2. Instruction Area. The access protection type for this area is 01 (read or execute only). This area contains
   - Data Control Blocks (DCBs) for the program.
   - RFLOAD table. This is a table of external references created by the Link Editor to enable the implicit loading of overlay segments when they are referenced at execution time (see REF option in Chapter 4 of this manual).
   - The program (instructions). In the case of an overlay program, this area is divided into several parts corresponding in size and location to the root and its various overlay segments.
3. Constant Data Area. The access protection type for this area is 10 (read only). In case of an overlay program, this area is broken into several parts corresponding in size and location to the root and different overlay segments' constant data areas.

Since the XOS (Sigma 6, 7, 9) memory access protection feature operates on page (512-word) units, each of the areas defined above begins on a page boundary.

Note: Variable data, constant data, and instruction areas are necessarily distinguished on the basis of user-defined control section protection types. See CSECT and DSECT commands in Meta-Symbol/LN, OPS Reference Manual, 90 09 52.

DYNAMIC AREA

The dynamic area ranges from the first whole page following the constant data area to the last page of memory assigned to the job (see !LIMIT and !SLIMIT commands). This area contains

1. Unallocated Dynamic Area. The access protection type for most of this area is 11 (no access). Initially, i.e. at job initiation, this area includes the whole dynamic area. Later it may contain a certain number of words in pages also containing words allocated in the common dynamic area. Those portions of the unallocated dynamic area contained in such pages have the same access protection type as the common dynamic area.

   As the user job requests the allocation or deallocation of local or common dynamic area, the extent of the unallocated dynamic area diminishes or increases so the sum of the three areas (LOCAL, COMMON, and UNALLOCATED) remains constant during a job step. The size of the unallocated dynamic area, in contiguous whole pages between the highest address in the local dynamic area and the lowest address in the common dynamic areas is available to the user job via the M:GL system service.
2. Local Dynamic Area. The access protection type for this area is 00 (all access). It begins at the first page of the dynamic area and extends in the direction of

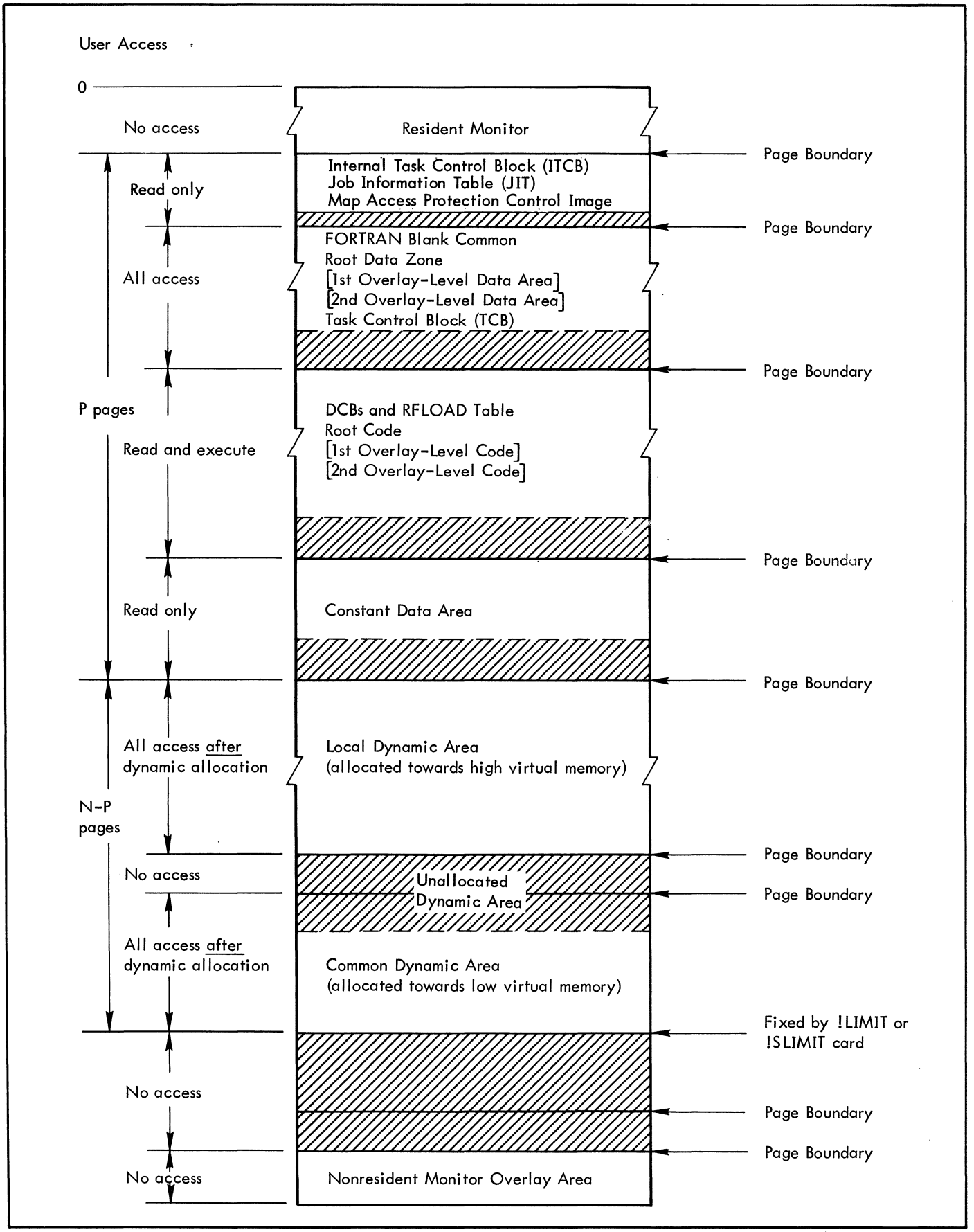


Figure 8-1. User's Virtual Memory

increasing addresses. Allocation is performed in multiples of a page (512 words). See discussions of M:GP and M:FP for details of allocation control.

3. Common Dynamic Area. The access protection type for this area is 00 (all access). It begins at the highest address of the last page of the dynamic area and extends in the direction of decreasing addresses. This area is called common because it is shared by the user and the system. It is in this area that the system stores necessary tables for processing the job and the user buffers for the assisted access methods. Allocation is performed in multiples of four words. See discussions of M:GSP and M:FSP for details of allocation control.

SPACE ALLOCATION PROCEDURES

M:GL Get Limits of Dynamic Space

The M:GL procedure allows the user to obtain the number of contiguous unallocated whole pages between the highest address in the local dynamic and the lowest address in the common dynamic areas.

Syntax

[label(s)] M:GL

The system returns, in register R5, the number of pages currently in the unallocated dynamic area.

M:GP Get Pages in the Local Dynamic Area

The M:GP procedure permits the user to request the allocation of a specified number of pages in the local dynamic area.

Syntax

[label(s)] M:GP [*]n

where n is an integer that specifies the number of pages requested in the local dynamic area.

If the number of pages requested is available, they are allocated, the address of the first word in the allocated area is placed in R5, and the condition code (CC1) is reset to 0. If the number of pages requested is not available, no pages are allocated and the condition code (CC1) is set to 1.

The LOCAL dynamic area may be viewed (by the user) as a set of contiguous whole pages. Whenever one or more pages are allocated they may be thought of as extending the upper bound (or increasing the highest available address) of the contiguous set of whole pages forming the LOCAL dynamic area. When one or more pages are freed (deallocated) the highest available address is reduced by some multiple of 512 words thereby lowering the upper bound of the contiguous set of whole pages forming the LOCAL dynamic area.

M:FP Free Pages in the Local Dynamic Area

The M:FP procedure allows the user to free (deallocate) a specified number of previously allocated pages in the local dynamic area.

Syntax

[label(s)] M:FP [*]n

where n is an integer that specifies the number of pages to be freed in the local dynamic area.

If the number of pages to be freed is less than or equal to the number of pages currently allocated in the local dynamic area, they are freed and the condition code (CC1) is reset to 0.

If the number of pages to be freed is greater than the number of pages currently allocated to the user in the local dynamic area, no pages are freed and the condition code (CC1) is set to 1.

The pages are freed in order of decreasing addresses beginning with the highest address of the currently allocated pages.

M:GSP Get Space in the Common Dynamic Area

The M:GSP procedure allows the user to request the allocation of a block of specified size in the common dynamic area.

Syntax

[label(s)] M:GSP [*]n

where n is an integer that specifies the number of words requested in the common dynamic area. If n is not a multiple of four, it is automatically rounded up to the next multiple of four (e.g., if 37 is specified, 40 is assumed).

If the space requested is available, it is allocated. The lowest address of the allocated block is placed in R5 and the condition code (CC1) is reset to 0.

If the space requested is not available, no space is allocated and the condition code (CC1) is set to 1.

Blocks allocated by two successive requests are not necessarily contiguous since the system may have requested and received space between two successive user requests.

M:FSP Free Space in the Common Dynamic Area

The M:FSP procedure allows the user to free a previously allocated block of space in the common dynamic area.

Syntax

[label(s)] M:FSP [*]n,adr

where

n is an integer that specifies the number of words to free in the common dynamic area.

adr is the address of a memory word or register containing the first address of the previously allocated block of space to be freed.

If the exact block of space to be freed had previously been obtained by the M:GSP procedure, the block is freed and the condition code (CC1) is reset to 0.

If the exact block of space to be freed had not previously been obtained by the M:GSP procedure, the block is not freed and the condition code (CC1) is set to 1.

DYNAMIC OVERLAY AND PROGRAM LOADING

The services described in this section permit a user to explicitly request

1. The loading into memory of a specified overlay segment.
2. The loading into memory of, and transfer of control to, a specified separate load module.

M:SEGLD Loading of an Overlay Segment

The M:SEGLD procedure allows a user to bring a specified overlay segment into memory. See Tree Structure in Chapter 4 of this manual for a description of tree structures and overlays.

Syntax

[label(s)] M:SEGLD [*]address

where address is the word address of a string of characters giving the name of the segment to be loaded. The first byte of the string of characters must specify the number of characters in the string (see TEXTC directive of Meta-Symbol).

The specified segment is loaded as well as all those not already loaded which lie on the path of the tree between the calling segment and the specified segment.

Before using the M:SEGLD procedure, it is necessary that all I/O operations that have been requested by the calling segment or any higher level overlay be completed.

The programmer should take the precaution of verifying (e.g., via M:CHECK) that all requested I/O is completed. If any I/O is still in progress the results of an M:SEGLD are unpredictable.

Once the M:SEGLD is completed, the user may reference or execute it just like any previously core-resident section of his program. Normally, symbolic references in accordance with the Meta-Symbol REF/DEF specifications are used for intersegment communication.

M:LDTRC Loading and Executing a Program without Preserving the Calling Program

The M:LDTRC procedure allows an executing program to dynamically request the loading into memory of, and transfer of control to another program without preserving the calling program.

Syntax

[label(s)] M:LDTRC [*]address

where address is the address of a memory word containing the operational label of the file containing the load module of the desired program. The operational label consists of up to four alphanumeric characters, left justified and blank filled (X'40').

The memory space occupied by the calling program and the space allocated to it in the local dynamic area are freed. The common dynamic area is unchanged, thereby permitting module-to-module communication. The called load module is loaded into memory (replacing the calling program) and activated at the start address specified at the time it was formed.

The state of the registers upon entering the called program are as follows:

Register	Contents
R0	Address of the stack-pointer doubleword for the user's temporary stack.
R1	Number of whole contiguous pages in the unallocated dynamic area.
R2	X'80000000' merged with the word address of the options specified with the !RUN command which began this step. (Starting address of string of characters in Meta-Symbol TEXTC format).
R3-R15	Their existing state prior to the execution of M:LDTRC.

In order to access the file containing the desired load module the user must have employed an !ASSIGN control command (see Chapter 3) specifying the same operational label that was specified by the procedure argument, or have otherwise assigned that label (possibly by default, e.g., LM).

Since the common dynamic area is not affected by the execution of this procedure, it can be used to transmit information from the calling program to the called program.

The M:LDTRC procedure results in a final exit of the calling program. In the same way as the M:RETURN procedure, it provides for the execution of a certain number of controls on the part of the system. The M:LDTRC procedure can be used only on the principal level of execution. That is, the M:LDTRC procedure cannot be used in a sequence of exception processing or asynchronous processing (e.g., user routines for abnormal returns from M:OPEN, M:CLOSE, M:CHECK, as well as routines associated with M:TRAP, M:INT, M:STIMER).

The system waits for the completion of any asynchronous processing (input/output operations, messages to the operator) before performing an M:LDTRC. The system closes any DCB's that are opened. If the M:STIMER, M:INT, or M:TRAP services are active, they are canceled.

The called load module may terminate its execution with an M:RETURN procedure just like any other load module or with another M:LDTRC procedure (initiation of a new called program)

M:LINK Loading and Executing a Program with the Preservation of the Calling Program

The M:LINK procedure allows an executing program to dynamically request the loading into memory of, and transfer of control to, another program while preserving the state of the calling program for a later return.

The called load module is loaded into memory and control is transferred to it at the start address specified at the time it was formed. If no transfer address is associated with the called load module, the job is aborted.

Syntax

[label(s)] M:LINK [*]address

where address is the address of a memory word containing the operational label of the file containing the load module of the desired program. The operational label consists of up to four alphanumeric characters, left justified and blank filled (X'40').

The contents of the memory space occupied by the calling program and the space allocated in the local dynamic area are copied onto the system disk in a temporary file, for later reactivation, and that space is then freed. The common dynamic area is unmodified, thereby permitting module-to-module communication.

The called program is transferred to memory and initiated at its entry address, the state of the registers upon entering the called program are as follows:

Register	Contents
R0	Address of the stack-pointer doubleword for the user's temporary stack.
R1	The number of whole contiguous pages in the unallocated dynamic area.
R2	X'80000000' merged with the word address of the options specified with the !RUN command which began this step (starting address of a string of characters in Meta-Symbol TEXTC format).
R3-R15	Their existing state when the M:LINK was executed.

In order to access the file containing the load module for the called program, the user must have employed an !ASSIGN control command (see Chapter 3) specifying the same operational label that was specified by procedure argument, or have otherwise assigned that label (possibly by default, e.g., LM). Since the common dynamic area is not affected by the execution of this procedure, it can be used to transmit information from the calling program to the called program.

The M:LINK procedure results in a conditional exit from the called program. In the same way as the M:RETURN procedure, it provides for the execution of a certain number of controls on the part of the system. The M:LINK procedure can only be used on the principal level of execution. That is, the M:LINK procedure cannot be used within a sequence of exception processing or asynchronous processing (e.g., user routines for abnormal returns from M:OPEN, M:CLOSE, M:CHECK, as well as routines associated with M:TRAP, M:INT, M:STIMER).

The system waits for the completion of any asynchronous processing (input/output operations, messages to the operator) before performing an M:LINK. The system closes any DCB's that are opened. If the M:STIMER, M:INT, or M:TRAP services are active, they are canceled.

It is possible to nest many M:LINK procedures. The return from a called program to its calling program is performed, in reverse order of calls, by an M:RETURN at the principal (nonasynchronous) level. The environment of the calling program is reinstated, except for the registers and common dynamic area which remain in the state of the called program. The temporary file used to save the calling program is released.

PROGRAM MANAGEMENT

The system services described in this section provide the user with dynamic control (within the constraints of the !LIMIT command) of CPU-time utilization and the condition (normal or error) of job step termination. Thereby, the user program can determine when it will execute, when it will wait before continuing execution, and when, how, and the reasons for which it will terminate execution.

PROGRAM INITIAL CONDITIONS

When a user program is activated, it begins execution without control over the handling of CPU-detected abnormal conditions (abnormal traps). If an abnormal trap occurs, while in this condition, the monitor aborts the job or job step in accordance with the status of bit 0 of the Job Switch Word (JSW). At program initiation, the contents of the general registers are as follows:

Register	Contents
R0	The word address of the stack-pointer doubleword for the user's temporary stack.
R1	The number of whole contiguous pages in the unallocated dynamic area.
R2	The word address where the !RUN command options are stored in Meta-Symbol TEXTC format. If the keyword OPTION is not specified on the !RUN command, this register is zero.
R3-R15	Unpredictable.

CPU-DETECTED PROGRAM ABNORMAL CONDITIONS (ABNORMAL TRAPS)

The Sigma hardware system is capable of diagnosing those program abnormal conditions resulting from the attempted execution of instructions that are in some way incompatible with their hardware environments. Such instructions are categorized as follows:

1. Undefined or implemented instructions, or privileged instructions being executed in the slave (normal XOS user) mode.
2. Instructions referencing nonexistent memory addresses or in some way attempting to use a part of memory in a way not consistent with its protection type.
3. Instructions containing an operand conflict such that execution of the instruction causes a stack overflow or an arithmetic fault.

Certain CAL instructions (all of which are programmed traps), are treated as abnormal traps because they result in undefined branches into the monitor. CAL1 instructions, which are defined by the system, result in normal traps to the monitor.

M:TRAP Trap Management

The M:TRAP service enables an executing user program to specify that it be allowed to handle certain CPU-detected program abnormal conditions.

While additional capabilities are provided, there are three basic occurrences that must be understood by a user before attempting to handle his own traps. They are as follows:

1. Request for M:TRAP service. The executing program must inform the system (in advance) that if any of the specified abnormal traps occur, the program is to be placed in the "active trap" state. It then is allowed to resume execution at a location specified in the M:TRAP request.
2. Occurrence of abnormal trap. When the user program executes an instruction resulting in an abnormal trap, the system immediately obtains control and determines if user handling of that specific trap has been requested. If not, the job or job step (depending on bit 0 of the JSW) is aborted. If user handling has been requested, via M:TRAP, the system:
   a. Saves the user's PSD and registers.
   b. Places the program in the "active trap" state.
   c. Returns control to the user at the location specified in the M:TRAP request.
3. Termination of user trap processing (M:RETURN). When the user program finishes its trap processing, it must either abort the job step (or job) or request the system to clear the "active trap" condition and allow it to return to normal processing. An abort can be effected by requesting the M:ERR service (see M:ERR). The "active trap" can be cleared and the program returned to normal execution by requesting the M:RETURN service (see M:RETURN).

Syntax

There are two syntactic forms:

[label(s)] M:TRAP [adr1,(TRAP,spec[,...])

$$[,(\text{OST},[*]\text{adr2})]]\left[,(\text{IGNORE},\left\{\begin{matrix}\text{FX}\\\text{DEC}\\\text{BOTH}\end{matrix}\right\})\right]$$

[label(s)] M:TRAP (RESTORE,[*]adr3)

where

adr1 is the entry address of the user's routine to be used for processing the traps specified with the TRAP option. This address must be specified if, and only if, TRAP is specified.

TRAP is the keyword that introduces the list of traps for which the system is to give control to the user's program. If specified, adr1 must also be specified.

spec is a mnemonic listed below corresponding to a given trap:

PS	Stack overflow.
UI	Unimplemented instruction.
NI	Nonexistent instruction.
NMA	Nonexistent memory address.
PSM	Privileged instruction in slave mode.
MPV	Memory protection violation.
FP	Floating-point fault.
DEC	Decimal arithmetic fault.
FX	Fixed-point arithmetic fault.
CL2	CAL2 instruction.
CL3	CAL3 instruction.
CL4	CAL4 instruction.
NAO	Nonallowed operation (i.e., all of MPV, PSM, NMA, and NI).
ALL	All traps listed above.

OST specifies that a copy of the current abnormal condition processing specifications is to be saved prior to their modification by the current M:TRAP.

adr2 is the location into which the system is to place a pointer to the information saved by OST.

IGNORE is the keyword that introduces the arithmetic faults that are to be ignored.

FX is the keyword that specifies that the fixed-point arithmetic fault is to be ignored.

DEC is the keyword that specifies that the decimal arithmetic fault is to be ignored.

BOTH is the keyword that specifies that both the fixed-point and decimal arithmetic faults are to be ignored.

RESTORE specifies that a set of abnormal condition processing specifications, that were previously saved using OST, are to be reinstated.

adr3 is the location from which the system is to obtain a pointer to a set of specifications previously saved using OST.

USAGE CONSIDERATIONS

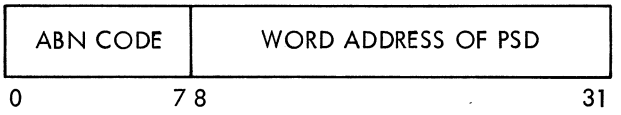
The execution of an M:TRAP request does not disturb the user's registers; his condition codes are modified, however.

TRAP Option.

1. At M:TRAP execution time, the user informs the system which abnormal traps he wants to handle. He then specifies the entry point of his trap handling routine.

2. At trap time, the system saves the user's PSD and general registers for later restoration (at M:RETURN time). Then it places the program in the "active trap" state, changes the contents of R5-R8, and returns to the user at the address specified with the TRAP option.

   When the system transfers control to the user's trap handling routine the contents of R5 are:



   The first byte is an abnormal code that identifies which abnormal trap has occurred. The last three bytes contain the word address of an image of the PSD that was active when the trap occurred. The address field of the PSD image points to the instruction whose execution was trapped.

   The legal abnormal codes are:

X'38'	Nonexistent instruction.
X'3A'	Privileged and nonexistent instruction.
X'3B'	Nonexistent instruction and memory protection violation.
X'3C'	Nonexistent address.
X'3E'	Privileged instruction.
X'3F'	Memory protection violation.
X'41'	Unimplemented instruction.
X'42'	Stack overflow.
X'43'	Fixed-point overflow.
X'44'	Floating-point overflow.
X'45'	Decimal arithmetic fault.
X'49'	CAL2 instruction.
X'4A'	CAL3 instruction.
X'4B'	CAL4 instruction.

If the user's trap handling routine needs to change the contents of the registers that are to be restored at M:RETURN time, it must make those changes to the appropriate register save locations. These save locations may be found as follows:

R0-R4 and R9-R15. Assuming the user has already found the saved PSD (using the pointer returned in R5), he can find R0 through R15 saved, in order, in the next 16 memory locations. However, the R5-R8 saved here have been changed and are not the images that will be restored.

R5-R8. If it is necessary to adjust the contents of R5-R8, their save locations must be found immediately upon initiation of the user's trap handling routine. At that time, R5-R8 are the last four words on the user's temporary stack. Hence, he must have saved, at program start-up time, the address of the stack's stack-pointer doubleword that was initially presented in R0. By interpreting that stack-pointer doubleword, he can locate the four saved registers in the user's program stack.

3. At M:RETURN time, the user's trap handler requests the M:RETURN service to clear the active trap and reinstate the user's PSD and general registers that were saved at trap time. Reinstatement of the PSD implies a transfer of control to the user program beginning with the previously trapped instruction. If the executed M:RETURN request specifies an address, control is then transferred to that location. If the user has adjusted any of the saved registers as described in item 2 above, the reinstated registers reflect those adjustments. As an alternative, the user can request the M:ERR service to abort his program's execution.

IGNORE Option. At M:TRAP time the user requests the system to reset the fixed-point arithmetic mask (AM) to zero and/or reset the decimal mask (DM) bits in the PSD. By default, every M:TRAP requests that these bits be set to 1 unless the IGNORE option is used to specify otherwise. The IGNORE option precludes the occurrence of the specified traps.

OST Option.

1. At M:TRAP time, before implementing the concurrently specified TRAP and IGNORE specifications, the system saves a copy of the currently active specifications for those options in the user's memory.

2. At trap time, processing is according to the TRAP and IGNORE options also specified.

3. At M:RETURN time, processing is according to the TRAP and IGNORE options also specified.

RESTORE Option.

1. At M:TRAP time, the system reinstates the specifications previously saved by an OST option. In effect, the RESTORE option again requests the TRAP and IGNORE options that were in effect when the specifications were saved.

   The RESTORE option may designate one of many sets of specifications, each saved by a previous use of OST.

2. At trap time, processing is according to the reinstated TRAP and IGNORE options.

3. At M:RETURN time, processing is according to the reinstated TRAP and IGNORE options.

M:RETURN Return Control to the System

An executing user program can return control to the system in a normal (i.e., nonerrored) condition only by executing a request for the M:RETURN service. The system will, depending on the context of the request, either terminate the job step or return control to the executing user program.

With an Active Trap. An executing user program has an active trap if a trap specified in a current M:TRAP request has occurred and the user program has not requested the system to clear that trap. If a program with an active trap executes a request for the M:RETURN service, the system clears the trap and transfers control to the user-specified address. If no address is specified, control is returned to the instruction that was executing when the trap occurred. Hence, execution of an M:RETURN with an active trap is a request to clear the active trap and return control to the user. Simulated traps (i.e., M:STIMER, M:INT) and abnormal returns (i.e., from M:CHECK, M:READ, etc.) are to be viewed in the same way. Execution with an active trap is frequently referred to as abnormal processing.

Without an Active Trap. A program executing without an active trap is referred to as being at the main program level or at program-level 0. Execution of an M:RETURN without an active trap is interpreted as an explicit request for a normal (nonerrored) job step termination.

Syntax

[label(s)] M:RETURN [*]address

where address is meaningful only when there is an active trap. In this case it is the return address for the trapped sequence. If this information is omitted when there is an active trap, the operation of the trapped routine is resumed at the address contained in the PSD when the trap occurred.

If there is an active trap, i.e., in the case of abnormal processing, the system restores the registers to the condition

at the time of the trap and returns to the trapped sequence. Otherwise, the system terminates the job step. In particular:

1. It cancels M:STIMER and M:INT if active.

2. It permanently closes (M:CLOSE procedure with the RELEASE option) all DCB's which are still open or temporarily closed (M:CLOSE procedure with the HOLD option).

3. Initiates the next job step or job.

M:ERR Error Job Step

The M:ERR procedure is used to signal the system that the user has specified an abnormal job step termination.

Syntax

[label(s)] M:ERR [*]value

where value is an integer, ranging from 0 to X'FF', which is sent as the abort code with the prefix G to the user on the job control file. For example, X'7C' is reported as 'G7C'.

If bit zero 0 of the JSW (see "!SWITCH Command" in Chapter 3) is zero, the job is aborted. If this bit is a 1, only the job step in process is aborted and the system proceeds normally to the next job step.

M:WAIT Program Wait

The M:WAIT procedure allows the user to place his program in a wait state until one or more events occur. This is necessary when the user has initiated one or more asynchronous processes (input/output for example) and wishes to synchronize his program with the terminating process(es).

Syntax

[label(s)] M:WAIT p,[*]address$_1$ [,...,[*]address$_n$]

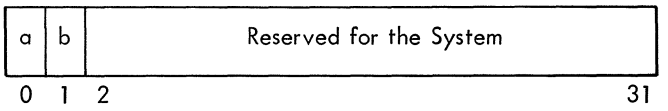
where

p is an integer specifying that p events of the n events indicated in the address list that follows must be completed before the system is to allow the program to regain control.

address is the address of a word containing the event control block (ECB) associated with an asynchronous process. A maximum of 16 ECB addresses may be specified in a single M:WAIT.

If the specified conditions are not fulfilled the program is placed in a wait state. When the specified conditions are fulfilled, control is returned to the program if its priority is greater than the program currently in execution.

To ensure system integrity, the ECB is in a read-only memory protection area; the user can only see what it contains. The ECB format follows:



The binary bits a and b indicate the event state:

a = 1 Event not completed.

b = 1 Event completed.

JOB SWITCH MANAGEMENT

Associated with each job is a Job Switch Word (JSW). Each of the 32 bits in this word is available to the user as a software managed switch (pseudo-switch). Except for bits 0 and 1, which have special meaning to the system, the user has complete freedom to set, reset, test, and associate meaning with the status of the bits in the JSW.

Special meanings of bits 0 and 1 for the system are

- If an abort occurs and bit 0 is set (on), job execution resumes with the next job step, if any. Otherwise, the entire job is aborted.

- If an abort occurs and bit 1 is set (on), a postmortem dump (PMD) of the user area is automatically taken, whether a :PMD command was or was not used (see Chapter 5).

Therefore, if a user runs a job with bits 0 and 1 both set, it is possible to have more than one job step abort with an associated PMD.

The user can define status of the JSW by using the !SWITCH command (see Chapter 3) or, except for bits 0 and 1, the system procedures M:SSS and M:RSS. He can test the status of bits in the JSW by using the UNLESS option of the !RUN and !EXEC commands (see Chapter 3) or the system procedure M:TSS.

Except for job initiation when all the bits are reset (0), the system does not modify the JSW except in accordance with an explicit user request. Consequently, the JSW is a device by which parameters may be communicated between job steps as well as between the job control language and and executing user program.

This section describes the use of M:SSS, M:RSS, and M:TSS system procedures.

M:SSS Setting of Pseudo-Switches

The M:SSS procedure allows an executing user program to specify that one or more bits of the JSW are to be set (turned on).

Syntax

[label(s)] M:SS $v_1[,v_2,\ldots,v_n]$

where v is an integer value, such that $2 \leq v \leq 31$, which is used to specify a bit position in the JSW.

The specified bits (pseudo-switches) are set. Bits not explicitly specified remain unchanged. The status of bits 0 and 1 may be specified only with the !SWITCH control command.

M:RSS Resetting of Pseudo-Switches

The M:RSS procedure allows an executing user program to specify that one or more bits of the JSW are to be reset (turned off).

Syntax

[label(s)] M:RSS $v_1[,v_2,\ldots,v_n]$

where v is an integer value, such that $2 \leq v \leq 31$, which is used to specify a bit position in the JSW.

The specified bits (pseudo-switches) are reset. Bits not explicitly specified remain unchanged. The status of bits 0 and 1 may be specified only with the !SWITCH control command.

M:TSS Test of Pseudo-Switches

The M:TSS procedure allows an executing user program to test the status of one or more bits of the JSW.

Syntax

[label(s)] M:TSS $v_1[,v_2,\ldots,v_n]$

where v is an integer value, such that $0 \leq v \leq 31$, which is used to specify a bit position in the JSW.

The status of the tested bits in the range 1-31 is returned in CC3. If any bit in this range is both specified and on, CC3 is also set to on. Otherwise, CC3 is reset to 0.

The status of bit 0 of the JSW is returned in CC4, whether requested or not. If it is on, CC4 is set to a 1. Otherwise, CC4 is reset to 0. (The settings of CC1 and CC2 are unspecified.)

CC3	CC4	Tested Condition of JSW
-	1	Bit 0 on whether specified or not
-	0	Bit 0 off whether specified or not

CC3	CC4	Tested Condition of JSW
1	-	At least one of bits 1-31 both specified and on
0	-	None of bits 1-31 both specified and on

In addition to setting the condition codes, M:TSS also returns the test detail word in R5. This detail word is a mask containing 1's for all bits except 0 that were both specified and on.

EXTERNAL COMMUNICATION

The procedures described in this section provide communication capabilities between the user and the operator.

M:KEYIN Message to the Operator With Reply

The M:KEYIN procedure permits a user program to display a message on the operator console and then wait for the operator to reply to the message.

Syntax

[label(s)] M:KEYIN (MESS,[*]adr1)

,(REPLY,[*]adr2),(SIZE,[*]n)

where

MESS is a keyword that introduces the message address.

adr1 is the word address of the starting location of the message to be sent to the operator. This message is a string of EBCDIC characters. The first byte specifies the total number of bytes in the message (legal range: 0-127). (See Meta-Symbol TEXTC directive.)

REPLY is a keyword that introduces the reply address.

adr2 is the address of the first word where the operator's reply is to be stored. When the reply is received, the first byte of this area indicates the total number of bytes in the reply (mandatory minimum of one byte).

SIZE is a keyword that introduces the maximum size of the reply.

n is a decimal integer indicating to the monitor the maximum number of characters to be allowed in the response (legal range: 1-31). The response will be truncated if necessary.

The message is sent to the operator console preceded by a message number (supplied by the system) and by the system ID of the job. The message and reply are also output to the

OUT symbiont file (see M:PRINT). The operator console then changes to the sending mode to accept the reply. The carriage-control and character-delete characters are intercepted by the system and not transmitted. The end of message is designated by the EOM character (X'80').

The operator may defer his response to this message just as in the case of system-issued messages that require a reply.

M:TYPE Message to the Operator

The M:TYPE procedure permits the user's program to send a message to the operator. In contrast to the M:KEYIN procedure, the monitor does not solicit a reply for the user program.

Syntax

[label(s)] M:TYPE (MESS,[*]adr)

where

MESS is the keyword that introduces the message address.

adr is the word address where the message to be sent to the operator begins. This message is a string of EBCDIC characters. The first byte designates the total number of bytes in the message (legal range: 0-127).

The message is sent to the operator's console after the system has appended two exclamation points, and the job's system ID to the beginning of the message. These appended characters are used to identify the source of the message. The message is also output to the OUT symbiont file (see M:PRINT).

M:PRINT Writing on the Listing Log

The M:PRINT procedure permits writing a record on the listing log (job control file), normally output to the symbiont line printer.

Syntax

[label(s)] M:PRINT (MESS,[*]adr)

where

MESS is the keyword that introduces the message address.

adr is the word address where the record to be written begins. This record must consist of a string of printable EBCDIC characters. The first byte indicates the total number of bytes in the record (127 maximum; a zero value causes a line space).

The record is printed on the OUT symbiont-printer file associated with the job. Printing occurs either at job end or during its progress. Writing in the OUT output stream can be done either by an M:PRINT procedure or by the standard I/O procedures via a DCB assigned to the OUT device (see Chapter 3).

M:INT Interrupt Initiated by the Operator

The M:INT procedure allows the user's program to receive control (at a specified user routine) from the system upon occurrence of a console interrupt sent by the operator (see XOS/OPS Reference Manual, 90 17 68).

Syntax

[label(s)] M:INT address

where address is the entry point of a user's routine that is entered in response to the operator interrupt.

After the M:INT service has been executed, the system recognizes that the user program is prepared to receive control if the operator presses the INTERRUPT button. When the user routine receives control after an operator interrupt, an abnormal code X'4C' is contained in the first byte of register R5.

The monitor prevents reentrance of the user routine if the operator repeats the interrupt before the user routine is exited.

The interrupt processing is terminated two ways:

1. By the M:ERR procedure, which aborts the job or step depending upon bit 0 of the JSW.

2. By the M:RETURN procedure, which returns control to the system for restoration the program state and registers as they were before the interrupt and to restart the user program at one of two possible addresses. If no address is supplied in the argument field of the M:RETURN procedure, control is returned to the point of interruption. Otherwise, control is returned to the specified address.

TIME AND DATE FACILITIES

The services described in this section enable an executing user program to obtain the absolute time and/or to initialize and test a relative time counter.

M:TIME Obtain Absolute Time

The M:TIME procedure allows an executing user program to obtain the date and time of day, to within one second.

Syntax

[label(s)] M:TIME [*]address

where address is the address of a block of five words where the system stores the date and time as shown below. The date and time are sent to the user program,

in the form of 20 EBCDIC characters, in the specified five words of the procedure argument:

Word 1	h	h	.	m	
Word 2	m	.	s	s	
Word 3	.	c	c	ƀ	
Word 4	d	d	.	m'	
Word 5	m'	.	y	y	

Two decimal characters specify each of the following:

hh	–	hour
mm	–	minute
ss	–	second
cc	–	hundreth of a second
dd	–	day
m'm'	–	month
yy	–	year

also

.	–	period
ƀ	–	blank (X'40')

M:GETDAY Obtain the Date

The M:GETDAY procedure allows the user program to obtain the date (year and day), in a contracted form during program operation.

Syntax

[label(s)] M:GETDAY [*]number

where number is an integer specifying the number of days by which to modify the current date before its return to the user.

The date, modified by the value of the procedure argument, is transferred to the user program in R6 and R7 in the form of a series of eight EBCDIC characters:

R6 register – ƀ y y d

R7 register – d d ƀ ƀ

yy – two decimal characters specifying the year.

ddd – three decimal characters specifying the number of the day in the year.

Note: The contents of register R5 are also modified by this procedure; the result in R5 is unpredictable.

M:STIMER Start Countdown Timer

The M:STIMER service requests that the system initialize a job-unique clock counter for a specified interval, activate it only while the requesting program is running, and branch to a user-specified routine when the interval has elapsed.

Syntax

[label(s)] M:STIMER {(MIN, value) / (SEC, value) / (TUN, value)}, [*] address

where

value is an integer specifying the number of time units in the interval. Depending upon the keyword, the time unit adopted is

MIN – minute

SEC – second

TUN – elementary interval. (One pulse on hardware clock 3; normal for XOS is 500 Hz.)

address is the starting address of the user routine.

The clock counter is started at a specified value. It is regularly decreased only while the user program is executing. When the counter reaches zero, the system simulates a trap (see M:TRAP) and transfers control to the user specified address. The abnormal code value X'4D' is contained in the first byte of register R5.

If several calls to M:STIMER are executed in a program, the most recent call cancels the effect of the previous calls.

An M:STIMER time out (interval elapse) is treated by the system the same as a trap whose handling has been requested by the user via M:TRAP. Hence, as with M:TRAP, the user must exit his interval-elapse "trap" handling routine with either an M:RETURN or M:ERR. Since the considerations are the same, refer to the discussion of M:TRAP for particulars.

M:TIMER Test Countdown Timer

The M:TIMER procedure allows the user program to obtain the time remaining before a clock counter, previously initialized by M:STIMER, reaches zero. Optionally, an active M:STIMER may be canceled.

Syntax

[label(s)] M:TIMER {MIN / SEC / TUN} [, CANCEL]

The following keywords specify the units in which the time remaining in the interval is returned to the user:

MIN – minute.

SEC – second.

TUN – elementary interval (one pulse on hardware clock 3; normal for XOS is 500 Hz).

CANCEL – is an optional keyword. If present, the active M:STIMER is canceled.

The time remaining in the interval is returned to the user in R5 (in the units specified).

9. TELECOMMUNICATION FACILITIES

INTRODUCTION

Fundamental aspects of computer-based data communications systems of the type accommodated by the XOS Telecommunications Management System (TMS) are explained in this chapter. This description explains telecommunications systems for the TMS user; it does not attempt to encompass all types of telecommunications systems. Moverover, concepts and terminology are presented from the programmer's viewpoint and presupposes knowledge of telecommunications.

A telecommunications system consists basically of

- Central computer and associated transmission control equipment.
- Remote stations.
- Electrical circuits (transmission lines or data links) that connect remote stations to the central computer (see Figure 9-1).

The equipment by which the CPU is connected to the transmission lines is called the Transmission Control Unit.

TERMINOLOGY

The terminology defined herein applies to telecommunications system usage within this chapter.

TRANSMISSION LINE

A transmission line is the media of data exchange between two stations. It can be classified according to whether it connects two or more than two stations and whether the connection between the central computer and the station is continuously established.

NONSWITCHED LINE

A nonswitched line continuously links the associated stations with the central site regardless of the amount of time it is used. This type of line (e.g., telephone or telegraph) is usually provided by a common carrier on a contractural basis.

SWITCHED LINE

A switched line provides a connection between the central computer and the remote station as established by a dialing procedure.

TYPES OF OPERATION

Simplex. Permits transmission in only one direction.

Half-duplex. Permits data transmission in two directions but in only one direction at a time.

Full-duplex. Can accommodate data transmission in both directions simultaneously.

LINE CONTROL (TRANSMISSION PROCEDURE)

Line control is effected by a group of control characters framing transmitted data, which allows the control of the progression of a transmission. Examples of the type of control are

- Addressing a terminal.
- Verification of data transmitted.
- Assure restarts in the case of an error.

GROUP OF LINES

A group of lines is a set of transmission lines linked to the same computer and used in one given applicaton. These lines, along with the terminals to which they are linked, must have identical characteristics (essentially, the same transmission characteristics and data transmission rate).

TERMINAL AND COMPONENT

A terminal consists of a control unit and one or more input and one or more output devices; each individual device is called a component of that terminal.

STATION AND NETWORK

The equipment constituting a station can be either a terminal, as defined above, or another computer. Stations

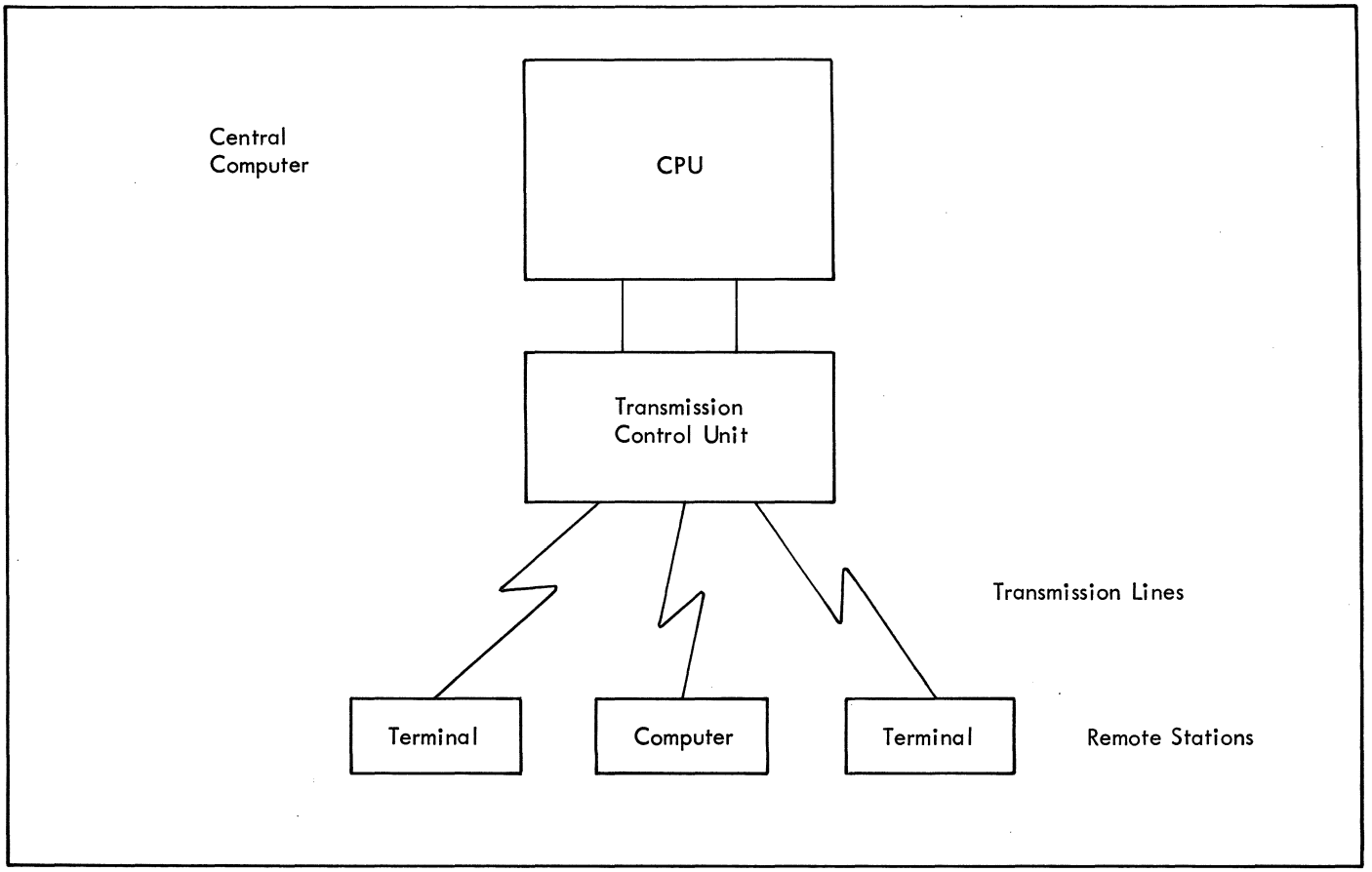


Figure 9-1. Basic Elements of Telecommunications System

are usually separated from the central computer by a distance sufficient to require common carrier facilities and transmission techniques to accomplish communication between the central computer and the remote station. "Station" is synonymous to "terminal."

A network is a group of one or more stations connected by a single transmission line (see Figure 9-2).

TYPES OF STATIONS

Central Station. The network station that has the capability of inviting one of the other stations to send or receive data.

Remote Station. Any of the other stations of the network that responds to the invitation of the central station to send or receive a transmission.

TYPES OF NETWORKS

Bipoint. A line is called bipoint if it connects the central computer (station) to a single remote station.

Multipoint. A line is called multipoint if several remote stations are connected to a single transmission line.

DATA STRUCTURE

Character. Composed of a predefined number of significant binary digits, or bits.

TRANSMISSION CODE

A transmission code is a finite number of characters each having the same number of bits.

Example:

EBCDIC, an 8-bit character code.

ANSCII, a 7-bit character code.

(See Appendixes G and H.)

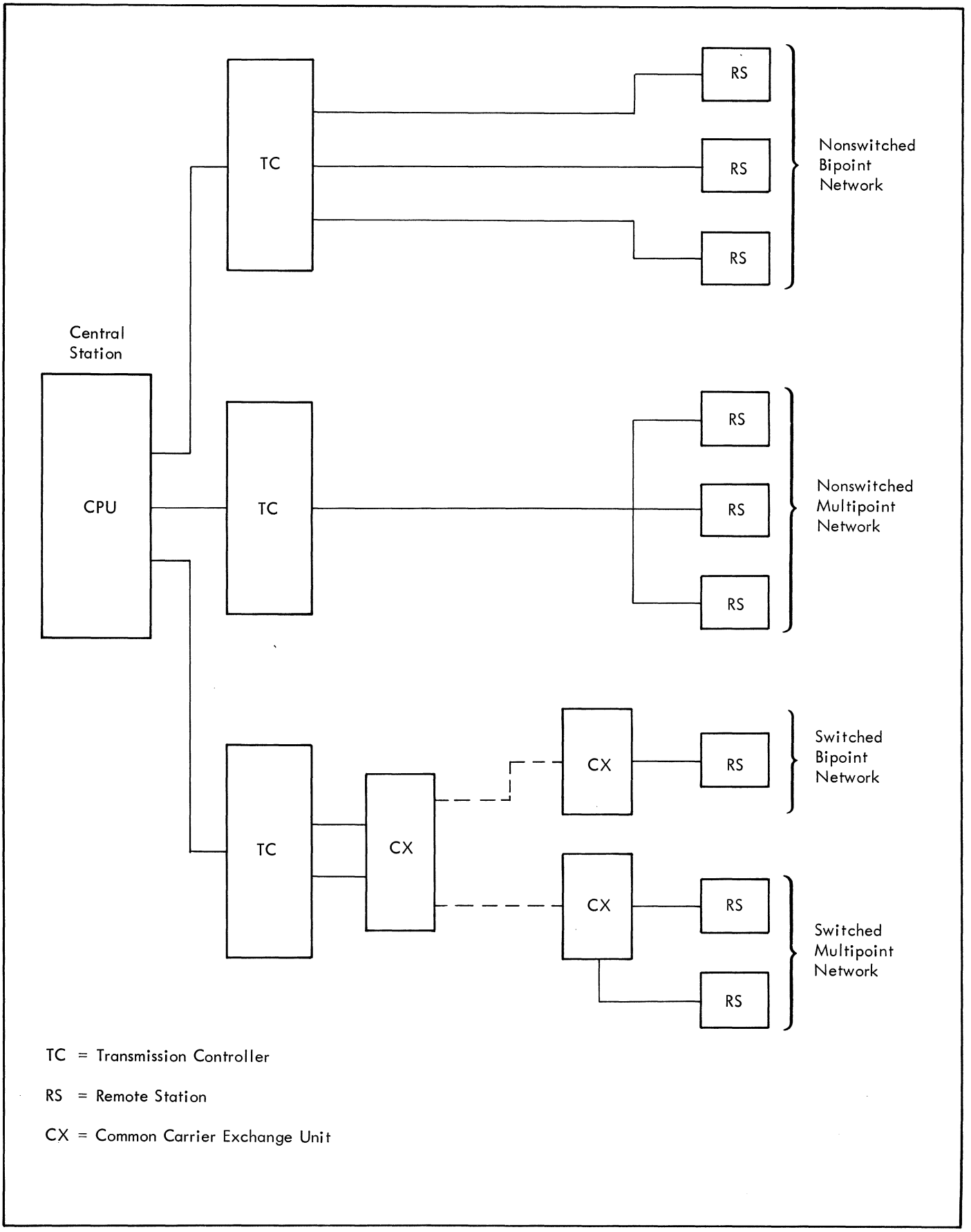


Figure 9-2. Network and Station Example

TRANSMISSION BLOCK

A transmission block is a finite number of characters of the same code transmitted on the line in groups. A block can be of fixed or variable length.

MESSAGE

A message is any sequence of data characters considered as a unit, including any control characters necessary for transmission on a communications line.

RECORD

A record is a collection of related items of data, treated as a unit of information.

PACKING

Packing consists of eliminating non-significant blanks at the end of a record, before the transmission of that record.

TRANSMISSION MODES

Character Mode. Where the data unit transmitted is the character. It is used with terminals that do not have a buffer memory.

Message Mode. Where the data unit transmitted is the block. It is used with terminals having a buffer memory.

TRANSMISSION PROTOCOL

The rules that govern the data exchange on a transmission line are called transmission protocol. In message mode, these procedures append special characters framing the text, permitting transmission of special messages called supervisory sequences. These supervisory sequences also permit control of the data transferred and assure proper transmission.

SUPERVISORY SEQUENCES

Supervisory sequences can be classified in two categories: addressing interrogation sequences and receipt sequences. Supervisory sequences are normally associated with message mode transmissions.

ADDRESSING SEQUENCES

These sequences select or identify the station.

SPECIFIC POLLING SEQUENCES

Polling sequences are executed by the central station and invite a component of a remote station to transmit a message. The particular component being polled (a request to transmit) is identified by its unique address. An example of a typical polling sequence follows:

Format:

0. POL. ADS. ENQ

where

0 is a string of synchronization characters (SYN).

POL is a control character that specifies polling and gives the terminal address.

ADS is a control character that gives the address of the component.

ENQ is a control character that indicates the end of the supervisory sequence.

GENERAL AND CONTINGENT POLLING SEQUENCE

This sequence generally concerns the application of concentrators of teleprinters or of visual display stations. These two polling sequences either request a specific component to transmit or request any component of the remote station to transmit. A general polling sequence is executed by the central station. The central station invites the remote station to transmit all the messages coming from the different components having made a transmit request. An example of a typical general polling sequence follows:

Format:

0. POL. ADG. ENQ

where ADG is a control character that explicitly specifies general polling.

A contigent polling sequence is executed by the central station. The central station invites the remote station to transmit the message of a component determined by the interrogating concentrator.

An example of a typical contigent polling sequence follows:

Format:

0. POL. ADA. ENQ

where ADA is a control character that specifies contigent polling.

NORMAL SELECTION SEQUENCE

This sequence is executed by the central station. It requests a remote station to prepare to receive a message. A typical normal selection sequence follows:

Format:

0. SEL. C2. ENQ

where

SEL is a control character that specifies selection and gives the address of the terminal.

C2 is a control character that gives the address of the component.

FAST SELECTION SEQUENCE

This sequence concerns teleprinter concentrators and visual display applications.

The central station does not send a warning to the remote station but sends the data in the same sequence. A typical example of a fast selection sequence follows:

Format:

0. SEL. data block

RECEIPT SEQUENCES

Receipt sequences are sent by a slave station to inform the master station of the reception of the block sent by the central station and the state of the slave station for the continuation of the transmission. The two most frequently used receipt sequences follow:

Standard format:

X. ACK positive acknowledgement (no errors in transmission)

X. NAK negative acknowledgement (errors in transmission)

where X is a control character capable of taking several configurations (N0B or DLE or SEL).

TELECOMMUNICATIONS ACCESS METHOD

The Telecommunications Management System (TMS) is comprised of a collection of system services, provided under XOS, to manage various telecommunications devices connected through transmission lines. TMS provides the means by which a user can communicate with various remote terminals via a set of system procedures called Telecommunications Access Method (TAM). TAM enables the user to work at a level removed from the actual transmission procedure.

TAM permits the user to work at the block level and exchanges logical data blocks with the user managing the physical blocks transmitted.

TAM corresponds to other XOS access methods in that it performs functions as a result of a user executing various system procedures. Although some of the TAM procedures are similar to those for other access methods, extended forms are provided for full remote-processing capabilities.

Some of the major functions provided by TAM are as follows:

- Line protocol handling.
- Controller buffering/handling.
- I/O and external interrupt processing.
- Error detection, notification, and retry processing.
- Request queuing.
- Line time-out processing.
- Polling/selection.
- Automatic data translation.
- Special character recognition and notification.
- Line initialization.
- Network management.

PROGRAM-LINE RELATIONSHIP

It is necessary before performing any input or output operations on a line or group of lines to establish a connection between the line and the program using it.

The program using a line or group of lines must define its usage of the line by means of the M:DCB procedure which, during assembly time, creates a Data Control Block (DCB).

The final link between the line or group of lines in accomplished by the M:OPEN procedure. It establishes the linkage, by means of the operational label, between the real line defined by the !ASSIGN command and the DCB defined in the program using the line. Once the linkage has been established the program is allowed to issue read and write procedures to perform data transmission operations over a transmission line.

Termination of the program-line connection is accomplished through the M:CLOSE procedure. Close terminates the

availability of a line or group of lines and causes the fields in the DCB to be restored to the condition that existed before the DCB was opened.

RESOURCE ALLOCATION

Allocation of a line or group of lines to a job is accomplished through the !ASSIGN command.

This command has a dual role; it

1. Reserves a group of lines, or a unique line, as resources for the job.

2. Informs the access method of the address of the lines and terminals of the group.

Format of !ASSIGN Control Command for TAM:

```
!ASSIGN op-label[,{MTN}],DEV,(LIN;
                  {FRE}
!      ,{group-id}){,(LLK,op-label)]
        {line-id }
```

where

op-label is a one- to four-alphanumeric-character identifier. It assures the link between the physical resource defined by the !ASSIGN control command and the DCB of the user program (see "Data Control Block").

MTN is a keyword signifying that the matching between the operational label of the DCB and the !ASSIGN control command is valid for the job step in which the !ASSIGN control command appears and for subsequent steps until the end of job or until the appearance of a new !ASSIGN control command using the same operational label.

FRE is the option by default. In this case, the operational label and the corresponding association with the !ASSIGN control command are valid only for the current step.

group-id is a four-alphanumeric-character identifier that specifies the symbolic name of the group of lines defined at system generation.

line-id is a four-alphanumeric-character identifier that specifies the symbolic name of the line defined at system generation.

LLK allows the chaining of the operational label to another operational label defined in another !ASSIGN control command. Therefore, several lines or groups of lines are linked simultaneously to the same DCB.

The !ASSIGN control command used in transmission does not allow the definition or redefinition of DCB parameters as is possible with the other access methods.

The maximum number of resources of the DEV type (lines included) that can be assigned to a job is 12. However, the number of lines belonging to the group of lines defined at system generation is not limited (see XOS/SM Reference Manual, 90 17 66).

M:ASSIGN PROCEDURE

This procedure allows the dynamic association of an operational label of the program DCB to an operational label defined in an !ASSIGN control command.

Syntax

```
[label(s)] M:ASSIGN (OPL,{[*]addr   })
                        {'op-label'}

      ,(UNT,OPL,{[*]addr   })
                {'op-label'}
```

where

OPL introduces the operational label one- to four-alphanumeric-characters) providing the link with the DCB of the user program.

UNT,OPL implies use of the resource (device and volume) defined and allocated to the job step by the specified operational label.

DATA CONTROL BLOCK

The Data Control Block (DCB) groups the fixed parameters defining the line or group of lines along with the specifications on their usage. The DCB is used in the interpretation of I/O requests made by the user.

CREATION OF THE DCB

The DCB is created at assembly time by the M:DCB procedure. It enables the user to introduce, in a write protected area, all or part of the following parameters:

[label(s)] M:DCB (OPL,'op-label')[,(ERR,address)]

[,(ABN,address [,code],...)]

[,(SIM,value)],(AM,{BT / VS})

,(MOD,{ASC / EBC / BIN})

[,(LST,address [,address] [,address]

[...])]

where

OPL introduces the DCB operational label. The operational label is used in the !ASSIGN command for the assignment of a line or group of lines to the user program (a maximum of four alphanumeric characters).

ERR specifies the entry point address in case an error detected upon execution of an M:CHECK procedure. If this parameter is omitted, the job is aborted if an error occurs during a transmission request.

ABN specifies the address of an abnormal exit. The absence of this address causes a program abort if an abnormal condition occurs. The abnormal classes are

X2 End of special transmission.

X3 Processing abnormalities.

The absence of one of these codes causes the job to be aborted if an abnormal condition corresponding to one of these classes is detected.

SIM specifies the maximum number of simultaneous I/O's which can be performed on the line or group of transmission lines linked to the DCB.

AM specifies the access method:

BT for TAM

VS for VSAM

TAM is compatible with the VSAM access method (i.e., a DCB coded for TAM usage can be used with TAM or with VSAM). This allows the debugging of a transmission program: read requests serviced by means of a card reader, write requests serviced by means of a printer.

The user cannot issue any of the special read requests (e.g., read survey), special write options, or procedures pertaining to data transmissions (M:DEVICE, M:MDFLST).

MOD specifies the data code used by the user:

BIN Binary (valid only in message mode).

EBC EBCDIC (default option).

ASC ANSCII.

The data codes allowed are dependent upon the transmission code of the lines assigned to the program as shown in Table 9-1.

Table 9-1. Allowable Data Codes

Line Code \ User Code	ANSCII	EBCDIC	Binary
ANSCII	yes	yes	no
EBCDIC	yes	yes	yes
Binary	no	yes	yes

LST defines the explicit list or lists used. The word address of one to four component lists can appear in this case (see Terminal and Component Lists). The first list coded is activated when the DCB is opened. Only one list is linked to a DCB at a given time.

M:MOVEDCB PROCEDURE

This procedure reserves space required for a DCB in the common dynamic area in read-only protection.

It returns the address of the inactive DCB stored in the common dynamic area. Default values are assigned to the parameters not coded in the DCB.

This procedure can be utilized in

- Creating, at the execution of a program, DCBs for which the need is not possible to anticipate at assembly time.

- Saving the content of inactive DCBs in a program before the loading of another program is called for by M:LINK.

Syntax

[label(s)] M:MOVEDCB [*]addr, (PTR, dcb-addr)

where

addr is address of the sending area. This field is either an inactive DCB or a nonprotected field where a DCB image has been created by the program.

PTR introduces the address of a pointer where the address of the dynamically created DCB will be placed.

DCB MODIFICATION (M:SETDCB)

M:SETDCB allows modification of DCB parameters during program execution. Before opening the DCB, the M:SETDCB allows the modification of the DCB content. Between the opening and the final closing of the DCB the M:SETDCB procedures allows only the modification of ABN and ERR DCB parameters (for M:CLOSE option MTN,RLS, see "Opening and Closing a Line").

Syntax

[label(s)] M:SETDCB [*]adr-1[,(OPL,{[*]adr-2 / 'op-label'})],options

where

adr-1 is the address of the DCB to be modified.

OPL is a keyword introducing the new operational label to be given to the DCB to be modified.

options are the same options and have the same syntax as those described for the procedure M:DCB (see "System Procedures").

TERMINAL AND COMPONENT LISTS

The user performs I/O operations by referring to a terminal or component appearing in a list. The list consists of all components for all terminals of a line or group of lines assigned to the user upon initiation of his job.

There are two types of lists:

- An implicit list is constructed by the system when a line or group of lines is opened.
- An explicit list is created by the user via the M:LIST procedure.

An implicit list allows flexible line usage in that the user can process different lines by changing the assignment of a line or group of lines.

An explicit list allows the user to specify a subset of components of the group or the order in which they appear in the list.

In either case, once the line or group of lines is opened, the implicit or explicit lists are processed identically.

OPENING AND CLOSING A LINE

The opening of a line or a group of lines is accomplished through the M:OPEN procedure which must be executed before any transmission procedure reference and before any M:MDFLST procedure reference. The M:OPEN procedure allows a user to establish a temporary connection between the program and the line, or the group of lines.

Functions provided are

- Verification of the compatibility between the explicit lists described by the user in his program and the group of lines assigned as resources.
- Initialization of the lines: initiation of the transmission device controller and the line adapters.
- Verification of the connection of the lines and the operational status of the intermediate telecommunications equipment.
- Creation in the user area of the list of components or terminals if it is for an implicit list (see Terminal and Component Lists).
- Creation of the Data Extension Block (DEB) in which TAM places the line specific parameters of the line or group of lines linked to the DCB.
- Reservation and setting up of the communication tables (IOB) between the access method and the I/O supervisor; allocation of the default values for all the parameters remaining undefined in the DCB; creation of the tables containing the variable parameters of the transaction in process.

- Validity checking of the DCB parameters, their coherence with the processing mode of the transmission line.

When specific transmission lines of the group linked to the DCB cannot be initialized correctly, the opening of the group is performed normally. The lines that are not initialized are brought to the attention of the user, when the first I/O on these lines is performed, by an abnormal code during the execution of the M:CHECK procedure. The user can attempt to open the line not initialized by using the M:DEVICE procedure (option OPN – see Input/Output Procedures).

Syntax

[label(s)] M:OPEN [*]dcb-address, S

where S indicates the processing mode: Scratch; it is the only mode recognized for opening a line or group of lines.

The M:CLOSE procedure terminates the program line and causes the closing of the DCB (i.e., breaks the linkage). Functions performed are

- Wait and test for completion of the last I/O operations.
- Temporary or final DCB locking.
- Releasing the memory space reserved for the tables constructed when the DCB was opened.
- Disconnecting the lines and releasing the transmission device controllers and line adapters.

Syntax

[label(s)] M:CLOSE [*]dcb-address

$$,\left[\left(\left\{\begin{matrix}\text{HLD}\\ \text{RLS}\\ \text{MTN}\end{matrix}\right\}\right)\right]$$

where

HLD specifies temporary close which allows the user to wait on all the I/O events currently active on the DCB.

RLS causes the release of the DCB-group connection of lines (or line) and job-group of lines. The resources constituted by the lines of the group are released when the option FRE appears in the !ASSIGN command (see "Resource Allocation").

MTN indicates the suppression of the connection DCB-line group but saves the link between job-group of lines allowing the DCB to be used to process this same group of lines within the same job. The lines are not released, even if the option FRE appears in the !ASSIGN command.

BUFFER MANAGEMENT

BUFFER AREAS

All I/O operations performed on transmission blocks are sent or received from buffers provided by the user.

The handling of these buffers is a user responsibility. The user must reserve their location and specify their address at the I/O procedure reference level.

ACCESS METHOD

TAM (Telecommunications Access Method) is a basic access method that performs I/O operations on remote devices. This access method is composed of a group of services to which the user refers by the following procedures:

- M:READ, M:WRITE to execute transmit or receive operations to a terminal.
- M:DEVICE to specify a mode change or perform a specific operation on a terminal.
- M:LIST, M:MDFLST to dynamically construct or modify the terminal lists.
- M:CHECK to test the progression of the I/O while waiting for its completion.
- M:WAIT to wait for the end of one or several I/O events.

An event control block (ECB) may be associated with each of these procedures to enable the user to perform other asynchronous processing while issuing several READs or WRITEs (up to a maximum of the SIM parameter specified in the DCB). When the user has performed an M:CHECK procedure on this event, the user is set in a wait state to regain control when the block has been transmitted. The contents of register 5, upon return, contains a code that indicates the manner in which the transmission was performed (specifically, if an error occurred).

Two data transmission modes exist: message mode and character mode.

- Message mode for handling terminals having a buffer memory.

- Character mode for handling terminals when the data unit transmitted is the character.

JOB PRIORITY ASSIGNMENT

The telecommunications jobs are generally characterized by a high content of input and output operations.

It is desirable to assign them to a high interrupt priority level in relation to the other classes of jobs. To avoid a job controlling the CPU time at the expense of other jobs, there is a systematic rotation between these jobs based on the regular progression of an interval timer. If the user wishes to use this procedure, the time quantum allocated is fixed at the system generation.

DEBUGGING

The dubugging of a telecommunications program is a delicate operation. Such a program must be executed in a serial job class and should take advantage of the XOS debugging aids (DEBUG processor). Thus, various telecommunications processing points of the program can be correctly covered; the terminal lists and the corresponding index values can be simulated.

The interface requires usage of the following two steps:

STEP ONE

Hardware simulation by replacing the lines and terminals with local devices accepting the same data formats. This is made possible by a local assignment in an !ASSIGN command by means of the similarity between TAM procedures and conventional I/O procedures.

There can be no special operations as in DEVICE and MDFLST procedures.

STEP TWO

Debugging of the program connected to the device controller and to the real terminals first locally through lines of reduced length then remotely through real lines.

ERROR AND ABNORMAL PROCESSING

The normal execution of an I/O operation, and opening or closing of a transmission file can be interrupted for the following reasons:

- Programming error (e.g., transfer length greater than the maximum length permitted for the component declared at system generation).
- Initiation error of a job, invalid DCB.
- Abnormal conditions arising during the processing of a file (e.g., initial polling or selection rejected).
- Errors during file processing (e.g., an invalid response from a terminal.

The occurrence of an abnormal or error condition during the processing of a transmission file causes the execution of a user exit specified by the address in the DCB under ABN and ERR parameters.

The abnormal conditions and errors lead to an abort of the user program if no user exit address has been provided to process the condition either by omission of the ABN or ERR address, or nonselection of the corresponding abnormality class (ABN parameter in the DCB – see "Data Control Block").

Upon entry the following registers are loaded by TAM:

Register 5	Error or abnormal code, left justified; return address right justified.
Register 6	DCB address, right justified.
Register 7	Event Control Block address (ECB) associated with the M:CHECK procedure.

The user may initiate other I/O procedures within the exit routine, with the exception of the M:CHECK procedure. The user must exit via the M:RETURN procedure.

STATISTICS AND ACCOUNTING

TAM records all the errors and their characteristics in the job accounting file. The error characterisitcs saved are: line name, terminal name, type of operation performed, number of retries, and status of the line.

Accounting information produced by TAM includes the number of blocks exchanged during transmission, the elapsed time between opening and closing of the lines, the number of calls to the I/O supervisor, the number of

characters transferred, and the number of on-line transmission errors.

MESSAGE MODE

Successful communication with a message mode remote station requires that the data stream between the central station and the remote station contain the appropriate line control characters and character sequences (see Figure 9-3). TAM assumes responsibility for processing the control characters which are appended to the data block being transmitted on the line. If the block is a multirecord block, it is the user's responsibility to provide record header characters and record separators. It is also the user's responsibility to provide page formatting or line number characters contained in each record.

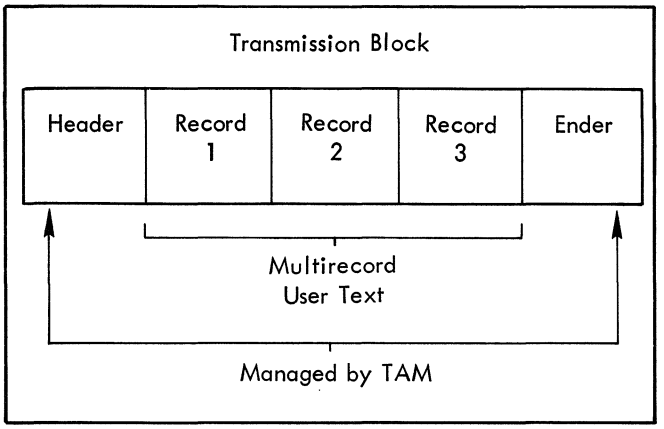


Figure 9-3. Control Characters Appended by TAM

TAM provides the facilities, called error recovery procedures, that diagnose a variety of error conditions that occur during message transmission, and attempt to recover those conditions that are considered recoverable so that the transmission can continue.

TAM provides the code translation between the transmission code employed by the remote station and that which the user has specified as a working code.

COMPONENT LIST

Message mode utilizes a component list. The user can process two types of component lists:

- Implicit list constructed at the opening of the line by the system (M:OPEN procedure).
- Explicit list defined by the means of the M:LIST procedure at assembly time. The user specifies in this DCB by the keyword parameter LST the address of the list.

POLLING/SELECTION LIST

The transmission procedure in message mode uses specialized characters called supervisory sequences to control and chain the transfer of data. These sequences are referred to as

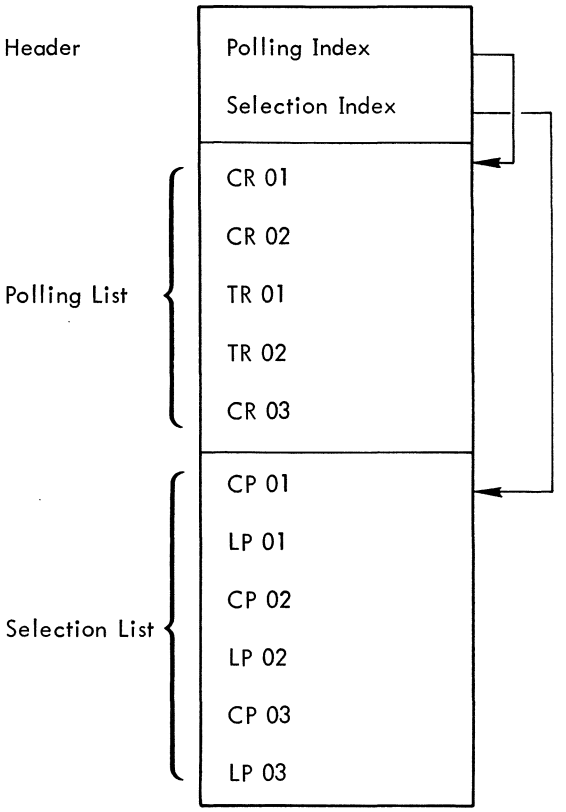
- Polling sequence (an invitation to the terminal to transmit data).
- Selection sequence (an invitation to the terminal to receive data).

List definitions:

- A polling list is one in which the components appear for the different stations capable of transmitting data to the central station.

A list appears as a table that consists of as many entries as there are components in the list. The list consists of a header containing an index that points to the operational entry at the time considered. The indexes include

- Polling index in the case of a polling list.
- Selection index in the case of a selection list.



Depending on the type of transmission line (multipoint or bipoint) there are two types of component lists:

- Sequential access list. In the case of a multipoint line, a list in which the user can access the various

components of the list sequentially. The user cannot perform simultaneous I/O on several components of the list.

- Mixed list. A list in which the user can access directly certain components located on different lines and sequentially other components located on the same line.

  In this type of list, the user cannot use automatic polling (see M:READ procedure, SUR option).

M:LIST PROCEDURE

The declaration of an explicit list is accomplished by means of the nonexecutable M:LIST procedure. This procedure reserves, in a read-only area, the space necessary for the list. The list is partially constructed at assembly time and completed at execution time when the DCB is opened.

Syntax

[label(s)] M:LIST (IN,{OPN / WRP},name[,name][...])

,(OUT,name[,name][...])

where

IN describes the polling list. Using automatic polling, the user can specify if the list is OPN or WRP.

OPN (linear) TAM scans the component list of the different stations of a multipoint line only once, even if a station has answered. When the polling index reaches the end of the list, any subsequent I/O request causes an abnormal return unless the user has forced the index to another entry in the list. When a selected terminal is not ready to receive, the selection index list is not changed, and no return is made to the next terminal list. The user is warned by an abnormal exit.

WRP (looped) TAM interrogates the various components of the list until one of the stations interrogated answers. The list index is automatically repositioned at the beginning of the list when the end of the list is reached.

Note: Automatic polling can only be used with a sequential list, all the components of which must belong to the multipoint line.

name corresponds to the symbolic name of the component defined at system generation. It can be cited explicitly by a string of four characters between brackets or implicitly by the user program word address where a string of four EBCDIC characters is located.

This last method enables the user to keep the component name and to be able to reference it symbolically.

OUT describes the selection list.

An explicit list can consist of as many as 254 polling entries and 255 selection entries. (See Appendix F for explicit format of a list after assembly.)

M:MDFLST PROCEDURE

Implicit or explicit lists are reserved in read-only memory. The M:MDFLST procedure enables the user to modify these lists during the execution of his program.

Syntax

[label(s)] M:MDFLST [*]DCB-address

{ ,({SKP / ACT / IND}[,[*]value]),({POL / SEL / PS}[,(LST,[*]word-address)])

,(RSP)

,(NEW),(LST,[*]word-address) }

where

LST is the address of the list. When this option is not specified, the active list of the DCB is used. When the option NEW is present, the list specified becomes the active list attached to the DCB. The address of this new list must have been previously specified in the DCB (LST option). The new option is accepted, between OPEN and CLOSE only when no I/O is in progress or waiting. Otherwise, the user program is aborted. Therefore, it is preferable for the user to execute a CLOSE HLD before trying to activate a new list.

SKP – Suppress
ACT – Restore } a specified component either (1) by the index value of his entry in the polling list (POL) or selection list (SEL), or (2) by a user program word address where the index value of the entry is located. In this case, the value must be left justified in the word. The suppression (or restoration) of a component in a

list can be done either in the polling list, in the selection list, or in the polling and selection list (PS).

IND the index of a selection list (SEL) or of a polling list (POL) or of polling and selection list (PS) is to be changed to a value specified directly or by a word address of the index value. In this last case, the index value must be left justified in the word.

This operation is refused when there is automatic polling in progress. In this case, the condition code is set to 01 (CC3 CC4 = 01).

PS when the list comprises a polling list and a selection list, it is possible to designate, at the same time, a component in the polling list and the same component (or its equivalent) in the selection list, on the condition that there is correspondence of the polling and selection index.

The value of the index must not be greater than 255. If the value indicated is erroneous, the operation requested is not performed and the condition code is set to 01. (CC3 CC4 = 01). An active component, one that has I/O outstanding, cannot be suppressed. In this case, the operation requested is not performed and the condition code set to 01 (CC3 CC4 = 01).

RSP stop automatic polling operation on a looped (WRP) list. In this case, the list concerned is the active list. When the option LST is coded, it is ignored.

When an automatic polling operation is in progress and all the components of the list answer negatively, the operation is stopped at the end of the list. Upon the execution of the corresponding M:CHECK procedure, a negative polling abnormality code and the index of the last station interrogated is returned to the user. The index value is only returned if IND option is specified in the M:CHECK procedure.

INPUT/OUTPUT PROCEDURES

The I/O operations on terminals are performed by means of M:READ and M:WRITE procedures. The user issues one of these procedures each time he wishes to receive a message from a remote station or send a message to a remote station. The read and write procedures have special options that are dependent on whether the line is bipoint or multipoint.

M:READ PROCEDURE

Syntax

[label(s)] M:READ [*]dcb-address,(BUF,[*]byte-adr)

[,(TRL,[*]value)][,(PTR,[*]word-adr)]

[,(SUR)][,IND,[*]value)]

where

dcb-address is the address of the DCB that specifies the line on which the read operation is to be performed.

BUF defines the byte address of the memory area where the transfer is to be performed.

TRL defines the maximum number of bytes to transfer. This transfer length must be equal to or exceed maximum transfer length defined for the component at system generation. When this option is absent, the default value assigned is the maximum component transfer length fixed at system generation.

PTR defines the address of the event pointer (ECB) associated with the I/O at the execution of the procedure. This option is necessary if the user wants to perform an I/O wait M:WAIT procedure or a M:CHECK on multiple I/O requests.

SUR requests an automatic polling operation on a multipoint line. The user requests receipt of a message from the component corresponding to the index specified by IND or if the index is omitted on the active component in the polling list.

In a loop list when a terminal has no message to send, it answers negatively, and a new polling operation is automatically initiated to the component corresponding to the next entry of the list, etc., until one of the components interrogated answers positively or, for linear lists, when end of the list is encountered.

For a loop list, the polling index is reinitialized, at the end of list and the operation resumes. Polling ceases in this case only upon the acquisition of a message or at the end of the list after the execution of an M:MDFLST procedure with the RSP option.

The detection of a nonrecoverable error stops the automatic polling operation. If the user is in a

wait state for the I/O event after issuing an M:CHECK, he regains control at the address specified by the ERR parameter in the DCB.

IND forces the request to the component identified by the index value of the active list associated with the DCB. When this option is absent, the entry is the one corresponding to the value of the polling index contained in the list. The value of the polling index of the list is not disturbed by the value of the IND option of the procedure reference. The IND value range is from 1 to 254. When it is specified indirectly in a word of the user program, it must be placed in the left-most byte of the word.

Note: The polling list index is the users responsibility. The list index can be modified by the execution of an M:MDFLST procedure (IND) or M:READ(SUR). In the last case, the list index points the component which has answered positively to the automatic polling. The user can specify the component on which he wants to execute an M:READ procedure, but in no case will the value specified by IND affect the value of the list index.

M:WRITE PROCEDURE

Syntax

[label(s)] M:WRITE [*]dcb-address,(BUF,[*]byte-adr)

,(TRL,[*]value)[,(PTR,[*]word-adr)]

[,(IND,[*] value)][,(FIN)][,REP)]

where

BUF, PTR have the same meaning as for the M:READ procedure.

TRL defines the effective number of characters to transfer. This value cannot be zero or exceed the maximum transfer length defined for the component at system generation.

IND defines the component in the active list to which the message is to be transmitted. When this option is absent, the entry is the one corresponding to the value of the selection index contained in the list. The value of this index is not affected by the value of the IND option, The IND value range is 1 to 256. When it is specified indirectly in a word of the user program, it must be placed in the left byte of the word.

REP allows the user to reinitiate a WRITE request that could not be executed because of a transmission problem. The user buffer that has already been translated during the first request, in the case of an on-line transmission code different from the user code, is transmitted without a new code translation.

FIN indicates last block of message. Not used for terminals that accept only one block per message or do not uniquely identify the last block.

M:CHECK PROCEDURE

The M:CHECK procedure awaits the end of an I/O event if it has not occurred, and controls the progress of a transmission request. When a transmission error occurs, control is passed to the user at the address specified in the DCB (option ERR).

When this option is not present, the program is aborted. In the case where an abnormality occurred during processing and it corresponds to an abnormality class anticipated by the user in his DCB (option ABN), control is passed to the address specified, otherwise the program is aborted.

Syntax

[label(s)] M:CHECK [*]dcb-address[,(PTR,[*]address)]

[,(RSA,address)][,(IND,address)]

where

PTR,address indicates the event pointer address specified in the user's M:READ or M:WRITE procedure. If this option is present in READ or WRITE procedure reference, it must be present in the corresponding M:CHECK; if absent in a READ or WRITE procedure reference, it must be absent in M:CHECK for all M:CHECK procedure references applicable to the DCB.

RSA,address indicates the address where the effective number of bytes transferred in a read request is placed.

IND,address indicates an address where the user wants to recover the index in the polling or selection list, corresponding to the event.

M:DEVICE PROCEDURE

This procedure enables the user to specify a transmission mode change or to perform a specific operation on a component.

Syntax

[label(s)] M:DEVICE [*]dcb-addr[,({BEL / SUS / ABO})]

,(IND,{POL / SEL},[[*]value])[,(MOD,{BIN / EBC})]

where

BEL allows the user to send an alarm to a component. For some terminals, this sequence blocks the terminal and requires intervention by the local operator.

SUS suspends transmission with a component. After execution of a READ procedure, the suspension implies that the last transmission block has been currently received (i.e., execution of the M:DEVICE SUS acknowledges the last block sent by the terminal). The suspension resets the line to the inactive state.

ABO ceases transmission with a component. After execution of a READ procedure, the execution of this procedure with the ABO parameter requests the component to save the block previously sent for the next transmission (next READ on this component).

IND defines the entry of the polling (POL) or selection (SEL) list, specifying the component on which the operation must occur. When the parameter value is absent, the value of the index is that of the active list.

The value of IND must be greater than zero and less than 256. When it is placed in a word of the user program it must be left-justified in the word.

MOD enables the user to redefine the code of his data dynamically in the case where the code specified by the option MOD of the DCB is EBCDIC or binary. Some terminals have the capability of transmitting blocks in EBCDIC or binary depending on the type of data read on punched cards. Therefore, the user can be warned as soon as the block is received, that the next block is a binary block. By changing his working mode, the user does not receive the abnormal code at the return of the M:CHECK procedure after the reception of the binary block.

Note: Operations BEL, SUS, and ABO requested by the execution of M:DEVICE are systematically executed on the active component of the multipoint line, even if the value of the index specified by IND corresponds to a different component.

In the case of a bipoint group of lines, IND specifies the active component on which the operation occurs.

If the I/O request cannot be performed, or the index specified is erroneous, the condition code is set to 1 (CC3, CC4 = 01).

MULTIPOINT LINE MANAGEMENT

Before a transmit or receive operation can be performed on an idle line, the line must be initialized by sending a polling or selection sequence. TAM is responsible for this initialization. The user does not have to specify the first read or write request. However, the first read or write must be followed by an M:CHECK procedure which tests the progression of the M:READ or M:WRITE procedure and the acceptance of the polling or the selection sequence. When the user does not adhere to this rule, he risks an abnormal exit to the address indicated in his DCB (ABN parameter), upon the execution of the M:CHECK test corresponding to the I/O procedure.

Successive M:READs. If the result of the test for the M:CHECK procedure is correct, the user can initiate a number of successive reads, equal to the value of the SIM parameter coded in his DCB, on the same component of the polling list. The user must initiate reads, on the same entry of the list until the reception of the End of Message code or an error code upon the execution of the corresponding M:CHECK procedure. The user cannot initiate an I/O (read or write) on a different component as long as the line has not been set to idle by the access method.

Therefore, the line is set to idle upon the reception of the End of Message code or an error occurrence.

The user can also suspend or release the transmission by executing the M:DEVICE procedure. The line is then set to idle.

Successive M:WRITEs. If the result of the test for the M:CHECK procedure is correct (i.e., the message corresponding to this first write has been accepted), the user can initiate a number of successive writes equal to the value of the SIM parameter coded in his DCB on the same component of the selection list. He must specify if it is the last transmission block (option FIN of the M:WRITE procedure) for the access method indicates end of transmission and resets the line to idle. The user can then initiate an I/O (read or write) on a different component.

BIPOINT LINE MANAGEMENT

As for a multipoint line, the first procedure reference (M:READ or M:WRITE) which initializes the transmission on a bipoint line must be followed by an M:CHECK procedure testing the progression of the I/O.

However, as soon as all the lines of the group are initialized, the user can initiate a successive number of reads (or writes) equal to the value of the SIM parameter, on the different components of the list (SIM can be equal to the number of lines related to the DCB). On a given line, the user cannot change the direction of the transmission as long as the line has not been set to idle following the reception of end of message on a read (M:READ) or at the execution of the M:WRITE procedure with the FIN option, or at the execution of the M:DEVICE procedure (option SUS,ABO).

ABNORMAL AND ERROR CONDITIONS

FILE PROCESSING ABNORMALITIES

End of transmission abnormalities, not anticipated, are subject to being processed by a user routine for which the ABN address is related to the code X2 upon its introduction in the DCB. User handling abnormalities are processed by a user routine for which the ABN address is related to the code X3 upon its introduction in the DCB (see Table 9-2).

FILE PROCESSING ERRORS

These errors assume a bad operation on the lines or the associated devices. They can be processed by a user routine, the address for which is introduced in the DCB following the keyword ERR. The exit to the user routine occurs only when the number of retries is reached, and the error persists. The I/O error code enables the user to decide whether to attempt a retry, warn the operator, or exit from his program (see Table 9-3).

The transmission errors are classed in two types

1. Recoverable errors for which the error code is less than X'2A'.

2. Irrecoverable errors for which the error code is greater than or equal to X'2A'.

When an irrecoverable error occurs, it is indicated to the user during the I/O test by an M:CHECK procedure, which exits to the user routine that is specified by the ERR parameter of the DCB. If the user initiates a new I/O operation on the same line, a new I/O is attempted on the line only when there is at least one line of the group linked to the DCB, which is not in an irrecoverable error state. Otherwise, the user program is aborted.

Example:

The user has requested the execution of three M:READ procedures simultaneously,

```
M:READ. . . ,(PTR, ad1)

M:READ. . . ,(PTR, ad2)

M:READ. . . ,(PTR, ad3)

M:CHECK     ,(PTR, ad1)
.
.
.
M:CHECK     ,(PTR, ad2)
.
.
.
M:CHECK     ,(PTR, ad3)
```

The execution of M:READ. . . ,(PTR, ad2) ends on an error or an end transmission. At the execution of M:CHECK PTR, ad2), the user program regains control at the address associted with the parameter ERR or ABN. M:READ (PTR, ad3) cannot be executed; the I/O request is purged from the wait queue and the code X'22' is passed to the user upon the execution of M:CHECK (PTR, ad3).

PROGRAM ABORT ERRORS

All the error and abnormal codes described are also possible abort codes when no special processing has been anticipated in the user program to process the conditions to which they correspond. The specific abort codes (see Tables 9-4 and 9-5) can also accompany abnormal end of jobs. All abort codes lower than the value X'80' are related to TMS procedures and operations (when TAM is being used).

An abort always causes the emission of a message on the job control file in the following format:

ABORT	literal	symbolic code	program status at time abort condition was detected

where literal indicates the operational label of the DCB processed at the time the abort condition is detected.

Table 9-2. Abnormality Codes

Hexadecimal Code	Description	Origin	Class
X'01'	End of transmission of message.	M:READ	X2
X'03'	Suspension at the request of the terminal.	M:READ or M:WRITE	X2
X'04'	Initial polling or selection refused.	M:READ or M:WRITE	X2
X'06'	Binary block received but not anticipated.	M:READ	X2
X'1A'	Initial read or write in progress.	M:READ or M:WRITE	X3
X'1B'	Line busy on another component.	M:READ or M:WRITE	X3
X'1C'	Line does not authorize the use of this procedure reference.	M:READ or M:WRITE	X3
X'1D'	Component deactivated in the list.	M:READ or M:WRITE	X3
X'1E'	Polling, selection index greater than the number of components.	M:READ	X3

Table 9-3. Error Codes

Hexadecimal Code	Description	Origin
X'22'	I/O request purged (see "File Processing Errors")	M:READ or M:WRITE
X'23'	Line in permanent error.	M:READ or M:WRITE
X'24'	No response from the terminal.	M:READ or M:WRITE
X'25'	Incorrect block or acknowledgement.	M:READ or M:WRITE
X'26'	Invalid block or receive acknowledgement.	M:READ or M:WRITE
X'27'	Synchronization error.	M:READ or M:WRITE
X'28'	Transmission error.	M:READ or M:WRITE
X'29'	Modem carrier default.	M:READ or M:WRITE
X'2A'	Line locked by the operator.	M:READ or M:WRITE
X'2B'	Modem not ready.	M:READ or M:WRITE

Table 9-4. DCB Open and Close Aborts

Literal	Symbolic Code	Internal Code	Description
'OPL'	S30	30E2	The lines of the group belong to different modes (character and message).
'OPL'	S3A	3AE2	Erroneous explicit list address.
'OPL'	S3B	3BE2	Total number of components related to the DCB greater than 254 in message mode.
'OPL'	S3C	3CE2	Nonconforming usage of the component cited in the list (in message mode).
'OPL'	S3D	3DE2	A component cited in the list does not belong to an assigned list.
'OPL'	S3E	3EE2	A component was cited twice in a direct list.
'OPL'	S3F	3FE2	No line could be activated (character mode).
'OPL'	S41	41E2	Unauthorized access method.
'OPL'	S43	4302	Mode prohibited (binary or ANSCII).
'OPL'	S50	50E2	One of the lines is already in use by another DCB.
'OPL'	S51	51E2	Operational label not assigned.
'OPL'	S55	55E2	One of the operational labels is already in use by another DCB.
(†)	S5E	5EE2	Invalid DCB address.
'OPL'	S6A	6AE2	Too much memory space requested.

†In this case of abort, the 'OPL' operational label, which cannot be found (since the DCB is not valid), is replaced by the following literals:

OPCL when abort has OPEN/CLOSE procedure for origin.

RDWR when abort has READ/WRITE procedure for origin.

DEVC when abort has DEVICE procedure for origin.

RMDF when abort has MDFLST procedure for origin.

CHCK when abort has CHECK procedure for origin.

Table 9-5. Procedure Aborts

Literal	Symbolic Code	Internal Code	Description	Origin
'OPL'	R51	51D9	Zero transfer length.	M:WRITE
'OPL'	R54	54D9	Transfer length greater than (in case of write) or less than (in case of read) the maximum length of the component.	M:READ or M:WRITE
'OPL'	R59	59D9	Erroneous address of the buffer or the event pointer, or of the byte count or index.	M:READ, M:WRITE, M:CHECK, M:DEVICE or M:MDFLST.
'OPL'	R5A	5AD9	No I/O in progress.	M:CHECK
'OPL'	R5B	5BD9	Too many I/O requests in progress (number of current I/O > SIM).	M:READ, M:WRITE, or M:DEVICE.
'OPL'	R5C	5CD9	DCB closed.	M:READ or M:WRITE.
(t)	R5E	5ED9	DCB invalid (e.g., erroneous address).	M:DEVICE, M:CHECK, or M:MDFLST.
'OPL'	R63	63D9	Operation prohibited.	M:MDFLST or M:DEVICE.
'OPL'	R64	64D9	Unknown list in the DCB.	M:MDFLST.
'OPL'	R67	67D9	BIN/EBC mode request prohibited.	M:DEVICE.
'OPL'	R68	68D9	All the lines of the group are in irrecoverable error.	M:READ or M:WRITE
'OPL'	R69	69D9	Nonidentical use of the pointers.	M:CHECK.

[t] In this case of abort, the 'OPL' operational label, which cannot be found (since the DCB is not valid), is replaced by the following literals:

OPCL when abort has OPEN/CLOSE procedure for origin.

RDWR when abort has READ/WRITE procedure for origin.

DEVC when abort has DEVICE procedure for origin.

RMDF when abort has MDFLST procedure for origin.

CHCK when abort has CHECK procedure for origin.

HARDWARE ERRORS

These messages are labeled as follows:

!!IOS ERR xy 'line-id' ?

where

xy is the error number.

line-id is the symbolic name of the lines.

? appears only for messages with answers.

Messages with answers allow the operator to restart the I/O if desired.

Answer R The operator requests restart of I/O from the system.

Answer V The operator, judging the device nonoperational, locks it, thus prohibiting any new I/O.

Errors of this type that could appear in message-mode transmission are shown in Table 9-6.

TRANSMISSION ERRORS

These messages are labeled in the following manner:

!! TEL IOS ERR xy 'line-id' 'terminal-id'

where

xy is the error number.

line-id is the symbolic name of the transmission line.

terminal-id is the symbolic name of the terminal.

Table 9-6. Message Mode Errors

Code	Meaning	Op. Response
ERR02	I/O address not recognized during a SIO, TIO, HIO.	R or V
ERR03	Response from the device controller not anticipated during a SIO, TIO, HIO.	V

Table 9-6. Message Mode Errors (cont.)

Code	Meaning	Op. Response
ERR05	Transmission adapter busy.	R or V
ERR06	Nonoperational adapter or modem default.	R or V
ERR08	Extraneous interrupt.	None
ERR09	AIO address supplied by the TDV not recognized.	None

These messages require no response. They are sent on the operator's console after the standard number of retries have been executed and the error is found to be irrecoverable. The errors of this type are shown in Table 9-7.

Table 9-7. Transmission Mode Errors

Code	Meaning	Op. Response
ERR30	Timing error in memory transfers.	None
ERR31	Transfer error between memory and device controller.	None
ERR32	Carrier default (Modem).	None
ERR33	Modem not ready.	None
ERR40	I/O outstanding without a response from the terminal.	None
ERR41	Incorrect response from the terminal.	None
ERR42	Invalid response from the terminal.	None

The operator always has the ability to lock the line on which frequent errors appear by a UNIT command.

CHARACTER MODE

TAM allows the user to perform I/O operations on terminals without a buffer; data transmission is done at the character level.

TAM offers, for character mode, the same transmission procedures as in message mode: M:DCB, M:LIST, M:OPEN, M:CLOSE, M:SETDCB, M:MOVEDCB, M:MDFLST, M:READ M:WRITE, M:DEVICE, M:CHECK, and M:WAIT. However,

some options of the procedure are specific to the character mode. This section describes the procedure in question and given various handling characteristics of character mode operations.

TAM provides the user with the following capabilities in character mode:

1. Read Capabilities:

   Verification of character parity.

   Correction of the record upon detection of function characters reserved for reading.

   Code translation of the characters received.

   Characters that are filtered by the terminal.

   Detection of end of record characters.

   Telecommand for the paper tape reader.

2. Write Capabilities:

   Code translation of characters transmitted.

   Format control for keyboard records.

   Telecommand for the paper tape reader.

   Format control for paper tape records.

The format capability for write requests are performed only at the request of the user. This allows him to work with records of several print lines with form control being his responsibility.

CHARACTER MODE COMPONENT LISTS

As in message mode, the user can operate with two types of component lists:

1. An implicit list constructed by the system when a line or group of lines is opened (M:OPEN procedure).

2. An explicit list defined by means of the procedure M:LIST in the user program. The user specifies in his DCB, by the list option, the address of that list.

CHARACTER MODE LIST

TAM character mode allows the processing of a group of bipoint lines connected to teletypewriters.

The user can communicate simultaneously with several components of the list. The term assigned this type of list is a direct access list.

Syntax

[label(s)] M:LIST (DIR,name[,name][...]

where name represents a transmission line name in character mode. It can appear as a string of four characters enclosed in apostrophes or a user-program word address where a string of four characters in EBCDIC is located.

This last method enables the user to refer to a component name symbolically. (See Appendix F for explicit list format after assembly.)

M:MDFLST PROCEDURE

As in the message mode, the M:MDFLST procedure allows the user to modify the active list during the execution of his program.

Syntax

[label(s)] M:MDFLST [*]dcb-address

$$\left\{\begin{array}{l} ,\left(\left\{\begin{array}{l}\text{SKP}\\ \text{ACT}\\ \text{IND}\end{array}\right\}[,[*]\text{value}]\right)[,(\text{LST},[*]\text{word-address})] \\ \\ ,(\text{RSS},[[*]\text{value}]) \\ \\ ,(\text{NEW}),(\text{LST},[*]\text{word-address}) \end{array}\right.$$

where

LST is the list address. When this option is absent, the DCB active list is used. When the option NEW is present, the specified list becomes the active list linked to the DCB.

This option is accepted between OPEN and CLOSE only when I/O is not in progress or waiting. Otherwise, the user program is aborted. Thus, it is preferable for the user to perform an M:CLOSE HLD before executing this procedure.

SKP suppresses a terminal designated by the index value of its entry in the direct access list.

ACT restores a terminal designated by the index value of its entry in the direct access list.

IND forces the index of the direct access list to a given value.

RSS stops a supervision operation (see M:READ procedure, SUR option) on the terminal designated by the value of the index, which must concern the active list.

INPUT/OUTPUT PROCEDURES

M:READ PROCEDURE

Syntax

[label(s)] M:READ [*]dcb-address

,(BUF,[*]byte-address)[,(TRL,[*]value)]

[,(PTR,[*]word-address)][,(IND,[*]value)]

$$\left[,(\begin{Bmatrix}\text{SUR}\\ \text{CNV}\end{Bmatrix},[*]\text{byte address},[*]\text{byte length})\right]$$

where

BUF defines the byte address of the memory area where the transfer occurs.

TRL defines the maximum number of bytes to transfer. This value must be at least equal to the terminal's maximum transfer length, fixed at system generation. If TRL is not specified, the value assigned at system generation is used.

PTR defines the address of the event pointer (ECB) associated with the I/O at the execution of the procedure. This option is necessary if the user wants to perform an I/O wait (M:WAIT) or a test (M:CHECK) on multiple I/O events.

IND defines the entry of the active list. When this entry is absent, the active entry in the list is used. The value of the list index is not affected by the value of the IND option.

Note: The list index is the user's responsibility. The list index can be modified only by the execution of the M:MDFLST (IND) procedure.

SUR places the transmission line in a supervisory watch state for the reception of the Attention characters (characters defined at system generation). The I/O event is ended when the system detects one of these characters on the transmission line.

CNV defines a "conversational read" allowing the characters contained in the area whose address and the length are specified as parameters to be written before placing the line in reception for the message transmitted by the terminal.

M:WRITE PROCEDURE

Syntax

[label(s)] M:WRITE [*]dcb-address

,(BUF,[*]byte-address),(TRL,[*]value)

[,(PTR,[*]word-address)][,(IND,[*]value)]

[,(DIS)][,(FIN)]

where

BUF defines the byte address of the memory area from whence the transfer occurs.

TRL defines the number of characters to effectively transfer. This value cannot be zero or exceed the maximum transfer length defined for the terminal at system generation.

PTR defines the event pointer address related to the I/O at the execution of the procedure. This pointer is necessary if the user wants to perform an I/O wait (M:WAIT) or a test (M:CHECK) for multiple I/O requests.

IND defines the entry in the active character mode list specifying the transmission line involved with the I/O. When this option is absent, the entry is the one defined by the value of the index contained in the list. The value of the index in the character mode list is not affected by the value of the IND option

DIS indicates the end of vacation of the terminal. The effect of this option is to reset the line to the wait status for a call in the case of a switched telephone line equipped with automatic answering (disconnection of the line).

FIN indicates the logical end of the message.

M:DEVICE PROCEDURE

Syntax

[label(s)] M:DEVICE [*]dcb-address;

$$\left[,(\text{TRD},\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\})\right]\left[,(\text{TPH},\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\})\right]$$

$$\left[,(\text{MTR},\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\})\right]\left[,(\text{FRM},\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\})\right]$$

$$\left[,(\text{ECH},\left\{\begin{matrix}\text{ON}\\\text{OFF}\end{matrix}\right\})\right]\left[,(\text{IND},\left\{\begin{matrix}[*]\text{value}\\\text{ALL}\end{matrix}\right\})\right][,(\text{OPN})]$$

where

TRD requests (ON) the initiation of the paper tape reader for each record read by a read on the component or stops it (OFF).

TPH requests (ON) the edit of the paper tape punch for each record by a write on the terminal, or stops (OFF) the edit.

MTR requests (ON) the initiation of the terminal motor telecommand (generally used on rented lines) or stops it (OFF). When the option is ON, the motor starts upon the first write following the M:DEVICE procedure.

FRM specifies if the access method must add (ON) the format command (line skip and carriage return) to the records output by writes. The option (OFF) stops this consideration by the access method. When the paper tape punch is used, the access method edits the tape for an automatic reread of the tape.

ECH requests (ON) or stops (OFF) the return of characters received from the component in echoplex mode.

Note: When the M:DEVICE procedure is omitted, the access method initializes the commands TRD, TPH, MTR, FRM to OFF, and the command ECH to ON.

IND, value defines the value of the list index specifying the transmission line involved by the operation. When this option is absent, the index value is equal to the index value contained in the list.

ALL indicates that the specified operations are valid for all lines of the active list.

OPN allows the reinitialization of the initialization of a line operation which could not occur at the time of the M:OPEN. This option is valid only when several lines are linked to the DCB.

The lines of the group that could not be initialized at the M:OPEN are subject to an abnormal return during the execution of the M:CHECK procedure corresponding to the first I/O initiated on each line in error. The OPN option cannot be used with the ALL option.

M:CHECK PROCEDURE

The processing and the syntax are the same as in message mode (see M:CHECK under "Message Mode").

LINE MANAGEMENT

Each line can occupy different states. A state change can occur either by a user issuing a procedure or by a transmission event.

Line States (See Figure 9-4).

- Disarmed State. In this state, the various lines can neither transmit nor receive characters. The only procedure accepted is M:OPEN which initializes the different lines and activates them.

- Inactive State. The line has been initialized and all characters received on the line are entered in memory but ignored by the access method. The accepted procedure references are

  - READ (SUR) which sets the line in the supervisory watch state.

  - READ which sets the line in the Input state.

  - WRITE which sets the line in the Output state.

  - CLOSE which sets the line in the Disarmed state.

- Supervisory State. In this state only the 'Attention' characters entered by the local operator are detected and cause the user routine whose address is specified in the DCB (ABN – (class X2)) to be scheduled. All other characters entered are ignored. The reception of an attention character resets the line to an inactive state.

When the user is not waiting via the M:CHECK procedure, but initiates another I/O request (M:READ or M:WRITE), the line passes into the Input or Output state. The M:READ

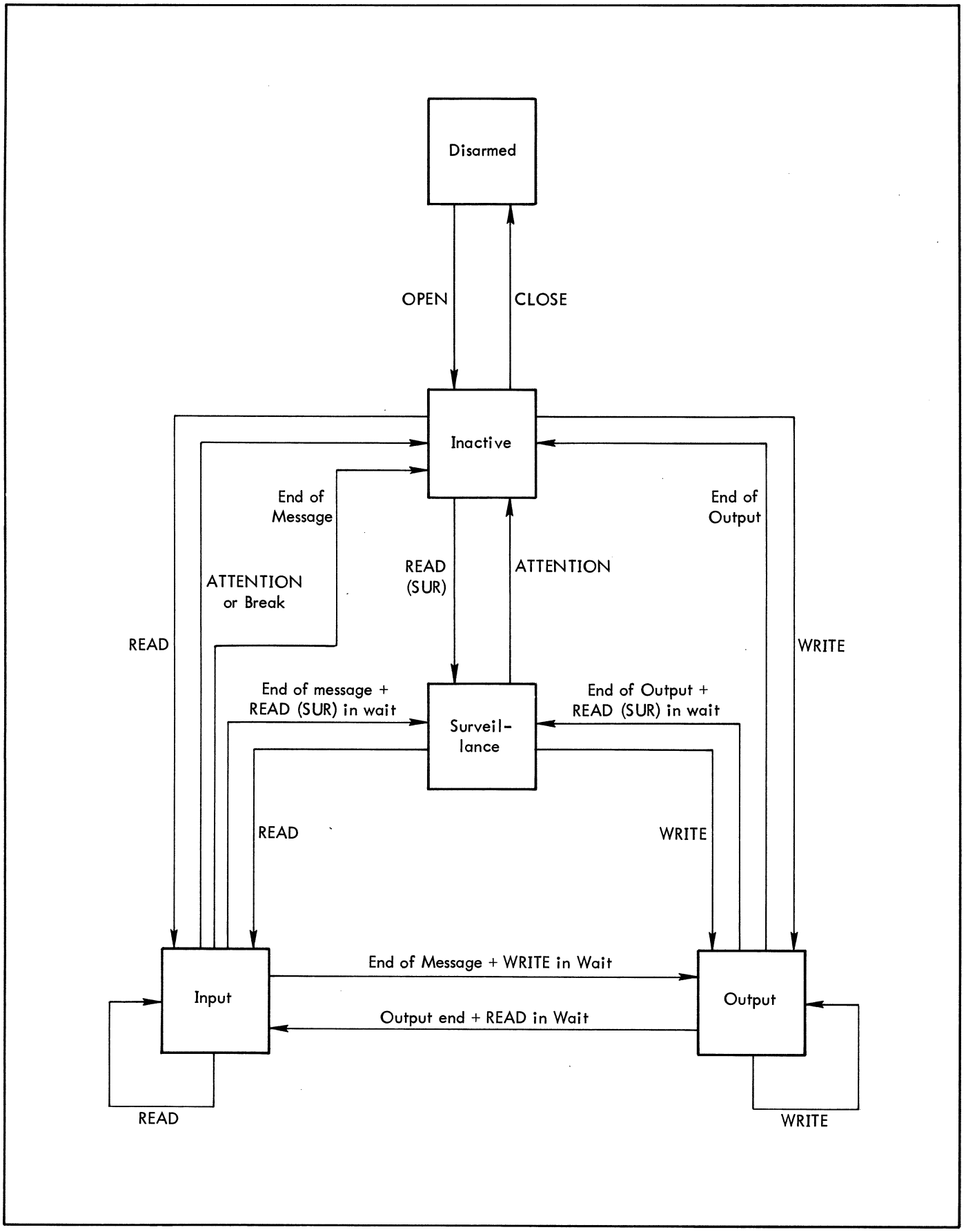


Figure 9-4. Flowchart of the States

(SUR) is then placed in a wait position of which the transition priority will be less than all the M:READ or M:WRITE procedure references. The execution of this procedure is not subject to a delay. However, it can be purged upon request of the user by M:MDFLST (RSS option) or M:CLOSE.

- Input State. When the user executes an M:READ procedure, the characters arriving on the line are transferred into his buffer until
  - The detection of an end-of-message character.
  - The depletion of the number of characters to read (specified by TRL option in the M:READ procedure).
  - The end of the delay fixed for the reception of a message (parameter fixed at system generation).

  In this state, the user can initiate several read and write requests on the same line. These requests are accepted and placed in a system wait queue in their order of arrival.

Upon the execution of an M:READ procedure, the reception of an end-of-message character places the line in the following states:

- Inactive when no read or write is waiting for execution.
- Output when a write is at the beginning of the wait queue.
- Supervisory when a READ (SUR) is waiting.
- Input when a READ is at the beginning of the wait queue.

The reception of an Attention character or Break causes the line to pass to the inactive state.

All I/O procedures in the wait queue are purged when the user issues an M:CHECK procedure corresponding to the read, the control is passed to the user at the address specified by ABN with an abnormal code. The I/O requests, then in a wait state, are released with an error code (IOB purged).

- Output State. When the user executes an M:WRITE procedure, his buffer is transmitted on the line until the depletion of the number of characters specified by the TRL option in the M:WRITE procedure.

In output state, the user can initiate simultaneous read and write requests on the same line. They are accepted and placed in a system wait queue in order of their arrival.

Upon the execution of an M:WRITE procedure, the end of output event (i.e., the depletion of the character count), or the reception of the ETX character places the line in the following states:

- Inactive when no READ or WRITE is waiting.
- Input when a READ is heading the wait queue.
- Supervisory when a READ (SUR) is waiting.
- Output when a WRITE is heading the wait queue.

INPUT STATE PROCESSING

Input state processing concerns the execution of the M:READ procedure. The transmission processing mode for character mode is described in the following paragraphs.

Echoplexing

The characters entered from the keyboard or read on the terminal reader are not printed simultaneously on the printer. The central station returns each character received to terminal output components. This mode of operation is called echoplexing. In this mode, the system controls the characters returned and can offer more services to the terminal operation. Two states are possible (on and off) and are commanded by the user issuing an M:DEVICE procedure.

In this type of functioning, TAM character mode assumes the following functions:

1. End of Message Characters Received (see Table 9-1). The end of message characters recognized by TAM character mode follow:

   CR Carriage return

   LF Line feed

   ETX End of text

   Attention 1

   Attention 2

   EM End of message

   HT Horizontal tabulation

   Table 9-8 presents the characters recognized by TAM on input and the processing characters stored in the user buffer and response to the terminal.

Table 9-8. Echoplex Mode in Function

Character Recognized	End of Message	Character Placed in the Buffer	Character Returned	Functions
CR		CR	CR+XOFF+LF	End of line and message
LF	Normal[t] 1	LF	CR+XOFF+LF	End of line and message
EM (Y^C)		EM	EM+XOFF	End of message
HT (TAB)		None	None	Software tabulation
ETX (C^C)	Abnormal[tt]	None	CR+XOFF+LF	End of special message
Attention 1		None	Attention 1 +CR+LF	Alarm 1
Attention 2		None	Attention 2 +CR+FL	Alarm 3
'LONG SPACE'	Error[ttt]	None	CR+XOFF+LF	Detection of a 'long space' or a line break
	No	None	\	Cancellation of the preceding character
CAN (X^C)	No	None	One to five characters	Cancellation of the message in progress
NUL	No	None	None	Buffer character
DEL	No	None	None	Delete or buffer character
DC1 (XON)	No	None	DC1	Initiation of the reader
DC2 (TAPON)	No	None	DC2	Initiation of the punch
DC3 (XOFF)	No	None	DC3	Reader stop
DC4 (TAPOFF)	No	None	DC4	Punch stop

[t]Control is passed to the user with no abnormal or error return.

[tt]Control is passed to the user exit routine at the address specified by ABN parameter in the DCB with abnormal code.

[ttt]Control is passed to the user exit routine at the address specified by ERR with an error code.

2. Filtered Characters. The following characters, if they are keyed by the operator or read on the tape reader, are detected by TAM but are not transmitted in the user buffer (see Table 9-8):

NUL	Null
DEL	Delete or erase character.
DC1(XON)	Initiate the paper tape reader.
DC2(TAPON)	Initiate the paper tape punch.
DC3(XOFF)	Stop the paper tape reader.
DC4(TAPOFF)	Stop the paper tape punch.

3. Cancellation Characters (see Table 9-1). During the execution of an M:READ procedure, the local operator has the capability to nullify the preceding character transmitted by keying a specific character (e.g., \) defined at system generation (see XOS/SM Reference Manual 90 17 66). In this case, the last characters (\) received one after the other, delete as many characters in the user buffer.

   During the execution of an M:READ procedure, the local operator has the capability to delete the entire message by keying in a character defined at system generation; for example, CAN (CAN = X + Control).

   In this case, all the user buffer is deleted and TAM character mode places a sequence of one to five characters defined for the line at system generation (see XOS/SM Reference Manual, 90 17 66) in the user's buffer.

4. Parity Check. During the execution of the M:READ procedure, the parity of the characters keyed by the local operator is controlled by the access method. Upon the detection of a parity error in a character, the character is replaced by a character defined at system generation; for example (↓), which is placed in the user buffer and sent to the terminal printer. The operator has the capability to cancel the character in error with the character and to key it again.

5. Translation. The emission of characters on the line is executed in ANSCII. During the execution of an M:READ procedure, the translation of characters received (ANSCII → EBCDIC) is performed by the access method, except if the user has specified that his work code is ANSCII.

6. Attention Characters. TAM character mode recognizes two Attention characters which are defined at system generation.

   The M:READ procedure (SUR) enables the user to be warned of the reception of one of these characters by an abnormal exit to the address specified by the parameter ABN (of the DCB) at the execution of a check of the read survey.

   When the character 'Attention 1 or 2' is received during the execution of a M:READ or M:READ (CNV), the M:READ (SUR) then waiting, is given along with the M:READ in question an abnormal code X'12' or X'13'. All the I/Os waiting are purged and transferred to the user with the abnormal code X'22'.

7. Break Detection (Long Space). A Long Space is initiated by use of the Break Key on the terminal keyboard or by an untimely line break.

   During the execution of an M:READ or M:WRITE procedure, a Long Space detected on the line is considered by the access method as an end of message. Upon issuing an M:CHECK, an error exit occurs at the address specified by ERR and an error code (X'27') is passed to the user exit routine.

8. Response Timing. At each initiation of the M:READ procedure, except M:READ (SUR), TAM character mode initializes a timer. When the timer has elapsed and the end of message has not occurred, the event is posted with an error code of X'24'.

   Upon checking of the I/O request an error exit occurs at the address specified by ERR and the error code X'24' is passed to him. All I/O requests then waiting are purged.

9. Paper Tape Use. The use of paper tape can be initiated by the user program or at the terminal operator. TAM character mode performs the same functions whether the data comes from the paper tape reader or the terminal keyboard. In character mode, the composition of the paper tape by the terminal operator is simple.

   Example: The operator wants to send the message "MESSAGE", the characters to type to prepare the tape are

   where

   TAPE ON is the character to type at the beginning of the text to initiate the paper tape punch.

   MESSAGE is the text of the message to be sent.

(fin) is the end of message.

[(fin)] is required for the off-line control to the printer.

(DEL) (DEL) (DEL) are the characters corresponding to the time necessary to stop the tape.

The combination of (fin) end-of-record characters that can be used are:

(fin)	[(fin)]
CR	LF
LF	CR
EM	-
ETX	-

OUTPUT STATE PROCESSING

Output State processing concerns the execution of the M:WRITE procedure. The message contained in the user buffer is transmitted character by character to the line specified by the user by means of the direct access list.

1. End of Output. TAM character mode transmits the characters contained in the user buffer until the character count is satisfied.

   The editing of the records is performed by TAM if the user requests it by the M:DEVICE procedure (FRM, ON) otherwise all editing is entirely under control of the user program.

2. Translation. TAM assures the translation of characters (EBCDIC → ANSCII) when the user has specified his working code is EBCDIC. Otherwise, the characters are transmitted on the line without any translation.

3. Suspension by ETX Character. During the execution of an M:WRITE procedure, entering of the ETX character by the local operator ends transmission of the message. TAM returns the characters CR+XOFF+LF to the terminal. Upon issuing of a check by the user, a normal end of transmission is passed to the user.

4. Paper Tape Usage. The initiation of the tape punch can be accomplished by the user program executing the M:DEVICE procedure (option TPH). In this case, before sending the corresponding message to the execution of the first M:WRITE procedure, the punch motor is started by a TAPON character being transmitted by TAM and stopped after the transmission of the first character of the message relative to the WRITE procedure by a TAPOFF transmitted by TAM. When the user has also requested editing – M:DEVICE (FRM, ON) procedure – the edit is assured so that the paper tape can be automatically reread by the program. The paper tape punch can also be operated by the terminal operator by intervening on the manual command and by keying TAPON and TAPOFF on the keyboard. All the characters sent to the paper tape punch are also sent to the printer. Punching cannot occur without printing.

5. Attention Characters. During the execution of an M:WRITE procedure the reception of an attention character does not interrupt the output operation in progress. Upon the check of the I/O, a normal condition is passed to the user. The user is warned of the reception of this attention character if an M:READ (SUR) is waiting; upon issuing an M:CHECK, an abnormal condition is passed to him. When an M:READ procedure is located in the wait queue at the time the attention character is received, this attention character is placed in memory and at the execution of the M:READ procedure the event associated with it is posted with an abnormal code corresponding to the Attention type received; all other procedures following the M:READ procedure in the wait queue are then purged.

SUPERVISORY STATE PROCESSING

This mode corresponds to the execution of the M:READ (SUR) procedure with no other I/O waiting. The only characters detected by the access method are the characters Attention 1 and Attention 2, for which the configuration was specified at system generation. Upon the checking of the I/O, the user regains control at the address specified by ABN and the associated abnormal code is passed to him depending on the character received.

ABNORMAL AND ERROR CONDITIONS

The processing of abnormalities and errors is performed in an identical manner whether it is the message or character mode.

The codes specific to the character mode appear in Tables 9-9 and 9-10.

PROGRAM ABORT ERRORS

The codes passed to the user are the same as in message mode.

Table 9-9. Abnormal Codes During File Processing

Hexadecimal Code	Meaning	Origin	Class
X'10'	End of text (ETX)	M:READ	X2
X'11'	Null byte count before reception of the end of message	M:READ	X2
X'12'	End of record: Attention 1	M:READ(SUR) or M:READ or M:READ(CNV)	
X'13'	End of record: Attention 2	M:READ(SUR) M:READ M:READ(CNV)	X2
X'14'	End of record: HT (tabulation)	M:READ or M:READ(CNV)	X2
X'1A'	M:READ (SUR) refused because another M:READ (SUR) is already active or waiting	M:READ(SUR)	X3
X'1D'	I/O requested on a line deactivated in the list by the M:MDFLST (option SKP) procedure	M:READ M:WRITE M:DEVICE	X3
X'1E'	List index greater than number of terminals in the list	M:READ M:WRITE M:DEVICE M:MDFLST	X3

Table 9-10. Errors During File Processing

Error Class	Hexadecimal Code	Meaning	Origin
Recoverable	X'22'	I/O request purged	M:READ or M:READ (SUR) or M:WRITE
	X'24'	End of record not received after the fixed time delay	M:READ
	X'26'	Line not activated during OPEN. The user must, in this case, initiate an M:DEVICE (OPN) procedure	M:READ M:WRITE M:DEVICE
	X'27'	Detection of a long space in line	M:READ M:WRITE

Table 9-10. Errors During File Processing (cont.)

Error Class	Hexadecimal Code	Meaning	Origin
Recoverable	X'29'	Line in permanent error	M:READ or M:READ (SUR) or M:WRITE
Irrecoverable error	X'2A'	Line locked	M:READ or M:READ (SUR) or M:WRITE
	X'2B'	Device controller in permanent error	M:READ or M:READ (SUR) or M:WRITE

OPERATOR COMMUNICATIONS IN CHARACTER MODE

Device Controller Errors. These messages are of the no response type and are labeled in the following manner:

!! IOS ERR xy CTMC z

where

xy is the error number.

z is the physical number of the character mode transmission coupler.

The errors of this type which can occur in the character mode transmission are shown in Table 9-11.

Table 9-11. Device Controller Error Codes

Code	Meaning	OP Response
ERR02	I/O address not recognized upon an SIO, TIO, HIO	None
ERR04	Nonoperational device controller	None

When these error messages appear frequently on the same coupler, the operator can prohibit I/Os on all the lines of the coupler by locking each line by a UNIT command.

Adapter Errors. These messages are also of the no response type. They are labeled in the following manner:

!! IOS ERR xy 'line-id'

where

xy is the error number.

line-id is the symbolic name of the line.

The errors of this type are shown in Table 9-12.

Table 9-12. Adapter Error Codes

Code	Meaning	OP Response
ERR06	Nonoperational adapter or modem	None
ERR20	Erroneous adapter blocking the operation of all the others. The line is automatically locked by the system in this case.	None

The operator can lock the line by a UNIT command when the error 06 appears frequently. In the case of error 20, the locking is done automatically by the system.

APPENDIX A. SYNTAX CHARTS FOR !ASSIGN COMMAND, M:DCB/M:SETDCB PROCEDURE, AND ACCESS METHOD PROCEDURES

Chart A-1. ASSIGN Control Command

Command	Comments	DCB Parameters
!ASSIGN op-label [,{MTN / FRE}],FIL[,option, . . .][,DCB,parameter, . . .]	See DCB Parameters.	General Processing Parameters:
FIL Options:		(ORG,{%C / %I / %P / %D})
(CTG[,acct-number])	See "Cataloged Files", Chapter 6.	(NBF,%value)
(STS,{NEW / OLD / MOD})	Default = NEW.	(MOD,{%BIN / %BCD / %EBC / %PK / %UPK})
(LNK,%op-label-2)		
(UNT,{{DM / MT}[,(VOL,%serial-no[, . . .])][,(SQN,%sequence-no)] / AC,acct-number / OP,op-label-3[,step]})	Default = user's account volume. VOL default = public volume.	
({PAR / MNT,number / DEF,op-label-4})	Default = PAR for disk and MNT for magnetic tapes.	Block Level Parameters:
(DSP,{DMT / KEP / RET})	Default = DMT.	(BHR,%value)
(SIZ[,({%RST / %SEP})],%size[,%increment])	Default = installation-determined file size.	(BKL,%value)
(NAM,%name[,{%absgen[,%version] / relgen}])	See Chapter 6 for absolute and relative generation specifications.	(MXL,%value)
(PRT{,NCT / [[,(R,{ALL / NO / %account[, . . .]})][,(W,{ALL / NO / %account[, . . .]})][,PAS]]})	Default = R,ALL and W,NO. If PRT appears, at least one of the suboptions must be chosen. (NCT excludes any others.)	NBC
(RET,%period)		Record Level Parameters:
!ASSIGN op-label [,{MTN / FRE}],DEV[,(STS,{NEW / OLD / MOD})][,option, . . .][,DCB,parameter, . . .]	See DCB Parameters. Unless file status NEW is applicable, the STS option must appear in the position shown in the command syntax, i.e., immediately following "DEV" and prior to any other DEV options.	%DLC
DEV Options: {{MT / M7 / DM}[,(VOL,serial-no[, . . .])][,({PAR / MNT,number / DEF,op-label-4})] / {CR / CP / LP / MT / M7 / DM} / (ADR,logical-address) / {IN / OUT / {SLP / SCP}[,NKP][,(STA,%terminal-id)]}}	VOL default = public volume.	(FRM,{%F / %V / %U})
		(KYL,%value)
		(KYP,%value)
		(REL,%value)
		Device Level Parameters:
		(CNT,%value)
		(DTA,%value)
		(LIN,%value)
		(SEQ[,%sequence-id])
		(SPC,%value[,%heading])
		(TAB,%value[, . . .])
!ASSIGN op-label [,{MTN / FRE}],OPL,op-label-1[,DCB,parameter, . . .]	See DCB Parameters.	{%VFC / %NVF}
!ASSIGN op-label [,{MTN / FRE}],DUM	No options apply.	None.

CHART A-1. ASSIGN CONTROL COMMAND

(fold out)

Chart A-2. M:DCB/M:SETDCB Procedure

PROCEDURE SYNTAX

[label(s)] M:DCB (OPL,'oplabel')[,parameter, . . .]

[label(s)] M:SETDCB [*]dcb-adr[,(OPL,{*adr-label / 'op-label'})][,parameter, . . .]

(Parameter addresses and values in M:SETDCB may be indirect.)

PARAMETERS

Parameter	Definition	Parameter	Definition
General Processing Parameters:		Record Level Parameters:	
(ABN,address[,class-code, . . .])	Abnormal Return	(DLC)	Deletion Control
(AM,{AS / AI / AP / VS / VD / BD})	Access Method	(FRM,{F / V / U})	Record Format
		(KYL,value)	Key Length
(BFA,byte-address)	Buffer Address	(KYP,value)	Key Position
(ERR[,address])	Error Return	({LOC / MOV})	Locate/Move
(MOD,{BIN / BCD / ECB / PK / UPK})	Data Mode	(REL,value)	Record Length
(NBF,value)	Number of Buffers		
(NRT)	No Retry	Device Level Parameters:	
(ORG,{C / I / P / D})	File Organization	(CNT,value)	Page Count
		(DTA,value)	First Print/Punch Column
(SIM,value)	Simultaneous Operations	(HDR,value,address)	Page Header
(TLB,address)	User Label Area	(LIN,value)	Lines Per Page
Block Level Parameters:		(SEQ['identifier'])	Sequence Numbering
(BHR,value)	Block Header	(SPC,value-1,value-2)	Line Spacing
(BKL,value)	Block Length	(TAB,value, . . .)	Tab Setting
(MXL,value)	Maximum Transfer Length	({VFC / NVC})	Vertical Format Control
(NBC)	No Block Count		

CHART A-2. M:DCB/M:SETDCB PROCEDURE

(fold out)

Chart A-3. Assisted Sequential Access Method (ASAM) Procedure

I/O Procedures	Processing Mode				Format		
	I	B	O	U	F	V	U
[label(s)] M:GET [*]dcb-adr,(REC,{[*]badr-1 / adr-2}) {MOV	X	X		X	X	X	X
{LOC	X	X		X	X	X	X
,(RSA,[*]adr-3)	X	X		X	X	X	X
[label(s)] M:PUT [*]dcb-adr,(REC,{[*]badr-1 / adr-2}) {MOV			X	X	X	X	X
{LOC			X		X	X	X
,(ARS,[*]value)			X			X	X
[label(s)] M:TRUNC [*]dcb-adr	X	X	X	X	X	X	
[label(s)] M:DELREC [*]dcb-adr				X	X	X	
[label(s)] M:CVOL [*]dcb-adr	X		X	X	X	X	X
[label(s)] M:NOTE [*]dcb-adr,(RCI,[*]adr-1)	X	X	X	X	X	X	
[label(s)] M:POINT [*]dcb-adr,(RCI,[*]adr-1)	X	X		X	X	X	
[label(s)] M:DEVICE [*]dcb-adr,{(CHF,[*]badr)			X		X	X	
{(PAG)			X		X	X	
{(POS,{BKS / FWS / BOF / EOF})	X	X	X†	X	X	X	X

†Only BOF or EOF are allowed in this mode, i.e., BKS/FWS are excluded.

Chart A-4. Assisted Indexed Access Method (AIAM) Procedure

I/O Procedures		Processing Mode			Format	
		I	O	U	F	V
[label(s)] M:GET [*]dcb-adr,REC,{[*]badr-1 / adr-2})	{MOV	X		X	X	X
	{LOC	X		X	X	X
[,(RSA,[*]adr-3)]		X		X	X	X
[,(KEY,{[*]badr-4 / adr-5})]		X		X	X	X
[label(s)] M:PUT [*]dcb-adr,(REC,[*]badr-1)	MOV only		X	X	X	X
[,(ARS,[*]value)]			X	X		X
[,(NWK)]				X	X	X
[,(DFW)]				X	X	X
[label(s)] M:TRUNC [*]dcb-adr		X	X	X	X	X
[label(s)] M:DELREC [*]dcb-adr				X	X	X

CHART A-3. ASSISTED SEQUENTIAL ACCESS METHOD (ASAM) PROCEDURE

CHART A-4. ASSISTED INDEXED ACCESS METHOD (AIAM) PROCEDURE

(fold out)

Chart A-5. Assisted Partitioned Access Method (APAM) Procedure

I/O Procedures		Processing Mode			Record Format	
		I	O	U	F	V
[label(s)] M:STOW [*]dcb-adr,(KEY,{[*]badr-1 / adr-2})[,({ADD / SYN / DEL})]			X		X	X
[label(s)] M:FIND [*]dcb-adr,(KEY,{[*]badr-1 / adr-2})		X		X	X	X
[,(RCI,[*]adr-3)]		X		X	X	X
[label(s)] M:GET [*]dcb-adr,(REC,{[*]badr-1 / adr-2})	MOV	X		X	X	X
	LOC	X		X	X	X
[,(RSA,[*]adr-3)]		X		X	X	X
[label(s)] M:PUT [*]dcb-adr,(REC,{[*]badr-1 / adr-2})	MOV		X	X	X	X
	LOC		X		X	X
[,(ARS,[*]value)]			X			X
[label(s)] M:TRUNC [*]dcb-adr		X	X	X	X	X
[label(s)] M:DELREC [*]dcb-adr				X	X	X
[label(s)] M:NOTE [*]dcb-adr,(RCI,[*]adr-1)		X	X	X	X	X
[label(s)] M:POINT [*]dcb-adr,(RCI,[*]adr-1)		X		X	X	X

CHART A-5. ASSISTED PARTITIONED ACCESS METHOD (APAM) PROCEDURE

(fold out)

Chart A-6. Virtual Sequential Access Method (VSAM) Procedure

I/O Procedures	Processing Mode			
	I	O	B	S
[label(s)] M:READ [*]dcb-adr,(BUF,{[*]badr-1 / adr-2})	X		X	X
[,(TRL,[*]value)]	X		X	X
[,(PTR,[*]adr-3)]	X		X	X
[label(s)] M:WRITE [*]dcb-adr,(BUF,{[*]badr-1 / adr-2})		X		X
[,(TRL,[*]value)]		X		X
[,(PTR,[*]adr-3)]		X		X
[label(s)] M:CHECK [*]dcb-adr[,(PTR,[*]adr-1)]	X	X	X	X
[,(RSA,[*]adr-2)]	X		X	X
[label(s)] M:CVOL [*]dcb-adr	X	X		
[label(s)] M:NOTE [*]dcb-adr,(RCI,[*]adr-1)	X	X	X	X
[label(s)] M:POINT [*]dcb-adr,(RCI,[*]adr-1)	X		X	X
[label(s)] M:DEVICE [*]dcb-adr,{(CHF,[*]badr) / (PAG) / (POS,{BKS / FWS / BOF / EOF}) / (WEOF)}	X	X X X X	X	X X

CHART A-6. VIRTUAL SEQUENTIAL ACCESS METHOD (VSAM) PROCEDURE

(fold out)

Chart A-7. Virtual Direct Access Method (VDAM) Procedure

I/O Procedures	Comments
[label(s)] M:READ [*]dcb-adr,(BUF,{[*]badr-1 / adr-2}),(ADR,[*]value-1)[,(TRL,[*]value-2)] [,(PTR,[*]adr-3)]	Input and Scratch modes.
[label(s)] M:WRITE [*]dcb-adr,(BUF,{[*]badr-1 / adr-2}),(ADR,[*]value-1)[,(TRL,[*]value-2)] [,(PTR,[*]adr-3)]	Output and Scratch modes.
[label(s)] M:CHECK [*]dcb-adr,[,(PTR,[*]adr-1)][,(RSA,[*]adr-2)]	RSA effective for read only.

Chart A-8. Basic Direct Access Method (BDAM) Procedure

I/O Procedures
[label(s)] M:READ [*]dcb-adr,(BUF,{[*]badr-1 / adr-2}),(ADR,[*]relative-sector-address)[,(TRL,[*]value)][,(PTR,[*]adr-3)]
[label(s)] M:WRITE [*]dcb-adr,(BUF,{[*]badr-1 / adr-2}),(ADR,[*]relative-sector-address)[,(TRL,[*]value)][,(PTR,[*]adr-3)]
[label(s)] M:CHECK [*]dcb-adr,[,(PTR,[*]adr-1)][,(RSA,[*]adr-2)]
[label(s)] M:CVOL [*]dcb-adr

CHART A-7. VIRTUAL DIRECT ACCESS METHOD (VDAM) PROCEDURE

CHART A-8. BASIC DIRECT ACCESS METHOD (BDAM) PROCEDURE

(fold out)

APPENDIX B. ABNORMAL, ERROR, AND ABORT CONDITIONS

This appendix lists the abnormal, error, and abort conditions which can occur during operation of a user's program. The abnormal and error conditions cause the user's program either (1) to be interrupted temporarily for processing of the condition if the user so specifies, or (2) to be aborted.

An abnormal condition is not considered detrimental to the user's program if the user elects to process the condition with an abnormal return routine (specified in his DCB).

An error condition is the result of bad performance of a device or improper specifications. A message is sent to the job control file. The user can elect to retain control of processing by means of an error return routine (specified in his DCB).

An abort condition is irrecoverable and the user's program is unconditionally aborted.

Abnormal conditions and error conditions are referred to collectively as exception conditions. When an exception condition occurs, the system responds either by

1. Taking a branch specified by the user to a routine which processes the exception condition. (The routine addresses are specified in the DCB. For abnormal conditions, the class code of the particular condition must also be specified in the DCB.)

2. Aborting the job, if no branch was specified by the user.

Therefore, all abnormal conditions and error conditions are also possible abort conditions. When a program (i.e., job-step), is aborted, the system will either continue processing the subsequent job-step(s), if any, or abort the entire job. The !SWITCH card allows the user to predetermine whether or not the job-steps following the offending step are to be processed (if possible). The numerical portion of the error code associated with the particular condition indicates whether or not processing can continue with subsequent job-steps. When the numerical value is less than X'80', the job can continue with the next job-step. Otherwise, the entire job is aborted.

Whenever a job or job-step is aborted, the following message is sent to the job control file:

ABORT literal abort code program status doubleword

where

literal is either an operational label of the file being processed when an input/output condition occurred that caused the abort, or a four letter code that helps the user identify the origin of the condition.

abort code is a code identifying the specific condition that caused the abort.

program status doubleword is the PSD at the time the condition that caused the abort was detected.

GENERAL XOS ABORT CODES

In addition to listing the abnormal, error, and abort conditions, this appendix provides pertinent information about each of the conditions. This information may include:

Abort Code. The code given in the ABORT message that identifies the specific exception or abort condition. All abort conditions have been given abort codes. All exception conditions, except those related to TAM, have also been given abort codes. The first character of the code (always alphabetic) identifies the monitor module that detected the condition (compare ABN/ERR code).

ABN/ERR Code. A two digit hexadecimal number which is stored into the leftmost byte of register 5 when an exception condition occurs and the program is not aborted. Note that it is always the last two digits of the corresponding abort code.

Cause. A descriptive statement of the cause of the condition.

Class. A code (X1, X2, X3, X4, X5, or X6) that indicates the class to which an abnormal condition belongs.

Origin. A list of the procedures, segments of procedures, or operations during which the condition can occur. This information is supplied wherever possible to help the user pinpoint the specific cause of the condition.

ABORT CONDITIONS

Abort Code	Cause
A36	Privileged and nonexistent instruction.
A38	Nonexistent instruction.
A3B	Memory address protected and nonexistent.
A3C	Nonexistent memory address.
A3E	Privileged instruction in the slave mode.
A3F	Memory protection violation.
A41	Unimplemented instruction trap.

Abort Code	Cause
A42	Stack overflow trap (second occurrence).
A43	Fixed-point overflow.
A44	Floating-point fault.
A45	Decimal arithmetic fault.
A46	Watch dog timer runout.
A47	Stack overflow trap (first occurrence).
A48	Illegal CAL1 or CAL1 code.
A49	Illegal instruction CAL2.
A4A	Illegal instruction CAL3.
A4B	Illegal instruction CAL4.
A4C	M:INT return.
A4D	Return time allocated has elapsed (M:STIMER return).
A4E	Pointer of user stack in error.
A50	Instruction exception trap.
A7F	Maximum time allocated was exceeded.

DEVICE AND VOLUME ALLOCATION

ABORT CONDITIONS

Abort Code	Cause	Origin
D01 (BLAB)	Insufficient memory available to update disk volume label.	Volume exchange
D02 (BLAB)	I/O error reading to update disk volume label during volume exchange.	Volume exchange
D03 (APMD)	Required nonsharable volume already in use during device allocation.	Device allocation
D03 (BLAB)	I/O error writing to update disk volume label during volume exchange.	Volume exchange
D04 (ADCT)	Insufficient memory available to request reallocation of a device to a required volume.	Device allocation
D04 (CCLK)	Required device locked by computer operator during volume exchange.	Volume exchange
D04 (CAPM)	Required volume already in use during volume exchange.	Volume exchange
D04 (CRQB)	Insufficient memory available to request mounting or dismounting a volume.	Volume exchange
D07 (ASCG)	Insufficient memory available or entry not found in an attempt to identify an account volume through the Supercatalog.	Device allocation
D08 (APMP)	Required nonsharable pseudovolume already in use.	Device allocation
D09 (ALNA)	Insufficient memory available to request assignment or confirmation of device use.	Device allocation
D0B (AGSP)	Insufficient memory available for work space in allocation of devices for volume mounting.	Device allocation
D0C (ALCK)	Device required to mount a volume is locked by the computer operator.	Device allocation
D0F	Requested nonsharable volume has been found to be in use during automatic volume recognition.	Device allocation
D24 (BLAB)	Insufficient memory available for updating disk volume label during device release.	Device release
D25 (BLAB)	I/O error writing to update volume label during device release.	Device release
D26 (BLAB)	I/O error reading to update disk volume label during device release.	Device release
D40 (BSCG)	Insufficient memory available to update Supercatalog after pseudovolume release.	Volume release
D41 (BSCG)	I/O error reading to update Supercatalog after pseudovolume use.	Volume release
D42 (BSCG)	I/O error writing to update Supercatalog after pseudovolume use.	Volume release
D50 (GGSP)	Insufficient memory available to update device usage accounting records.	Device usage accounting
D51 (GLLG)	I/O error reading or writing to update device usage accounting records.	Device usage accounting

USER SERVICES

ABORT CONDITIONS

Abort Code	Cause	Origin
Gxx	User job issued M:ERR, with abort (xx) code specified by the user.	M:ERR
G00	User job initiated abort, but did not specify an abort code number.	M:ERR
G01	Insufficient free pages of dynamic storage available.	M:GL
G02	Writing on listing log impossible during Print, Type, or Key-in request.	M:PRINT, M:TYPE, M:KEYIN
G03	Core space insufficient to process the request.	M:TYPE, M:KEYIN
G04	A request to type or print a message where message was outside the user program area.	M:TYPE, M:KEYIN
G05	M:INT or M:STIMER has been issued in a user program section that was entered due to event occurrence (i.e., I/O ERR, ABN, or M:TRAP).	M:INT, M:STIMER
G06	Incorrect parameters specified.	M:INT M:STIMER
G07	With OST option, the address does not indicate an unprotected core area.	M:TRAP
G08	Incorrect beginning address for freeing memory space.	M:FSP
G09	With CANCEL option, no previous M:STIMER issued.	M:STIMER
G10	Number of events expected greater than number of ECB addresses provided.	M:WAIT
G11	Invalid user data area for M:TIME date group.	M:TIME
G12	Invalid request for M:IDLE.	M:IDLE
G20	M:LDTRC issued within a user abnormal routine (i.e., I/O ERR, trap, or interrupt).	M:LDTRC
G21	Common space not available on M:LDTRC.	M:LDTRC
G40	M:LINK issued in a user abnormal routine (i.e., I/O ERR or ABN; M:TRAP or interrupt).	M:LINK
G41	Insufficient memory space to process the M:LINK.	M:LINK
G42	Insufficient memory space to process the M:LINK.	M:LINK
G43	I/O error on temporary file creation during save of calling program environment.	M:LINK
G44	I/O error on temporary file reading during restore of calling program environment.	M:LINK
G7F	Maximum time allocated was exceeded as specified on !LIMIT command.	M:STIMER
G80	Maximum number of pages specified by !LIMIT exceeded.	M:PRINT

JOB MANAGEMENT

ABORT CONDITIONS

Abort Code	Cause
I01	Insufficient core space to initialize job.
I02	I/O error when reading system device volume blocks (DVB) from system disk.
I03	Core space insufficient to emit a message to the operator that job is initialized.
I04	I/O error when reading system !ASSIGN command table (TBASS) from system disk.
I05	Error in file generation group catalog entry.
I11	I/O error when reading system step information table (SIT) from system disk.
I12	I/O error when reading system !MESSAGE tables from system disk.
I13	I/O error when reading system !TITLE tables from system disk.
I14	I/O error when reading system processor options tables introduced on !PROCESSOR command from system disk.
I15	Core space insufficient to schedule the job step.
I43	I/O error when reading system !ASSIGN tables (TBASS) from system disk.
I81	Insufficient resources to satisfy !RESOURCE requests.
I82	I/O error when reading system tables containing information to start the job.
I83	Insufficient core space to build the operational label table during mount request.

LOADER

ABORT CONDITIONS

Abort Code	Cause	Origin
J01	Insufficient core space to build LM DCB or read load module HEAD/TREE tables into core.	Root Loader
J02	Unable to open LM file for reading load module.	Root Loader
J03	I/O error when reading load module HEAD and TREE records.	Root Loader
J04	The program load "BIAS" is not compatible with that of the installation.	Root Loader
J05	Program too large for the available virtual memory or !LIMIT too low.	Root Loader
J06	No start address specified for program.	Root Loader
J10	I/O error when reading a load module segment.	Segment Loader
J11	Unable to find requested overlay segment name in tree table of load module.	Segment Loader
J12	Requested segment (protection type 01) will overlay DCB (protection type 10). Insufficient memory space.	Segment Loader
J20	I/O error on write of requested debug dump on listing log.	Debug Dump
J21	Insufficient core space to write requested debug dump on listing log.	Debug Dump
J80	Maximum number of printed page output exceeded during print of requested debug dump.	Debug Dump

FILE PROCESSING

ABNORMAL CONDITIONS (ABN)

Abort Code	ABN Code	Cause	Origin
O01	01	End of file.	M:CHECK M:GET M:CVOL
O02	02	End of volume.	M:CHECK
O07	07	Incorrect length; some data ignored because user read-length specified was less than physical record length.	M:CHECK
O09	09	Access out of file limits.	M:WRITE (ADR) M:DEVICE (BKS/FWS) M:POINT
O0A	0A	File saturated or no more volumes to permit switch.	M:PUT, M:WRITE, M:CVOL
O0B	0B	Key not in order.	M:PUT (indexed O mode)
O0D	0D	Key not present.	M:FIND M:STOW (DEL) M:GET (KEY)
O0F	0F	Key already present.	M:STOW (ADD, SYN) M:PUT (indexed)
O13	13	More than 23 disk extents required.	M:PUT M:WRITE

ERROR CONDITIONS (ERR)

Abort Code	ERR Code	Cause	Origin
O20	20	Invalid sequence in the block numbers on tape.	M:GET, M:CHECK
O21	21	MT data ignored (incorrect length).	M:GET (formats F and V)
O22	22	Queued I/O request purged due to bad termination of prior I/O.	M:CHECK

Abort Code	ERR Code	Cause	Origin
O23	23	Device permanent error.	All I/O procedures except M:READ and M:WRITE
O2A	2A	Device locked by operator.	All I/O procedures except M:READ and M:WRITE
O2C	2C	Irrecoverable I/O error.	All I/O procedures except M:READ and M:WRITE

Abort Code	Cause
O5B	Number of simultaneous I/O operations requested exceeds DCB SIM specification or M:CHECK not issued after SIM I/O requests.
O5C	I/O operations issued on a closed DCB.
O5D	I/O requests issued to a DCB that has been closed with hold (temporary close) specified.
O5E	Erroneous DCB address.
O5F	I/O request issued is not allowed for the file organization/access method specified in the DCB.
O60	I/O request issued in a sequence that is illegal (i.e., M:PUT issued before an M:STOW in APAM file).
O61	!LIMIT specification for number of cards punched or pages printed is exceeded.
O6A	Insufficient memory to satisfy I/O request.

ABORT CONDITIONS

Abort Code	Cause
O47	Key position or key length invalid. Key or portion of key outside record limits.
O50	Negative response given by operator to a user issued M:DEVICE change form.
O51	Attempt to process a null record length.
O52	Record length greater than block length (REL > DCBBKL).
O53	Record length greater than defined record length (REL > DCBREL).
O54	Transfer length greater than maximum length specified (TRL > DCBMXL).
O55	Erroneous block length.
O56	During update of AIAM record, the key was changed by the user. This may only be done by NWK option during write.
O57	During update of V or U format record, the record length was changed.
O58	M:DELREC issued for file created without DLC option specified in DCB.
O59	Buffer address or pointer error. Usually due to not specifying a byte address in user program area.
O5A	M:WAIT issued with no I/O in progress.

OPEN/CLOSE

ABNORMAL CONDITIONS (ABN)

Abort Code	ABN Code	Cause	Origin
S01	01	End of file (DUM assignment).	M:OPEN
S02	02	End of volume.	M:OPEN
S0A	0A	Impossible to switch volumes.	M:CVOL
S0C	0C	!ASSIGN error: cataloged or generation file not resident on account volume.	M:OPEN
S10	10	User labels at the beginning of file.	M:OPEN
S11	11	User labels at the beginning of volume.	M:OPEN
S12	12	User labels at the EOF or EOV.	M:CLOSE (M:OPEN)
S13	13	Disk saturated (fields or quanta).	
S14	14	Attempt to open a tape[t] or disk file with status OLD or MOD and file does not exist.	M:OPEN

[t]The user receives control for this abnormal for tapes only when opened in OLD or MOD state and the file positioning parameter PID is specified.

Abort Code	ABN Code	Cause	Origin
S15	15	Attempt to open an existing disk file with status NEW.	M:OPEN
S16	16	File ID does not agree (tape only).[tt]	M:OPEN, CID
S18	18	Attempt to overwrite an unexpired file.	M:OPEN
S19	19	Password request.	M:OPEN

[tt]The user receives control for this abnormal only for files opened in OLD or MOD state with the CID option.

ERROR CONDITIONS (ERR)

Abort Code	ERR Code	Cause
S23	23	Permanent device error.
S2A	2A	Device locked by operator.
S2C	2C	Irrecoverable permanent error.

ABORT CONDITIONS

Abort Code	Cause
S40	Incompatibility between access method and file organization.
S41	Incompatibility between access method and peripheral type.
S42	Invalid block header length specification.
S43	BIN/EBCDIC mode not compatible with peripheral type.
S45	FIL option used with 7-track magnetic tape (7-track tapes permitted in DEVice mode only).
S46	Record format not supported by access method or peripheral type (i.e., APAM with undefined record format).
S47	KYL and KYP defined such that the key extends beyond record or overlay with DLC character.
S48	File extension attempted with organization different from that used during creation of the file.
S49	Access method requires use of MOV mode.
S4A	Number of buffers specified less than that required by access method, mode, or peripheral type.
S4B	Incompatible specification of REL, BKL, and/or MXL.
S4C	Erroneous justification of title on printer (i.e., title extends beyond printed line).
S50	Attempt to open a DCB using an operational label already associated with an open DCB.
S51	!ASSIGN or M:ASSIGN not provided for on operational label referenced in M:OPEN.
S52	Invalid DCB parameters specified.
S53	Designated volume is not available.
S54	Attempt to open a file in processing mode not allowed by access method (i.e., AIAM file open for read backward).
S55	Nonsharable volume is already in use.
S56	File nonsharable.
S57	User attempted to become queued for write access to a file that is already open to his task.
S58	File creation forbidden on this volume; it is write-protected by owner.
S59	Erroneous buffer address specified.
S5A	Sequence error in block numbers on magnetic tape.
S5B	File does not conform to XOS standards.
S5C	Creation of a null size temporary file.
S5E	Invalid DCB address specified.
S5F	Macro instruction not allowed for access method and Open mode being utilized.
S60	Unrecognizable disk header label.
S61	The number of files generated under account catalog exceeds limit specified at SYSGEN or volume preparation time.

Abort Code	Cause
S65	I/O error while processing disk file label record.
S6A	Insufficient memory space to satisfy M:ASSIGN or M:OPEN request.
S6B	Insufficient disk space to satisfy request for parallel mounting, or insufficient space to contain file on assigned volumes, or for status OLD file the last assigned volume is not last occupied volume.
S6C	Insufficient resources to satisfy mount request.
S6D	Attempt to create a temporary file with (STS,OLD).
S6E	Attempt to extend file when volume mounted is not the last volume of the file.
S6F	Volume assignment sequence does not correspond to sequence used during file creation.
S70	Password supplied by user is invalid.
S71	File deleted or file does not exist.
S72	Access not allowed.
S73	Nonexistent CFU entry at OPEN time.
S74	Nonexistent file at CLOSE time.
S75	Nonexistent CFU entry at CLOSE time.
S76	File extension attempted with null increment specified on SIZ parameter of !ASSIGN or M:ASSIGN.
S77	More than 23 extensions were required with !ASSIGN or M:ASSIGN increment specification provided.
S78	Required space not available within the parallel-mounted volume set or number of extents greater than 23.
S79	Unable to locate the desired file after an M:CVOL request processes a status OLD or MOD file.
S7A	File already exists on new volume after an M:CVOL request on status NEW file.
S7B	Nonexistent file version.
S7C	Disk allocation impossible.
S7D	Open in AIAM: BKL not a multiple of sector size, causing index-block algorithm to fail.

TAM MESSAGE MODE

ABNORMAL CONDITIONS (ABN)

ABN Code	Class	Cause	Origin	
			READ	WRITE
01	X2	End of message.	X	
03	X2	Suspension on terminal request.	X	X
04	X2	Initial POL/SEL I/O refused.	X	X
06	X2	Unexpected binary block received.	X	
1A	X3	Initial I/O in progress.	X	X
1B	X3	Line occupied by another component.	X	X
1C	X3	Operation not authorized for type of line used.	X	X
1D	X3	Component deactivated in list.	X	X
1E	X3	POL/SEL index greater than number of components.	X	X

ERROR CONDITIONS (ERR), RECOVERABLE

ERR Code	Cause	Origin	
		READ	WRITE
22	I/O purged.	X	X
23	Line in permanent error.	X	X
24	Terminal time-out (no response).	X	X
25	Incorrect response (BCC/ ACK, etc.).	X	X
26	Invalid response.	X	X
27	Synchronization error.	X	X
28	Transmission error.	X	X
29	Modem or carrier default.	X	X

ERROR CONDITIONS (ERR), IRRECOVERABLE

ERR Code	Cause	Origin	
		READ	WRITE
2A	Line locked (by operator).	X	X
2B	Modem not ready.	X	X

TAM CHARACTER MODE

ABNORMAL CONDITIONS (ABN)

ABN Code	Class	Cause	Origin		
			READ	READ (SUR)	WRITE
10	X2	Received ETX (during read).	X		
11	X2	Null byte count before completion.	X		
12	X2	ATTENTION1 received.	X	X	
13	X2	ATTENTION2 received.	X	X	
14	X2	HT received (tabulation if SYSGENed).	X		
1A	X3	Read survey refused (one already outstanding).		X	
1D	X3	List entry de-activated (by M:MDFLST).	X	X	X
1E	X3	List index greater than number of components.	X	X	X

ERROR CONDITIONS (ERR), RECOVERABLE

ERR Code	Cause	Origin		
		READ	READ (SUR)	WRITE
22	I/O purged.	X	X	X
24	Terminal (line) timed out (no response).	X		
26	Line not activated during open. User must issue M:DEVICE (opn).	X	X	X
27	Long space detected.	X		
29	Line in permanent error.	X	X	X

ERROR CONDITIONS (ERR), IRRECOVERABLE

ERR Code	Cause	Origin		
		READ	READ (SUR)	WRITE
2A	Line locked (by operator).	X	X	X
2B	Coupler in permanent error.	X	X	X

TAM OPEN/CLOSE

ABORT CONDITIONS

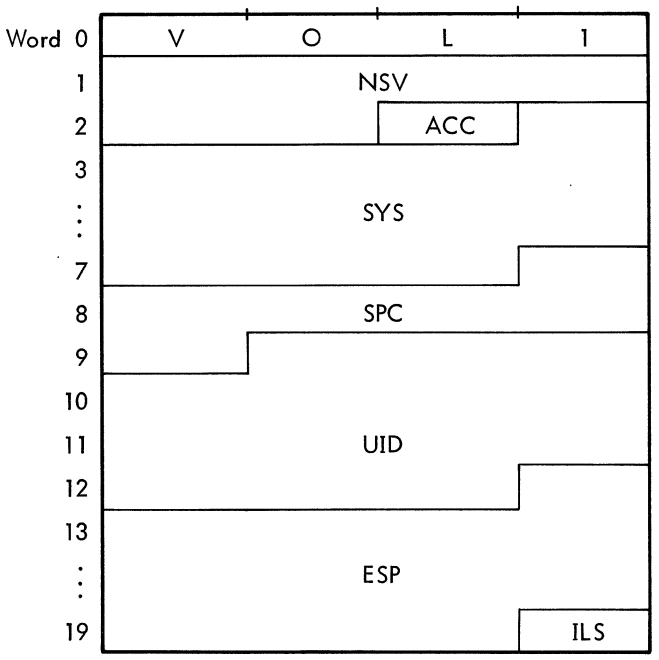
Abort Code	Cause
S30	The lines of a group belong to different modes.
S3A	Erroneous address of explicit list.
S3B	Total number of components attached to DCB greater than 254 in message mode.
S3C	Use of component (message mode) does not conform to the list entry.
S3D	A component in the list does not belong to the assigned line.
S3E	A component has been named twice in a direct list.
S3F	No line could be activated (character mode).
S41	Access method not authorized.
S43	Mode prohibited (binary or ANSCII).
S50	One of the lines is being processed by another DCB.
S51	Unassigned operational label.
S55	The operational label is being processed by another DCB.
S5E	Invalid DCB address.
S6A	Too much memory space requested.

ABORT CONDITIONS

Abort Code	Cause	Origin				
		READ	WRITE	DEVICE	CHECK	MDFLST
R51	Null transfer length.		X			
R54	Transfer length exceeds maximum length for component.	X	X			
R59	Erroneous address (buffer, event pointer, byte count, or index).	X	X	X	X	X
R5A	No I/O in progress				X	
R5B	Too many I/O's in progress (no. of current I/O's greater than SIM).	X	X	X		
R5C	DCB closed.	X	X	X	X	X
R5E	DCB invalid.	X	X	X	X	X
R63	Operation prohibited			X		X
R64	List unknown in DCB.					X
R67	Mode prohibited (binary or ANSCII).			X		
R68	All the lines in the group are in permanent error.	X	X			
R69	Usage of dissimilar pointers.				X	

APPENDIX C. STANDARD VOLUME AND FILE LABELS

VOLUME LABELS ON MAGNETIC TAPE



where

VOL1 is the label identifier; it contains the four characters VOL1.

NSV is the volume serial number; it contains one to six characters assigned during volume preparation (PREP Utility Program), left-justified and padded with nulls.

ACC is the access control; it contains a space character meaning "unlimited access" (i.e., no one is restricted at the volume level).

SYS is unused. it contains 20 space characters. Reserved for future system use.

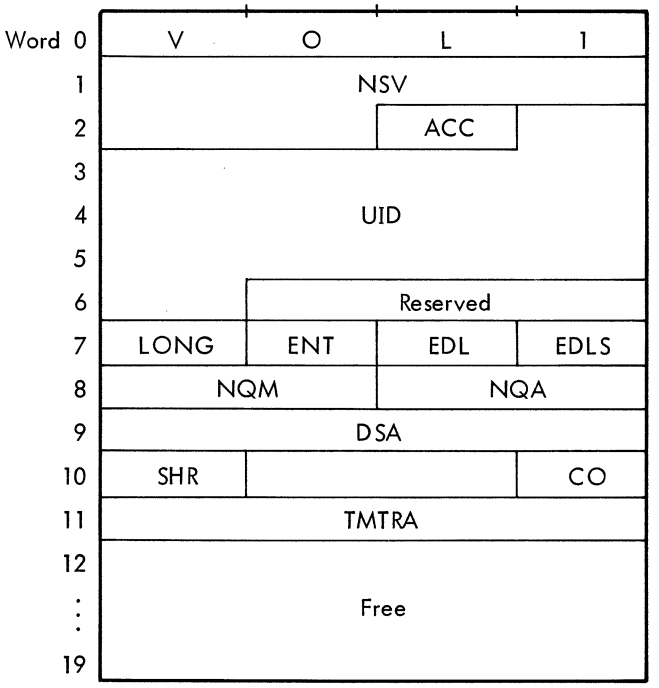
SPC is unused; it contains six space characters. Reserved for future system use.

UID is the owner identification; it contains one to four characters representing the account number, left-justified and trailed with space characters to a total of 14 characters.

ESP is unused; it contains 28 space characters. Reserved for future system use.

ILS is the standard label identifier; it contains the character 1 for all labels created by XOS. The 1 signifies that the label adheres to ANS standards.

VOLUME LABELS ON DISK



where

VOL1 is the label identifier; it contains the four characters VOL1.

NSV is the volume serial number; it contains one to six characters assigned during volume preparation (PREP Utility Program), left-justified and padded with nulls.

ACC is the access control; it contains a space character meaning "unlimited access" (i.e., no one is restricted at the volume level).

UID is the owner identification; it contains one to four characters representing the account number, left-justified and trailed with space characters to a total of 14 characters.

LONG specifies the length of an entry in the catalog, in number of bytes.

ENT specifies the number of entries being used in the primary part of the catalog.

EDL specifies the number of free entries in the primary part of the catalog.

EDLS specifies the size of the secondary part of the catalog, in multiples of the primary part.

NQM is the maximum number of quanta to be allocated for the volume (a quanta is 8192 bytes).

NQA is the number of quanta allocated.

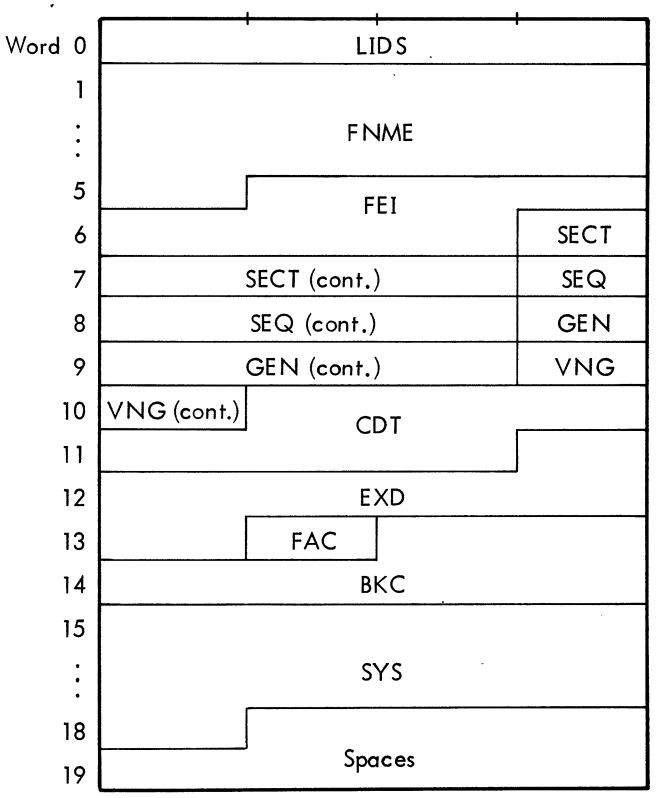
DSA is the disk address of the beginning of the volume catalog.

SHR is the sharing authorization code (0 = non-shareable; 1 = shareable).

CO is not used; it contains a zero. Reserved for future system use.

TMTRA is the disk address of the TMTRA table used for space allocation on this volume.

FILE LABELS ON MAGNETIC TAPE: HDR1, EOF1, EOV1



where

LIDS is the label identifier; it contains the characters HDR1, EOF1, or EOV1.

FNME is the file name; it contains 1 to 17 characters, left-justified and padded with space characters.

FEI is the file identifier; it contains the same volume serial numbers as the first (or only) volume of the group of files associated at the time the file was created. Represented in character format.

SECT is the file section number; it contains a sequence number (0001-9999, in character format) indicating the order number of this volume within the volumes containing this file. It contains 0001 for a monovolume file or the first part of a multivolume file.

SEQ is the file sequence number; it contains the sequence number (0001-9999, in character format) of the file within a volume or multivolume. At the point within a volume that a file terminates and another begins, this field is incremented by one.

GEN is the absolute generation number.

VNG is the version number of the generation.

CDT is the creation date; it contains the date that the file was created in the form

byyddd

where

b is a space.

yy is the last two digits of the year.

ddd is the day of the year (001-366).

EXD is the expiration date; it contains the date (in the same format as CDT) that the file is eligible to be overwritten. The file is regarded as expired when the current data is not earlier than the expiration date.

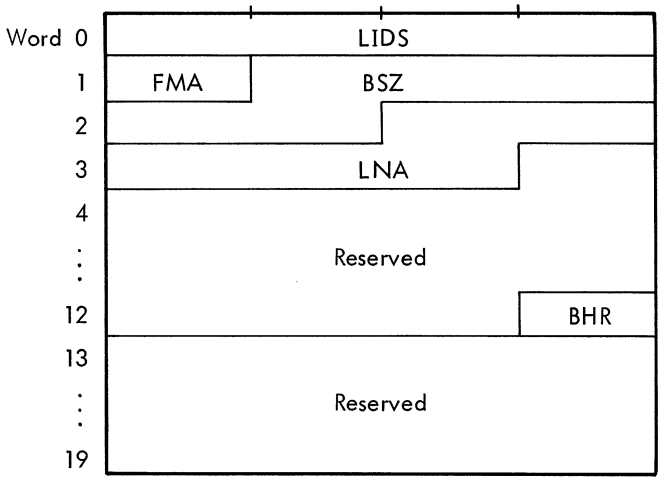
FAC is the accessibility; it contains a character code denoting the restrictions on who may have access to this file. The default character is B (see Chapter 6, Table 6-4).

BKC is the block count; it contains the characters 000000 for HDR1 type labels. It contains the count of the data blocks (not including labels and tape marks) since the preceding header label group for EOV1 and EOF1 type labels.

SYS is the system code; it contains 1 to 13 characters identifying the system that recorded this file, left-justified and padded with space characters.

Spaces is not used; it contains space characters. Reserved for future system use.

FILE LABELS ON MAGNETIC TAPE: HDR2, EOF2, EOV2



where

LIDS is the label identifier; it contains the characters HDR2, EOF2, or EOV2. This label is always written for files created by XOS.

FMA is the record format; it contains a single character:

F = fixed.

V = variable

U = undefined.

BSZ is the block length; it contains a number (in character format) specifying the maximum number of characters per block.

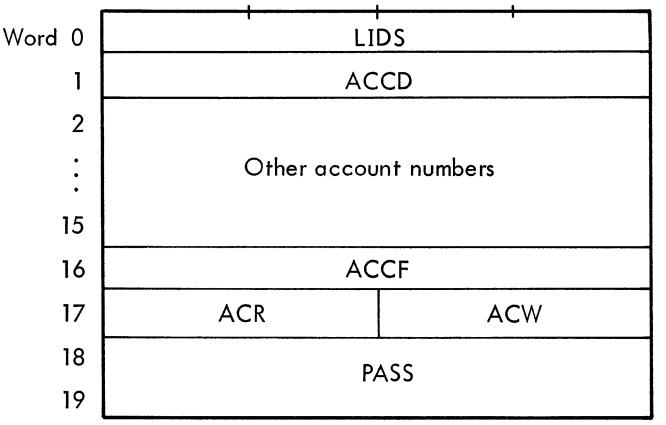
LNA is the record length; it contains a number (in character format) specifying either the record length if the format is F, or maximum record length if the format is V or U.

Reserved is not used; it contains space characters. Reserved for future system use.

BHR is the buffer offset; it contains a number (00-99, in character format) specifying the length of the data block header field.

Reserved is not used; it contains space characters. Reserved for future system use.

FILE LABELS ON MAGNETIC TAPE: HDR3, EOF3, EOV3



where

LIDS is the label identifier; it contains the characters HDR3, EOF3, or EOV3. This label is written only if special protection is assigned to the file.

ACCD is the first authorized account number; it contains an account number and may be followed by a list of other account numbers. Accounts authorized for reading are followed by those authorized for writing. The number of accounts authorized for reading and the number of accounts authorized for writing are recorded in the fields ACR and ACW, respectively (see below).

ACCF is the last authorized account number; it may contain an authorized account number. If this field or any preceding authorized account number fields are unused, they are filled with space characters.

ACR specifies the number of accounts authorized for reading (in character format).

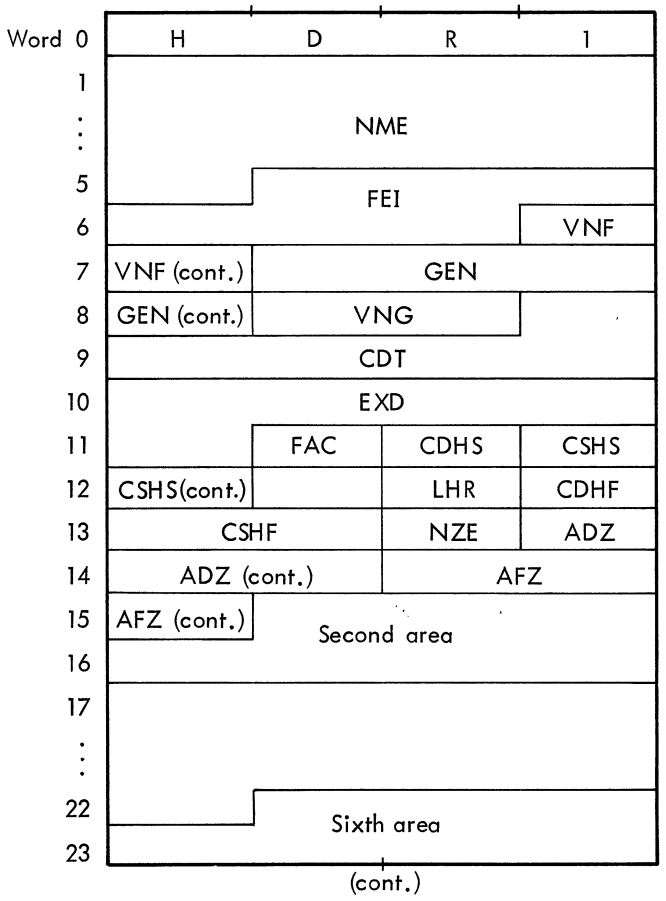
ACW specifies the number of accounts authorized for writing (in character format).

PASS is the password; it contains the password required for access to the file.

FILE LABELS ON DISK

ENTRIES IN THE PRIMARY PORTION OF THE CATALOG

NONCATALOGED FILES



where

HDR1 is the label identifier; it contains the four characters HDR1.

NME is the file name; it contains one to 17 characters, left-justified and padded with space characters.

FEI is the file identifier; it contains the volume number of the first (or only) volume associated with the group of files at the time the file was created. Represented in character format.

VNF is the volume number in the file; it contains a sequence number (00-99, in character format) indicating the order number of this volume within the volumes containing this file. It contains 00 for a monovolume file or for the first part of a multivolume file.

GEN is the absolute generation number.

VNG is the version number of the generation.

CDT is the creation date; it contains the date that the file was created in the form

yyddd

where

yy is the last two digits of the year.

ddd is the day of the year (001-366).

EXD is the expiration date; it contains the date (in the same format as CDT) that the file is eligible to be overwritten. The file is regarded as expired when the current date is not earlier than the expiration date.

FAC is the accessibility; it contains a character code denoting the restrictions on who may have access to this file. The default character is B (see Chapter 6, Table 6-4).

CDHS is not used; it contains a zero. Reserved for future system use.

LHR has two fields of information:

1. The leftmost bit specifies whether or not more HDR1 labels exist in the file (0 = yes; 1 = no).
2. The rightmost seven bits specify the total number of areas used in the file, including those described in the EHDR extensions.

CDHF is not used; it contains a zero. Reserved for future system use.

CSHF is not used; it contains a zero. Reserved for future system use.

NZE has two meanings:

1. For private volumes. Unconditionally set to 1; this has no particular meaning.
2. For system volumes. Contains the index to the DCT (Device Control Table).

ADZ is the sector address of the beginning of the first area.

AFZ is the sector address of the end of the first area.

Note: Space has been provided in this label to describe from one to six areas, the number of areas actually used being dependent on the size of the file and the size of the areas. NZE, ADZ, and AFZ describe the first area. For each of the remaining five areas, there are fields which are identical in function and format to NZE, ADZ, and AFZ. If the file becomes so large that the six areas do not provide sufficient space, then label extensions are used to describe additional areas.

(cont.)

Word 24	H	D	R	2
25	ORG	BSZ		FRMA
26	LNA		KYP	
27	LVL	ALB		
28	CNC	SIZE		LIN
29	LIN (cont.)	TAB		KYL
30		NBD		
31	INC		PEXT	

where

HDR2 is the label identifier; it contains the four characters HDR2.

ORG is the organization type of the file; it contains a single character:

C = sequential.

I = indexed.

P = partitioned.

D = direct.

BSZ is the block size, in bytes.

FRMA is the record format; it contains a single character:

F = fixed.

V = variable.

U = undefined.

LNA is the record length; it contains the actual record length for fixed format records, or the maximum length for variable and undefined format records.

KYP is the position of the key relative to the beginning of the file. Applicable to indexed organization.

LVL is the number of index levels. Applicable to indexed organization.

ALB is the address of the last data block; it contains the sector number with respect to the beginning of the file.

CNC has two fields of information:

1. The leftmost bit specifies whether or not a delete control character exists in the records (0 = no; 1 = yes).
2. The rightmost four bits contain the byte length of the block header (always 4).

SIZE is the size of the file in quanta (8192 bytes).

LIN is the index length, in number of bytes (in character format). Applicable to indexed and partitioned files.

TAB specifies the rate of access to the overflow blocks.

KYL is the key length, in number of bytes. Applicable to indexed and partitioned files.

NBD is the number of overflow blocks.

INC is the disk increment; it contains the number of quanta (8192 bytes) to be allocated each time the file is expanded.

PEXT is the pointer to the first extension; it contains the entry number with respect to the beginning of the catalog.

CATALOGED FILES

Word 0	H	D	R	1
1	NME			
5	FEI			
6				VNF
7	VNF (cont.)	GEN		
8	GEN (cont)	VNG		
9	CDT			
10	EXD			
11		FAC	CDHS	CSHS
12	CSHS(cont.)		LHR	CDHF
13	CSHF		NBECTG	PTR
14	Free			
15	ECTG		Free	
16	Free			
23				

(cont.)

where

HDR1 is the label identifier; it contains the four characters HDR1.

NME is the file name; it contains one to 17 characters, left-justified and padded with space characters.

FEI is the file identifier; it contains the same volume number as the first (or only) volume associated with the group of files at the time the file was created. Represented in character format.

VNF is the volume number in the file; it contains a sequence number (00-99, in character format) indicating the order number of this volume within the volumes containing this file. It contains 00 for a monovolume file or for the first part of a multivolume file.

GEN is the absolute generation number.

VNG is the version number of the generation.

CDT is the creation date; it contains the date that the file was created in the form

yyddd

where

yy is the last two digits of the year.

ddd is the day of the year (001-366).

EXD is the expiration date; it contains the date (in the same format as CDT) that the file is eligible to be overwritten. The file is regarded as expired when the current date is not earlier then the expiration date.

FAC is the accessibility; it contains a character code denoting the restrictions on who may have access to this field. The default character is Y (see Chapter 6, Table 6-4).

CDHS is not used; it contains a zero. Reserved for future system use.

LHR is an additional HDR1 label indicator. Only the leftmost bit of this field has meaning:

0 = more HDR1 labels exist in the file.

1 = the last HDR1 label in the file.

CDHF is not used; it contains a zero. Reserved for future system use.

NBECTG is the number of ECTG entries.

PTR is the pointer to the extension entry; it contains a number relative to the beginning of the catalog that points to an EHDR or HDR3 label, if there is any.

ECTG is the pointer to the ECTG extension entry; it contains a number relative to the beginning of the catalog.

(cont.)

Word 24	H	D	R	2
25	ORG	BSZ		FRMA
26	LNA		KYP	
27	ALB			
28	CNC	SIZE		LIN
29	LIN (cont.)	TAB		KYL
30		NBD		
31	INC		PEXT	

where

HDR2 is the label identifier; it contains the four characters HDR2.

ORG is the organization type of the file; it contains a single character:

C = sequential.

I = indexed.

P = partitioned.

D = direct.

BSZ is the block size, in bytes.

FRMA is the record format; it contains a single character:

F = fixed.

V = variable.

U = undefined.

LNA is the record length; it contains the actual record length for fixed format records, or the maximum length for variable and undefined format records.

KYP is the position of the key relative to the beginning of the file. Applicable to indexed organization.

ALB is the address of the last data block; it contains the sector number with respect to the beginning of the file.

CNC has two fields of information:

1. The leftmost bit specifies whether or not a delete control character exists in the records (0 = no; 1 = yes).

2. The rightmost four bits contain the byte length of the block header (always 4).

SIZE is the size of the file in quanta.

LIN is the index length, in number of bytes. Applicable to indexed and partitioned files.

TAB specifies the rate of access to the overflow blocks.

KYL is the key length, in number of bytes. Applicable to indexed and partitioned files.

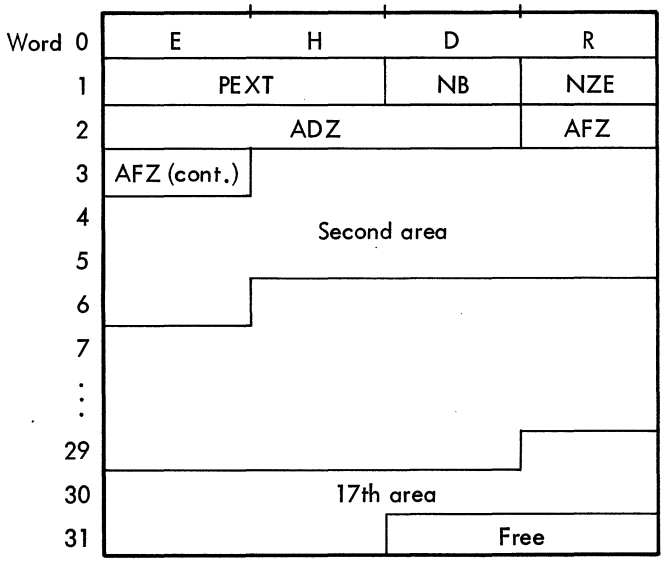
NBD is the number of overflow blocks.

INC is the disk increment; it contains the number of quanta (8192 bytes) to be allocated each time the file is expanded.

PEXT is the pointer to the first extension; it contains the entry number in respect to the beginning of the catalog.

ENTRIES IN THE SECONDARY PORTION OF THE CATALOG

EXTENSION OF THE FILE MAP: EHDR



where

EHDR is the label identifier; it contains the four characters EHDR.

PEXT is the pointer to the next extension; it contains the entry number in respect to the beginning of the catalog.

NB is the number of used areas stored in this extension.

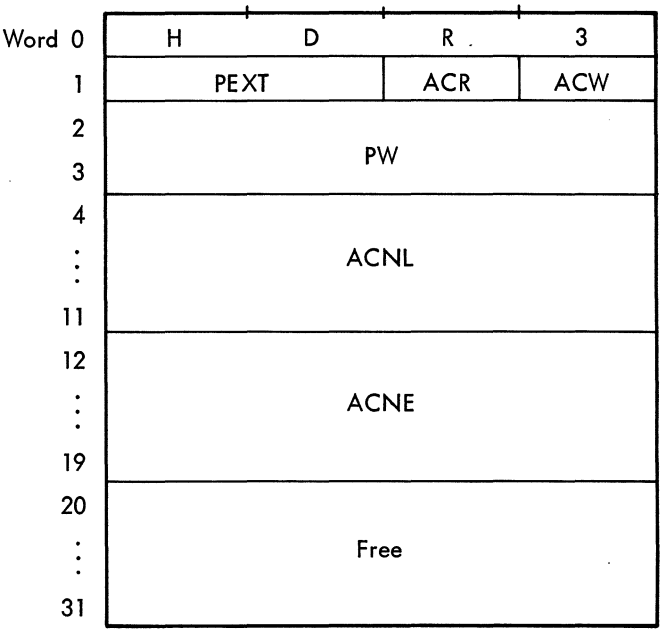
NZE is the number of the first block in the area.

ADZ is the sector address of the beginning of the area.

AFZ is the sector address of the end of the area.

Note: Space has been provided in this label to describe from 1 to 17 areas, the number of areas actually used being dependent on the size of the file and the size of the areas. NZE, ADZ, and AFZ describe the first area. For each of the remaining 16 areas, there are fields which are identical in function and format to NZE, ADZ, and AFZ.

FILE PROTECTION: HDR3



where

HDR3 is the label identifier; it contains the four characters HDR3.

PEXT is the pointer to the next extension; it contains the entry number in respect to the beginning of the catalog of the next label extension of the file.

ACR specifies the number of accounts authorized for reading (the maximum number is eight).

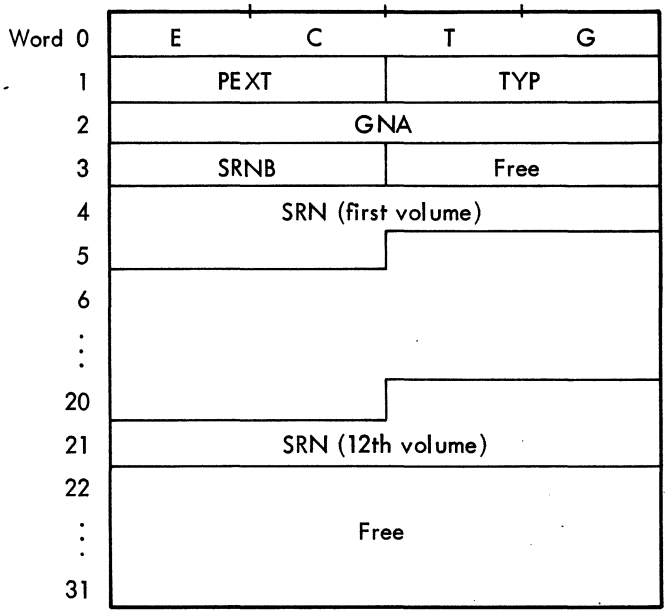
ACW specifies the number of accounts authorized for writing (the maximum number is eight).

PW is the password; it contains the password required for access to the file.

ACNL specifies the account numbers authorized for reading. It contains a list of from one to eight account numbers that are authorized to read the file.

ACNE specifies the account numbers authorized for writing. It contains a list of from one to eight account numbers that are authorized to write in the file.

EXTENSION OF THE CATALOG: ECTG



where

ECTG is the label identifier; it contains the four characters ECTG.

PEXT is the pointer to the next extension entry; it contains the entry number in respect to the beginning of the catalog.

TYP is the device type; it contains the type number of the volumes for which the serial numbers are specified.

Device Type	Number
RAD (7212)	0
RAD (7204)	3
RAD (7232)	4
Disk Pack (7242)	5
MT (9-Track)	11
MT (7-Track)	12

GNA is the absolute generation number.

SRNB specifies the number of volume serial numbers cataloged.

SRN is the volume serial number. A maximum of 12 volume serial numbers may be specified.

APPENDIX D. USER LABEL PROCESSING

User labels are optional labels created specifically by the user for his own purposes. Volumes and files that are to be recognized as labeled by the system must always have standard system labels, these being created by system routines. User labels are created as additional labels through user routines.

It is important to note that the capability to create user labels is only available for magnetic tape.

TERMINOLOGY

SVL (Standard Volume Label Group). SVL is comprised of the VOL1 label only.

SHL (Standard Header Label Group). SHL is comprised of HDR1, HDR2, and optionally, HDR3 labels.

STL (Standard Trailer Label Group). STL is comprised of EOF1, EOF2, and optionally, EOF3 labels for end-of-file; or EOV1, EOV2, and optionally, EOV3 labels for end-of-volume.

The standard labels are discussed in detail in Appendix C, Standard Volume and File Labels.

UVL (User Volume Label Group). UVL is comprised of up to nine 80-byte records. The first three bytes of each record contains the characters UVL while the fourth byte contains a digit that is consecutive within the group starting at 1.

UHL (User Header Label Group). UHL is comprised of any number of 80-byte records. The first three bytes of each record contain the characters UHL.

UTL (User Trailer Label Group). UTL is comprised of any number of 80-byte records. The first three bytes of each record contain the characters UTL.

All remaining bytes of user labels are free for the user to use as he wishes.

Figure D-1 provides an example of a tape volume with user labels.

GENERAL INTRODUCTION

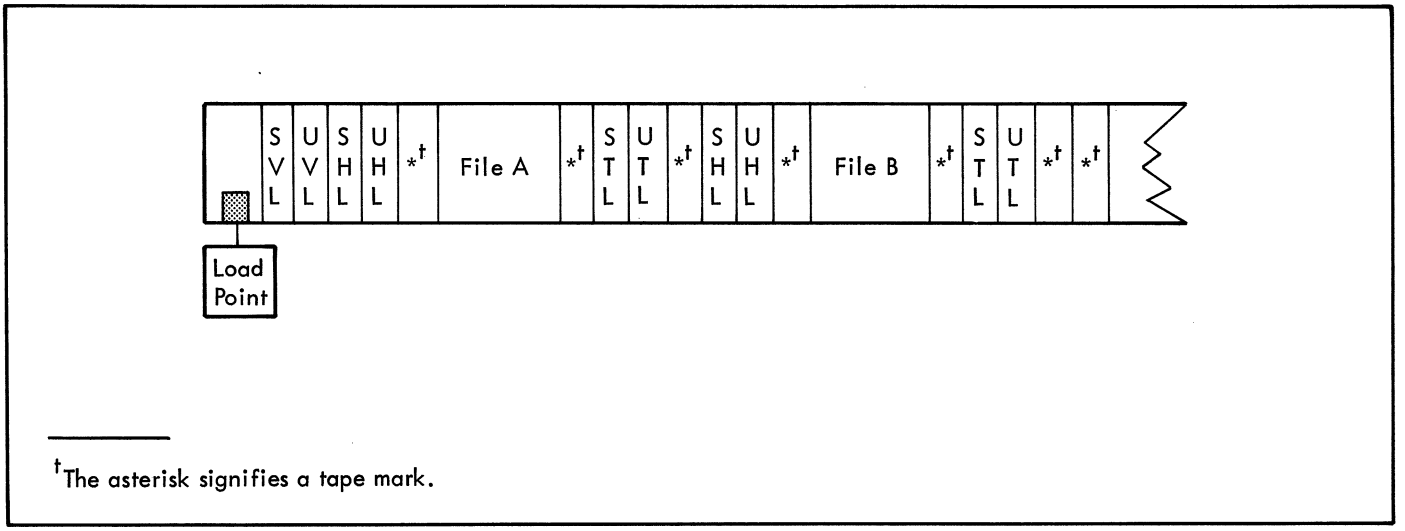
The processing of user labels is recognized and facilitated by the system. However, user labels are considered abnormal and the user routine that processes them is considered an abnormal routine. In fact, the user routine is entered with the aid of the ABN (abnormal) option associated with the DCB, as will be discussed later.

In order to process user defined labels, two conditions must be satisfied:

1. Preconditions must have been established. The location of the user abnormal routine, the abnormal class denoting the kind of user label processing desired, and the location of a label processing area must be specified.

2. The abnormal condition associated with the specified abnormal class must occur.

When both of the above conditions have been met, the system will give control to the user's routine, which has been written to process user labels as well as other selected abnormal conditions. The requirements for meeting the conditions is discussed below.



†The asterisk signifies a tape mark.

Figure D-1. Example of Tape Volume with User Labels

ESTABLISHING THE USER LABEL PROCESSING PRECONDITIONS

The system allows the user to specify which user label types he would like to process and the address of the user's abnormal processing routine through the use of the ABN option associated with the DCB. The ABN option has the format

(ABN, address [,class-code, . . .])

where

address specifies the address of a routine which processes any one (or more) of six possible "abnormal" conditions.

class-code is one or more codes that specify which of the six classes of abnormal conditions are to be handled by the routine at the given address. Only three of the six classes apply to user labels. They are

Class-Code	Kind of User Label Processing
X4	UHL at beginning-of-file.
X5	UTL at end-of-file.
X6	UVL at beginning-of-volume.

The specification of these classes and the address of the processing routine can be changed at any time. (However, while the file is open, they can only be changed via M:SETDCB.)

The address of the memory area into which the label is read (input) or from which the label is written (output) is specified using the TLB option associated with the DCB and must be specified before an M:OPEN is executed. The TLB option has the format

(TLB, address)

OCCURRENCE OF AN ABNORMAL LABEL CONDITION

An input abnormal label condition occurs as follows:

1. Beginning-of-volume label condition occurs when
   a. AVR (Automatic Volume Recognition) has occurred and either (1) the first OPEN occurs on the volume, or (2) the volume-beginning is being processed due to M:CVOL.
   b. The physical record following the SVL (1) is 80 bytes in length and (2) contains the four characters UVL1 in the four byte positions of the record.
2. Beginning-of-file label condition occurs when
   a. An M:OPEN or M:CVOL has occurred and the SHL has been processed.
   b. The physical record following the SHL (1) is 80 bytes in length and (2) contains the three characters UHL in the first three byte positions of the record.
3. End-of-file label condition occurs when
   a. An M:CLOSE or M:CVOL has occurred and the STL has been processed.
   b. The physical record following the STL (1) is 80 bytes in length and (2) contains the three characters UTL in the first three byte positions of the record.

An output abnormal label condition occurs as follows:

1. Beginning-of-volume label condition occurs when AVR has occurred and there are no files on the tape at the time the first M:OPEN or M:CVOL is being processed (i.e., the record following the SVL is a tape mark).
2. Beginning-of-file label condition occurs when an M:OPEN or M:CVOL has occurred and the SHL has been processed.
3. End-of-file label condition occurs when an M:CLOSE or M:CVOL has occurred and the STL has been processed.

SYSTEM RESPONSE TO AN ABNORMAL LABEL CONDITION

Whenever an abnormal label condition occurs, the system always checks to see if the preconditions have been established for the particular user label involved. If they have, the system passes control to the user's user-label processing routine. If the preconditions have not been established, the system handling of the abnormal label condition is as follows:

1. For input
   a. UVL. The UVL is bypassed; however, the sequencing is verified.
   b. UHL. The UHL is bypassed.
   c. UTL. The UTL is bypassed.
2. For output
   a. UVL. The abnormal label condition is ignored.
   b. UHL. The abnormal label condition is ignored.
   c. UTL. The abnormal label condition is ignored.

USER LABEL PROCESSING ROUTINE

The user can write only one routine for all abnormal conditions because only one address for such a routine can be

specified with the ABN option. Therefore, if more than one class of abnormality is preconditioned (this includes classes X1 through X6), the user's abnormal routine must be able to distinguish between and among the classes. A code is provided left-justified in register 5 which enables the user to program for this. For abnormal label conditions, the codes are:

Kind of User Label	Class	Hex Value of Code in Register 5
UHL	X4	10
UTL	X5	12
UVL	X6	11

In addition, the return address to the system routine is provided right-justified in register 5 and the DCB address is provided right-justified in register 6.

If the user label processing routine is entered during an input operation, the label that caused the abnormal condition to occur is available to the routine in the location specified by the TLB option. The tape is positioned following that label.

If the user label processing routine is entered during an output operation, the location specified by the TLB option is available for label creation. The tape is positioned following the label that has just been written.

The user labels can only be processed one 80-byte record at a time. The programming considerations for processing further label records of the same group are discussed below.

1. Input

   After processing the first user label record, the user routine may require that the next user label record in the group be processed. If so, the routine merely sets register 6 to the value zero before executing an M:RETURN. This is called a special return. If another label record in the group does exist, the user routine will be entered just as described for initial entry and that record can thereby be processed.

   However, if the user label group has no more labels, the system will continue processing with no adverse effects.

   At some point in this sequence, the user routine may return control to the system without setting register 6 to zero. The system will bypass the remaining labels in the group, if any, as if the abnormal label condition had not been preconditioned. This is called a normal return.

2. Output

   After creating a label the user routine can cause that label to be written to tape by setting register 6 to the value zero and then executing an M:RETURN. This is called a special return. After having written the label, the system will enter the user routine as described for initial entry. This provides the opportunity to process further labels in the group.

   When the user routine returns control to the system without setting register 6 to zero, the system will continue processing as if the abnormal label condition had not been preconditioned. This is called a normal return.

The user label processing routine must return control to the system via procedures such as M:RETURN or M:ERR (see Chapter 8). The choice of which procedure to use depends upon the logic of the user's routine.

APPENDIX E. DATA CONTROL BLOCK (DCB)

The DCB is a 19-word table that groups the parameters defining the logical structure of a file and its processing mode. This information is used to interpret the I/O requests given by the user.

A diagram of the DCB is shown in Figure E-1. Each of the fields is described, in alphabetical order, in Table E-1. A special discussion of the DCB for TAM (Telecommunications Access Method) is given at the end of this appendix.

	1 – 7	8 – 15	16 – 23	24 – 31
Word 0	DCBCLC	DCBDEB		
1	DCBORG	FCI, FCD, HLD, VFC, MOV, NRT, SIG, SEQ, NBC, DLC	MOD	DCBFRM
2	DCBAM	DCBMXL		
3	DCBABC	DCBABN		
4	DCBSIM	DCBERR		
5	DCBSYD			
6	DCBOPL			
7	DCBBCT			
8	DCBKYP		DCBREL	
9	DCBFUN	DCBCKP		
10	DCBBHR	DCBKYL	DCBBKL	
11	DCBNBF	DCBBFA		
12	DCBDTA	DCBTLB		
13	DCBHDC	DCBHDR		
14	DCBCNT	DCBLIN	DCBSPC	DCBSPL
15†	DCBSID			
15†	DCBTAB1	DCBTAB2	DCBTAB3	DCBTAB4
16	DCBTAB5	DCBTAB6	DCBTAB7	DCBTAB8
17	DCBTAB9	DCBTAB10	DCBTAB11	DCBTAB12
18	DCBTAB13	DCBTAB14	DCBTAB15	DCBTAB16

†Note that word 15 has two possible configurations.

Figure E-1. Data Control Block

Table E-1. Standard DCB Parameter Fields

Field Name	Length	Description
DCBABC	1 byte	Specifies how abnormal conditions are handled. Each of the last six bits of this byte have been assigned to represent one of the six possible abnormal conditions. If an abnormal condition occurs and its corresponding bit is set to 1, then the abnormal condition is processed by the user routine addressed in DCBABN. If an abnormal condition occurs and its corresponding bit is set to 0, then the job is aborted. The bit assignments are: Bit 7 – End of file exit (X1). Bit 6 – End of volume exit (X2). Bit 5 – Abnormal process exit (X3). Bit 4 – User label at beginning of file (X4). Bit 3 – User label at end of file or volume (X5). Bit 2 – User label at beginning of volume (X6). Bit 1 – Not used. Bit 0 – Not used.
DCBABN	3 bytes	Specifies the address of the user routine which processes abnormal I/O conditions. There are six possible abnormal conditions and those which are to be processed by this routine are specified in DCBABC.
DCBAM	1 byte	Specifies the access method. 0 – ASAM (sequential assisted). 1 – AIAM (indexed assisted). 2 – APAM (partitioned assisted). 3 – VSAM (virtual sequential). 4 – VDAM (virtual direct). 5 – BDAM (real direct). 6 – TAM (telecommunications).
DCBBCT	1 word	Contains the word address of the chain of opened DCBs.
DCBBFA	3 bytes	Contains the virtual word address of the first buffer reserved in the program. (If there is more than one buffer, the buffers must be contiguous.)
DCBBHR	1 byte	Contains the number of bytes in the block header and is applicable only to V and F record formats. The default value is four.
DCBBIT	(Not located within the DCB.)	Contains a value given by ths system which is a relative pointer to the word in the DCB that contains all the one-bit fields. (That word is Word 1 at the present time, but has been parameterized through DCBBIT to facilitate making changes to the system at a later time.)

Table E-1. Standard DCB Parameter Fields (cont.)

Field Name	Length	Description
DCBBKL	2 bytes	Specifies block size in bytes. It corresponds to the maximum transfer length for the assisted access methods. The default value is 1024 for disks and magnetic tapes, 133 for the printer, and 80 for punched cards.
DCBCKP	3 bytes	Is reserved for future system use.
DCBCLC	1 byte	Contains a code depending upon and given by the access method, controlling alternate, method-dependent branches to common service modules.
DCBCNT	1 byte	Indicates when a page number is to be printed (for print files). It is a column number and must be less than 130.
DCBDEB	3 bytes	Contains the virtual word address of the DEB (Device Event Block).
DCBDLC	1 bit	Indicates whether or not each record contains a delete control character. 0 – no delete control character. 1 – delete control character.
DCBDTA	1 byte	Specifies the number of the column in which printing or punching begins.
DCBERR	3 bytes	Specifies the word address of the user error exit procedure. If this address is omitted, the job is aborted when an error occurs.
DCBFCD	1 bit	Indicates whether the DCB is closed or opened. 0 – closed 1 – opened
DCBFCI	1 bit	Indicates, when the bit is set to 1, that a temporary close has taken place on this DCB with MTN. If the bit is set to 0, then this is not the case.
DCBFRM	1 byte	Specifies the record format. 0 – Fixed (default) 1 – Variable 2 – Undefined
DCBFUN	1 byte	Indicates the processing mode. 0 – Forward reading (I) 1 – Writing (O) 2 – Backward reading (B) 3 – Updating (U) 4 – Scratch (S)
DCBHDC	1 byte	Specifies the column number on which header centering is done (for print files).
DCBHDR	3 bytes	Contains the virtual word address of the title to be printed at the top of each page (for print files).

Table E-1. Standard DCB Parameter Fields (cont.)

Field Name	Length	Description
DCBHLD	1 bit	Indicates, when the bit is set to 1, that a temporary close has taken place on this DCB with HLD. If the bit is set to 0, then this is not the case.
DCBKYL	1 byte	Contains the length of the key in bytes.
DCBKYP	2 bytes	Specifies the key position in the record (number of bytes between the beginning of the record and the beginning of the key). The maximum value is equal to the maximum size of the record less the key size. The default value is 0.
DCBLIN	1 byte	Specifies the number of lines per printed page (for print files).
DCBMOD	3 bits	Indicates the data format. Note that some codes are used twice. This does not create a conflict because the codes have different meanings for different devices. 0 – EBCDIC (for anything except 7-track tape). 0 – BCD (for 7-track tape). 1 – Binary (for anything except 7-track tape). 1 – Packed (for 7-track tape. 2 – Unpacked (for 7-track tape). 3 – ASCII (for transmission devices).
DCBMOV	1 bit	0 – I/O is performed without moving the record into a user's word area (LOC mode). 1 – I/O is performed by moving the record into a work area specified by the user program (MOV mode).
DCBMXL	3 bytes	Indicates the maximum length of memory-peripheral transfer in bytes. This length can be greater than the length specified by DCBBKL for the nonassisted methods. The default value is equal to the block length.
DCBNBC	1 bit	Specifies whether or not there is a block count (applicable on magnetic tape). 0 – block count. 1 – no block count. ASAM creates and uses block counts. This bit is used to flag ASAM that a block count does not exist (as might be the case in files which are created by other access methods).
DCBNBF	1 byte	Contains the number of buffers used in the assisted access methods. The default value is 2.
DCBNRT	1 bit	Indicates what is to be done in case of an error. 0 – special handling. 1 – standard procedure to retry.
DCBOPL	1 word	Specifies the operational label of the DCB. This label is composed of a maximum of four alphanumeric characters, left-justified and padded with space characters.

Table E-1. Standard DCB Parameter Fields (cont.)

Field Name	Length	Description
DCBORG	1 byte	Indicates the type of file organization. 0 – sequential (default). 1 – indexed. 2 – direct. 3 – partitioned.
DCBREL	2 bytes	Contains the record length for V and U format records.
DCBSEQ	1 bit	Indicates whether or not a page count or card count has been requested. If it has been requested, the system will print page numbers or punch sequence numbers. 0 – no page count or card count. 1 – page count or card count.
DCBSID	1 word	Contains a card count identifier (four EBCDIC characters) to be punched in columns 73-76.
DCBSIG	1 bit	Indicates whether or not a card count identifier is present in Word 15 of the DCB. 0 – no card count identifier is present. 1 – card count identifier is present.
DCBSIM	1 byte	Indicates the maximum number of simultaneous I/O operations allowed for the file. The default value is the same as the number of buffers (see DCBNBF).
DCBSPC	1 byte	Indicates the type of spacing to be done for print files. 1 – single spacing (default). 2 – double spacing. 3 – triple spacing. etc.
DCBSPL	1 byte	Specifies the number of the first printed line (for print files).
DCBSYD	1 word	Contains four EBCDIC characters which are the SYSID. The first character is a letter that identifies the class under which the job is executed, the class being specified on the !JOB card. The last three characters contain the job number (000-999). The job number is set to 000 when the system is booted and is incremented by 1 for each new job.
DCBTAB1 to DCBTAB16	1 byte each (4 words)	Contain column numbers for tab stops (for print files).
DCBTLB	3 bytes	Specifies the virtual word address of the memory area into which a user label may be read or from which a user label may be written.
DCBVFC	1 bit	Indicates whether or not a vertical spacing control character exists in the record heading. This control character is only applicable for print files and indicates how much space there should be between lines. If no vertical spacing control character exists, the system will provide for single spacing by default. 0 – no vertical spacing character in the record body. 1 – vertical spacing character in the record body.

THE DCB USED BY TAM (TRANSMISSION ACCESS METHOD)

The DCB for TAM contains many of the standard DCB parameters and four special parameters peculiar to TAM:

DCBLST1

DCBLST2

DCBLST3

DCBLST4

A number of the fields of the standard DCB are not used for TAM but the unused fields can not be removed because the TAM DCB would no longer be compatible with the standard DCB macros.

One DCB is created by the user for a group of lines of the same characteristics.

A diagram of the TAM DCB is shown in Figure E-2. Only the characteristics peculiar to the TAM DCB are listed in Tables E-2 and E-3.

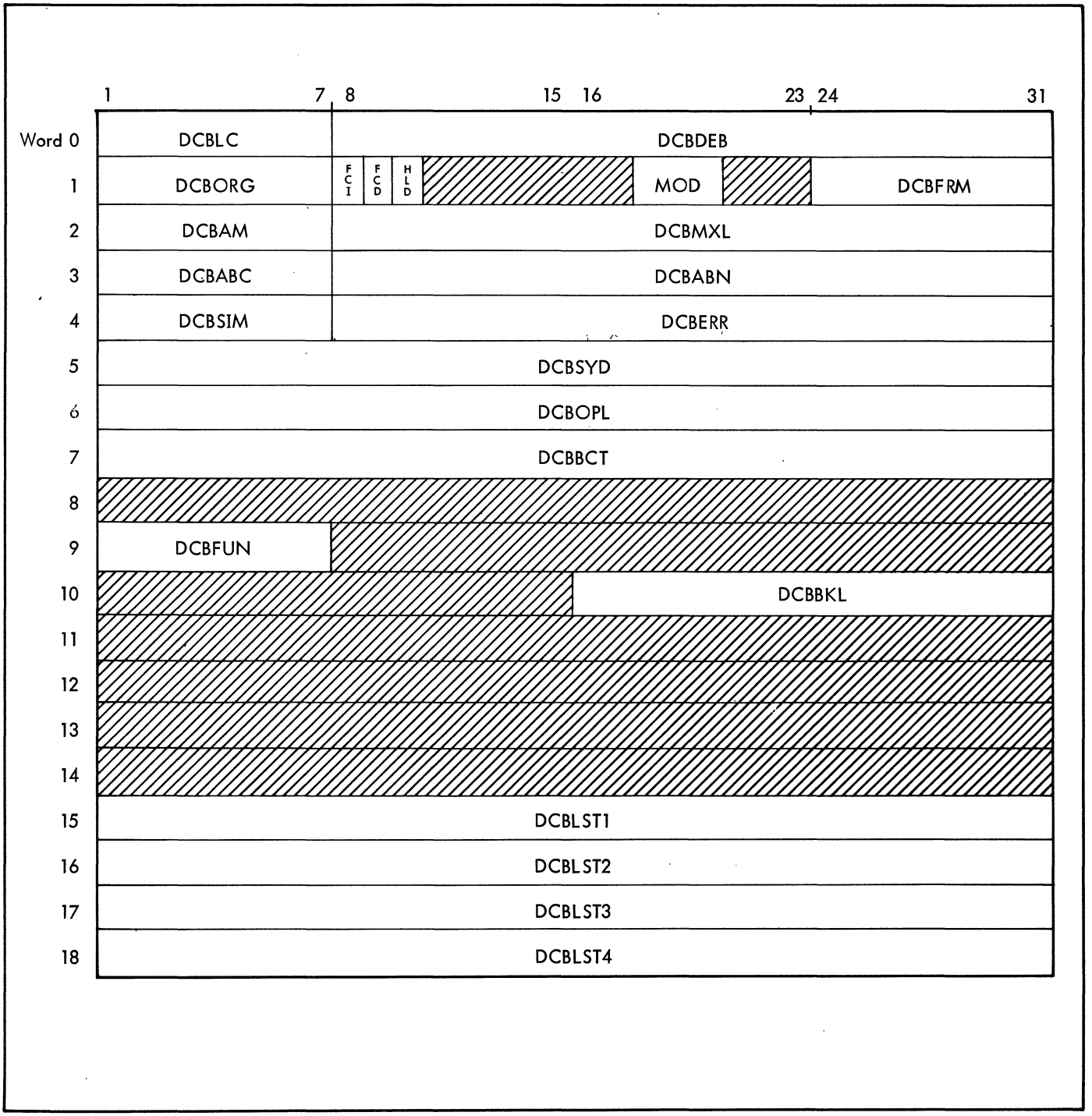


Figure E-2. TAM Data Control Block

Table E-2. DCB Fields Unique to TAM

Field Name	Length	Description
DCBLST1	1 word	Contains the virtual address of the first list named in M:DCB. (This list is placed in DCBLST by OPEN. It is the active list at initialization.)
DCBLST2	1 word	Contains the virtual addresses of any other lists associated to the DCB if there are any. Otherwise, these fields contain zeros.
DCBLST3	1 word	
DCBLST4	1 word	

Table E-3. Special TAM Values for Standard DCB Parameters

Field Name	Length	Description
DCBAM	1 byte	If VS has been coded it contains a 3 for virtual sequential; if BT has been coded it contains a 6 for TAM, before the DCB is opened. When the DCB is opened, VS (3) is changed to BT (6) if a transmission line is assigned.
DCBFRM	1 byte	Contains a zero for fixed record format. Zero is the default for this field also.
DCBMOD	3 bits	Indicates one of the following data formats: 0 – EBCDIC 1 – Binary 3 – ANSCII
DCBMXL	3 bytes	Depends on the transmission mode. In message mode, MXL indicates the maximum transfer length corresponding to the size of the buffers used in the read. MXL must be greater than or equal to the greatest transmission length of the components of the lines connected to that DCB. In character mode, MXL corresponds to the maximum size of the messages received from the terminals used. MXL must be less than or equal to the shortest length of print lines of the terminals connected to that DCB.
DCBORG	1 byte	Contains a value of zero which stands for sequential. Zero is the default for this field also.
DCBSIM	1 byte	Specifies the maximum number of simultaneous I/O operations for the group of lines managed by the DCB

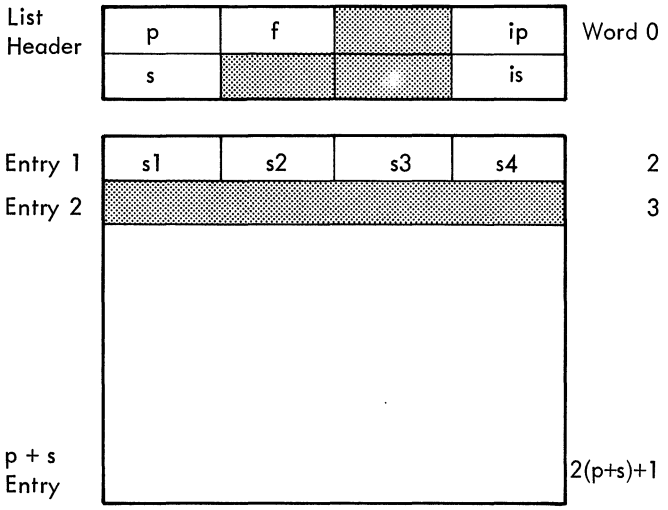
APPENDIX F. TAM LIST FORMATS

POLLING/SELECTION LISTS

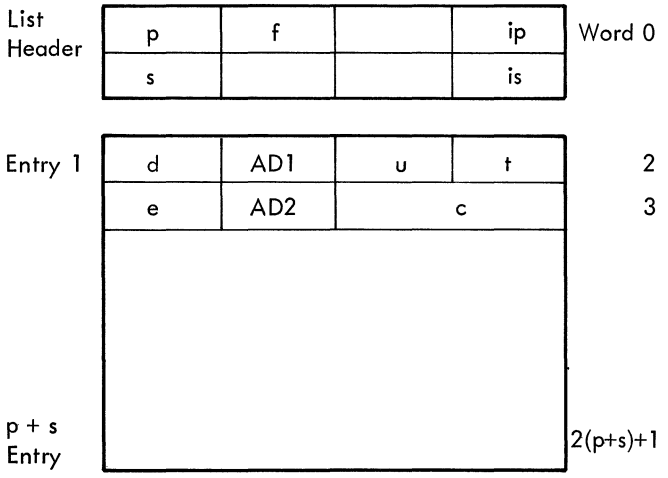
This list consists of a two-word header and two-word list entries. The first word of each entry contains either the component name (four EBCDIC characters, left-justified), of the word address of the user program containing the name of the component. The second word is not used before the DCB is opened. Assembly- and execution-time formats are shown below.

The p constitutes the polling list and is constructed as the first entry of the list. The s constitutes selection list entries.

Explicit List Format After Assembly



Explicit or Implicit List Format After Opening



Meaning of the parameters:

p is the number of components that can be polled. For the direct lists, this byte contains the number of stations of the list.

f is a list flag byte containing character/message mode indicators:

Loop/linear list.

Sequential/direct list.

Active/at-rest list.

List initialized at OPEN.

ip, is are bytes containing the immediate value of the polling index and the selection index, respectively.

s is the number of terminals in the selection list.

s1...s4 is the terminal name in EBCDIC characters, left-justified.

d, t, e, c are indexes in the system tables:

d index of the DEB subtable corresponding to the transmission line.

t terminal index (TML).

e index of the associated component in the polling or selection list.

c component index (CMP).

AD1 is the station address for polling/selection.

u is the status flag for the terminal.

AD2 is the component address for polling/selection.

DIRECT LISTS

This list consists of a header, one word in length, and terminal entries, one word in length. Word zero of each entry has the same role as the first word of the entries in the polling/selection lists.

APPENDIX G. EBCDIC 8-BIT COMPUTER CODES

		Most Significant Digits															
Least Significant Digits	Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0	0000	NUL	DLE	ds		SP	&	-								\	0
1	0001	SOH	DC1	ss				/		a	j			A	J		1
2	0010	STX	DC2	fs	SYN					b	k	s		B	K	S	2
3	0011	ETX	DC3	si						c	l	t		C	L	T	3
4	0100									d	m	u		D	M	U	4
5	0101	HT	LF							e	n	v		E	N	V	5
6	0110		BS	ETB						f	o	w		F	O	W	6
7	0111	DEL		ESC	EOT					g	p	x		G	P	X	7
8	1000	EOM	CAN							h	q	y		H	Q	Y	8
9	1001		EM				.			i	r	z		I	R	Z	9
A	1010					[†	] †		:								
B	1011	VT				.	$		# †								
C	1100	FF	FS		DC4	<	*	%	@								
D	1101	CR	GS	ENQ	NAK	(	)	_ †	'								
E	1110	SO	RS	ACK		+	;	>	=								
F	1111	SI	US	BEL	SUB	| †	¬ †	?	"								

†These graphic symbols have different representations depending on the teletypewriter models.

Notes:
1. The XOS EBCDIC character code set is SYSGEN-dependent; the code assignments shown in the table represent the SYSGEN default set.
2. The XOS EBCDIC character code set shown in the table differs significantly from the XDS standard set for several other operating systems, especially in the control-code assignments.

APPENDIX H. ANSCII 7-BIT COMMUNICATION CODES

		Most Significant Digits							
Least Significant Digits: Decimal (rows) (col's.) →		0	1	2	3	4	5	6	7
↓	Binary	x000	x001	x010	x011	x100	x101	x110	x111
0	0000	NUL	DLE	SP	0	@	P		p
1	0001	SOH	DC1	!†	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#†	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
10	1010	LF NL	SUB	*	:	J	Z	j	z
11	1011	VT	ESC	+	;	K	[†	k	
12	1100	FF	FS	,	<	L	\	l	
13	1101	CR	GS	-	=	M	]†	m	
14	1110	SO	RS	.	>	N	¬†	n	
15	1111	SI	US	/	?	O	_†	o	DEL

†These graphic symbols have different representations depending on the teletypewriter models.

INDEX

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

A

B

C

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

D

E

F

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

G

H

I

J

L

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

M

N

O

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

P

R

S

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

T

U

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

V

W

X

Xerox Data Systems

READER COMMENT FORM

XEROX®

We would appreciate your comments and suggestions for improving this publication.

Publication No.	Rev. Letter	Title	Current Date

How did you use this publication?

☐ Learning ☐ Installing ☐ Operating
☐ Reference ☐ Maintaining ☐ Sales

Is the material presented effectively?

☐ Fully covered ☐ Well illustrated
☐ Clear ☐ Well organized

What is your overall rating of this publication?

☐ Very good ☐ Fair ☐ Very poor
☐ Good ☐ Poor

What is your occupation?

Your other comments may be entered here. Please be specific and give page, column, and line number references where applicable. To report errors, please use the XDS Software Improvement or Difficulty Report (1188) instead of this form.

Thank you for your interest.

Fold and fasten as shown on back.
No postage needed if mailed in U.S.A.

Your name and return address.

STAPLE

STAPLE

FOLD

FIRST CLASS
PERMIT NO. 229
EL SEGUNDO, CALIF.

BUSINESS REPLY MAIL

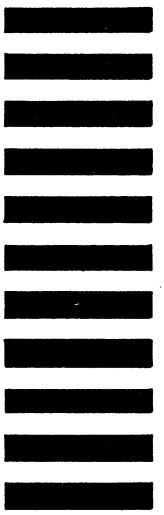
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

Xerox Data Systems

701 South Aviation Boulevard
El Segundo, California 90245

ATTN: PROGRAMMING PUBLICATIONS



CUT ALONG LINE

FOLD